interfaces graphiques
illustrées en Qt

Toolkit graphique Qt

Dessin interactif

Qt Designer

Statecharts

Threads

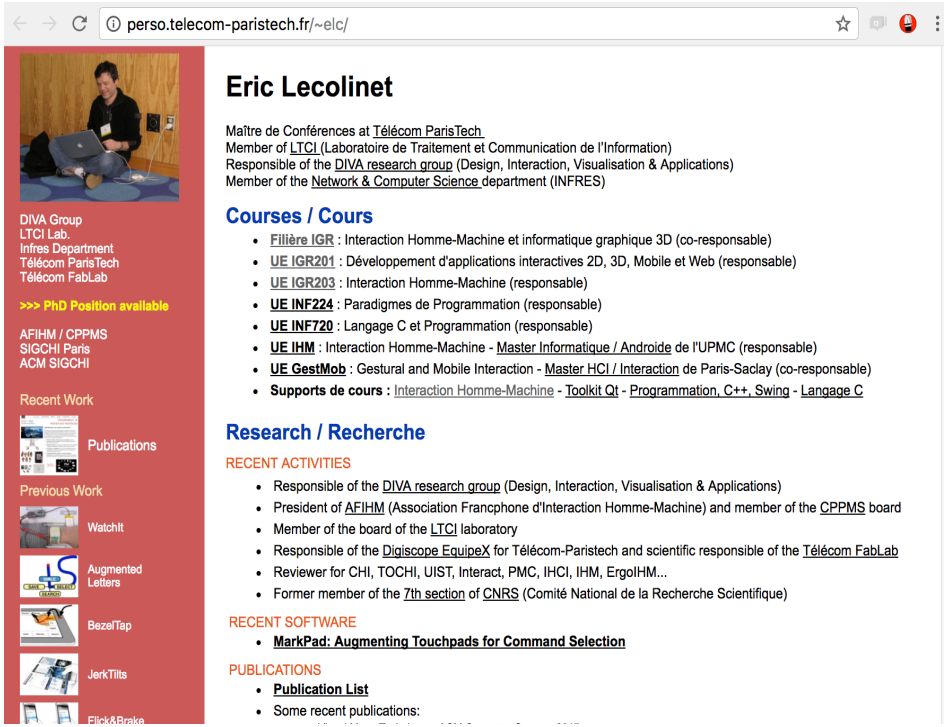
MVC

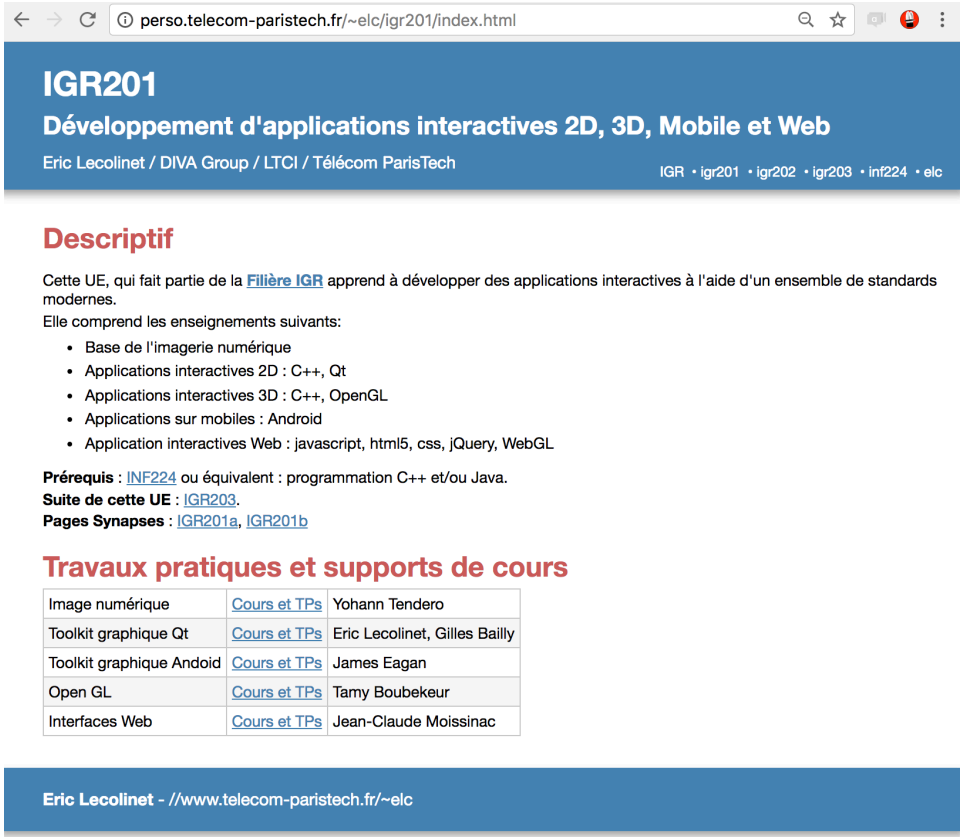
Eric Lecolinet - Télécom ParisTech

www.telecom-paristech.fr/~elc

Février 2018

Préambule : UE IGR201





http://www.telecom-paristech.fr/~elc/

http://www.telecom-paristech.fr/~elc/igr201/

Préambule : Filière IGR (2a + 3a)

perso.telecom-paristech.fr/~elc/igr/

Filière IGR

3D et Systèmes Interactifs

IGR • igr201 • igr202 • igr203 • inf224 • elc

Responsables : Tamy Boubekeur et Eric Lecolinet - Télécom ParisTech, Université Paris-Saclay
URL : http://www.telecom-paristech.fr/~elc/igr/

Descriptif

Cette filière vise à donner aux élèves une formation complète dans les domaines de l'interaction homme-machine et de l'informatique graphique 3D. Elle prépare les futurs ingénieurs à la conception de systèmes interactifs avancés en leur donnant les bases informatiques et mathématiques nécessaires à la modélisation numérique de ces systèmes.

Parmi les débouchés naturels de cette filière, on peut citer : la conception assistée par ordinateur (CAO), les jeux vidéo, les effets spéciaux, les applications mobiles, la simulation, le design d'interaction, la réalité virtuelle et la visualisation.

Cette filière prépare en outre aux métiers scientifiques liés à la recherche en IHM ou en informatique graphique 3D en proposant la possibilité de suivre ensuite un Master 2 spécialisé dans l'un de ces deux domaines (Master Interaction/HCI ou MVA de l'Université Paris-Saclay ou IMA de l'UPMC).

Voir aussi les **transparents de présentation de la filière**

Seconde année

- **Période 1 :** UE IGR201 : Développement d'applications interactives 2D, 3D, Mobile et Web
- **Période 2 :** UE IGR202 : Informatique Graphique 3D et Réalité Virtuelle
- **Période 3 :** UE IGR203 : Interaction Homme-Machine
- **Période 4 :** UE IGR204 : Visualisation (en anglais) et UE IGR205 : Séminaire de projet

Voir aussi la page Synapses de la Filière IGR et de ses UEs.

Troisième année

Trois possibilités :

- **1. Suivre l'Option interne IGR** (voir ci-dessous)
- **2. Candidater à un M2 :**
  - **M2 IMA** : Image, Spécialité de la mention Informatique (UPMC)
  - **M2 HCI** : Interaction, Human Computer Interaction (U. Paris Saclay), Mention Informatique
  - **M2 MVA** : Parcours Mathématiques, Vision et Apprentissage (U. Paris Saclay), Mention Mathématiques et Applications
- **3. Suivre une formation équivalente à l'étranger** :
  - Voir **cette page** et contacter James Eagan responsable mobilité internationale de la filière IGR

Il est également possible de choisir un cursus transverse (option entrepreneuriat) ou un des cursus alternatifs.

Voir aussi la page Masters M2 - UPSA et cohabilités.

Option interne IGR

Cette option de troisième année vise à compléter l'apprentissage dans les domaines de l'interaction et de l'informatique graphique 3D.

- **Prérequis :**
  - Seconde année de la Filière IGR ou connaissances équivalentes (ex : INF555 et INF584 à Polytechnique)
- **Premier semestre :**
  - **UE IG3DA/IMA904** : Informatique Graphique 3D Avancée (6 ECTS)
  - Modules d'IHM à choisir parmi **cette sélection dans le M2 HCI** de Paris-Saclay (6 ECTS ou plus)
  - **Projet PRIM** (12 ECTS)
    Effectué au cours du premier semestre, ce projet est à choisir dans la liste des PRIM de l'année en cours. Les PRIM en équipe de 4 à 6 personnes ainsi que les PRIM orientés Recherche sont encouragés. Certains PRIM seront proposés autour de la Fabrication numérique et de l'impression 3D.
- **Second semestre :**
  - Stage
- **Elèves "18 mois" :**
  - Les élèves "18 mois" issus de l'Ecole Polytechnique pourront faire un second PRIM au second semestre puis un stage terminal.

Voir aussi la page Synapses de l'Option Interne IGR et de ses UEs.

Equipe

- **Responsables de la filière :** Eric Lecolinet (bureau C201-5) et Tamy Boubekeur
- **Gestionnaire de scolarité :** Jérome Cahors
- **Responsable mobilité :** James Eagan
- **Responsable des stages :** Jean-Claude Moissinac
- **Responsables du séminaire de projet :** Gilles Bailly et Pooran Memari
- **Groupes de recherche associés :** HCI VIA Group et Computer Graphics Group

mail: nom.prenom@telecom-paristech.fr
adresse: Télécom ParisTech, 46 rue Barrault, 75013 Paris, Métro Corvisart

Eric Lecolinet - //www.telecom-paristech.fr/~elc

http://www.telecom-paristech.fr/~elc/igr/

comporte des informations sur la 3e année et les séjours à l'étranger !

Qt ("cute")

Un environnement complet

- Dont une **boîte à outils graphique** puissante et **multiplateformes**
- Mais aussi tout un ensemble de **bibliothèques** utilitaires et d'extensions
- Utilisé dans de nombreux **produits** libres (KDE…) ou propriétaires

Développement et licences

- Développé par TrollTech, puis Nokia, puis **Digia**
  - http://qt.digia.com
- Licences LGPL (**gratuite**) et **commerciales**

- Qt WebEngine WidgetsThe Qt WebEngine Widgets module provides a web browser engine as well as C++ classes to render and interact with web content.
- Qt WebKit Widgets The Qt WebKit Widgets module provides a web browser engine as well as C++ classes to render and interact with web content.
- Qt3DCore Qt3D Core module contains functionality to support near-realtime simulation systems
- Qt3DInput Qt3D Input module provides classes for handling user input in applications using Qt3D
- Qt3DRenderer Qt3D Renderer module contains functionality to support 2D and 3D rendering using Qt3D
- QtBluetooth Enables basic Bluetooth operations like scanning for devices and connecting them
- QtConcurrent Qt Concurrent module contains functionality to support concurrent execution of program code
- QtCore Provides core non-GUI functionality
- QtDBus Qt D-Bus module is a Unix-only library that you can use to perform Inter-Process Communication using the D-Bus protocol
- QtDesigner Provides classes to create your own custom widget plugins for Qt Designer and classes to access Qt Designer components
- QtGui Qt GUI module provides the basic enablers for graphical applications written with Qt
- QtHelp Provides classes for integrating online documentation in applications
- QtLocation Provides C++ interfaces to retrieve location and navigational information
- QtMultimedia Qt Multimedia module provides audio, video, radio and camera functionality
- QtNetwork Provides classes to make network programming easier and portable
- QtNfc An API for accessing NFC Forum Tags
- QtOpenGL Qt OpenGL module offers classes that make it easy to use OpenGL in Qt applications
- QtPositioning Positioning module provides positioning information via QML and C++ interfaces
- QtPrintSupport Qt PrintSupport module provides classes to make printing easier and portable
- QtQml C++ API provided by the Qt QML module
- QtQuick Qt Quick module provides classes for embedding Qt Quick in Qt/C++ applications
- QtQuickWidgets C++ API provided by the Qt Quick Widgets module
- QtScript Qt Script module provides classes for making Qt applications scriptable
- QtScriptTools Provides additional components for applications that use Qt Script
- QtSensors Provides classes for reading sensor data
- QtSerialPort List of C++ classes that enable access to a serial port
- QtSql Provides a driver layer, SQL API layer, and a user interface layer for SQL databases
- QtSvg Qt SVG module provides functionality for handling SVG images
- QtTest Provides classes for unit testing Qt applications and libraries
- QtUiTools Provides classes to handle forms created with Qt Designer
- QtWebChannel List of C++ classes that provide the Qt WebChannel functionality
- QtWebSockets List of C++ classes that enable WebSocket-based communication
- QtWidgets Qt Widgets module extends Qt GUI with C++ widget functionality
- QtXml Qt XML module provides C++ implementations of the SAX and DOM standards for XML
- QtXmlPatterns Qt XML Patterns module provides support for XPath, XQuery, XSLT and XML Schema validat

Toolkit graphique Qt

Boîte à outils graphique

- en **C++** à la base
- mais également extension **Qt Quick** basée sur le langage déclaratif **QML**

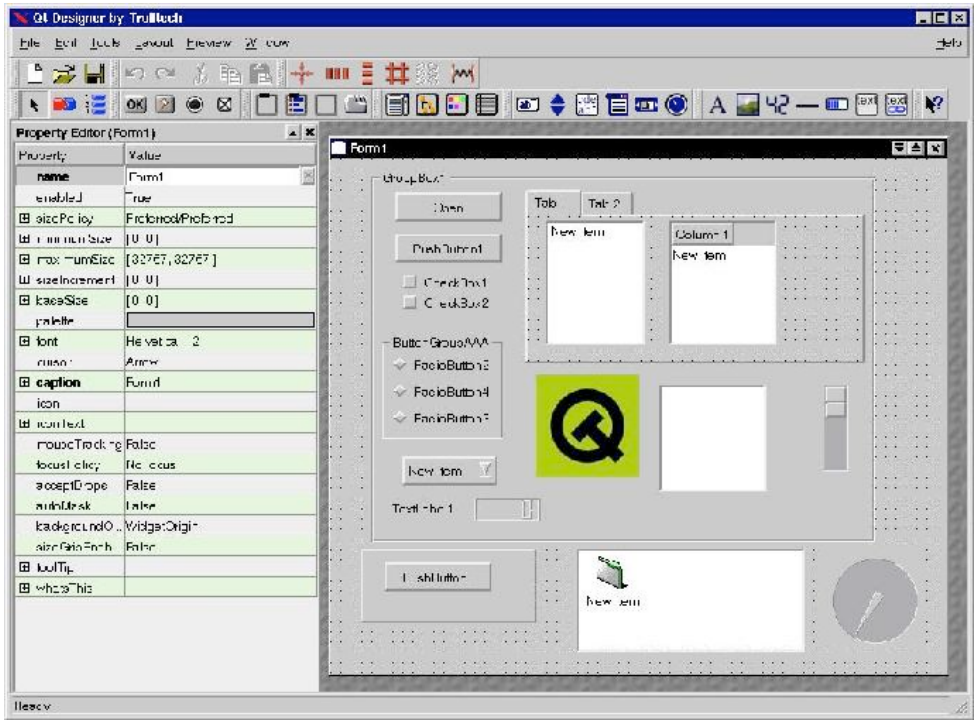
Multi-plateformes

- principaux OSs : Windows, Mac OS X, Linux/X11
- et plate-formes mobiles : iOS, Android, WinRT

Outils et facilités

Outils de développement

- **QtCreator** / QtDesigner
- qmake
- Qt Linguist
- Qt Assistant
- etc.



Utilitaires

- internationalisation:
  - QString et **Unicode**
- sockets, XML, SQL, outils Web, OpenGL, etc.

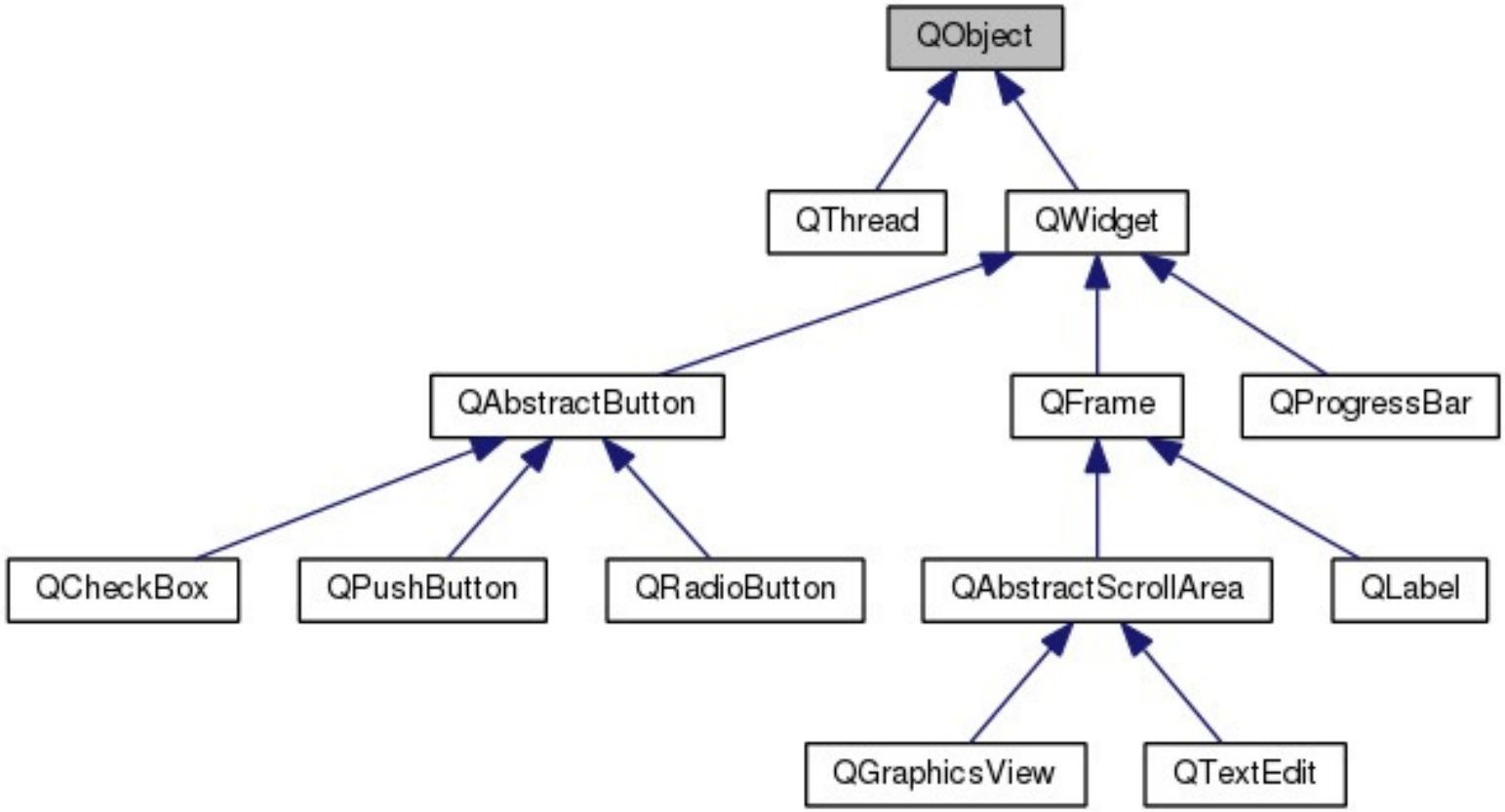
Liens

Liens utiles

- **Site Télécom** : http://www.telecom-paristech.fr/~elc/qt
- **Site général Qt** : http://www.qt.io/
- **Toolkit Qt :**
  - Page principale : http://doc.qt.io/qt-5/
  - Toutes les classes (par modules) : http://doc.qt.io/qt-5/modules-cpp.html
- **Qt Widgets :**
  - Documentation : http://doc.qt.io/qt-5/qtwidgets-index.html
  - Liste des classes : http://doc.qt.io/qt-5/qtwidgets-module.html

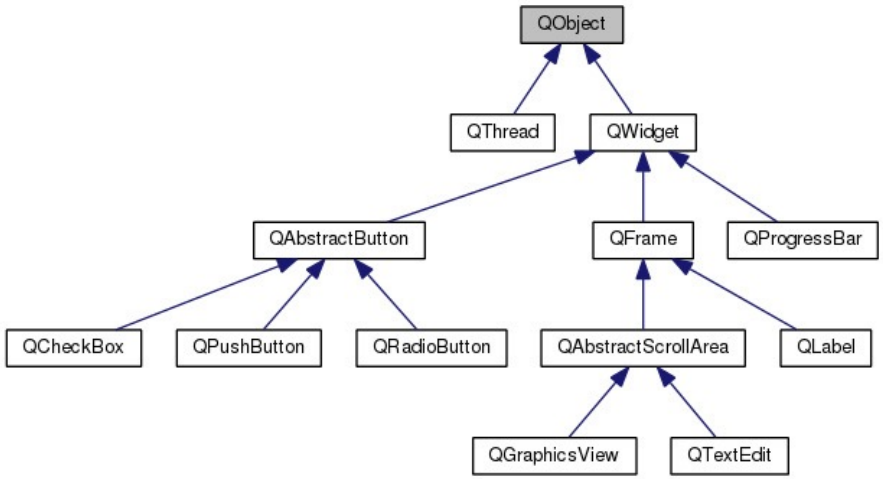
Principaux widgets

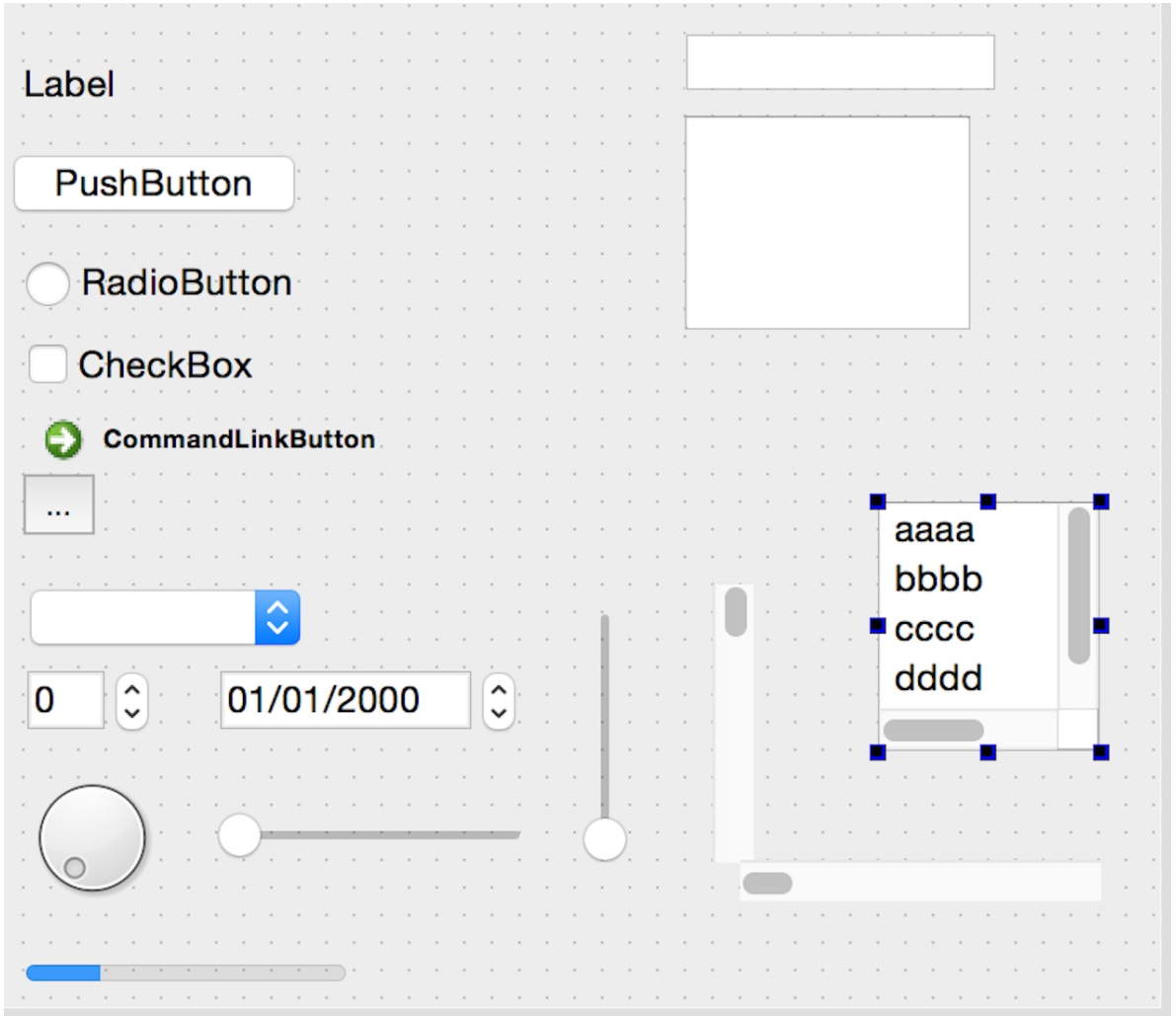
Arbre d'héritage (très partiel)



Principaux widgets

Interacteurs



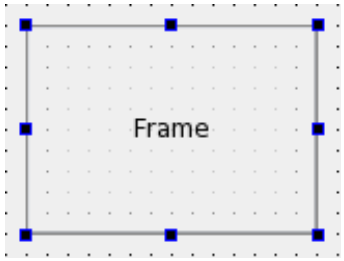


Principaux widgets

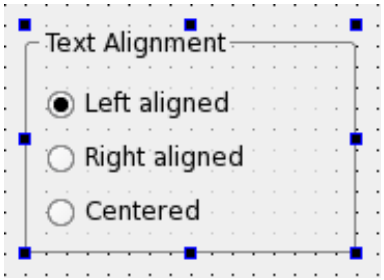
Conteneurs

QWidget

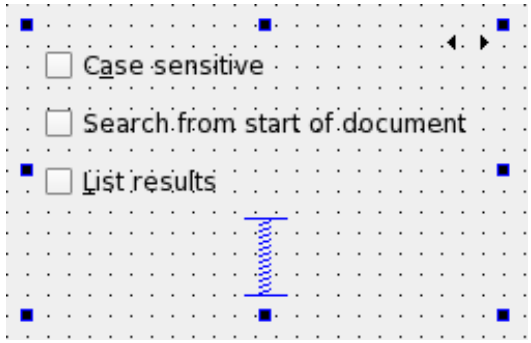
QDockWidget



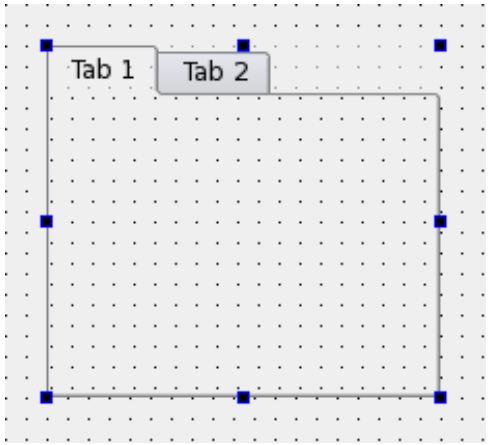
QFrame



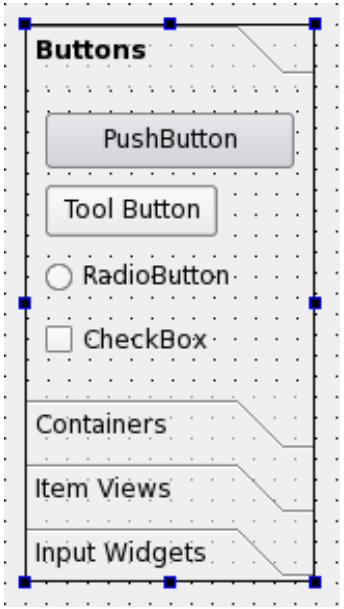
QGroupBox



QStackedWidget



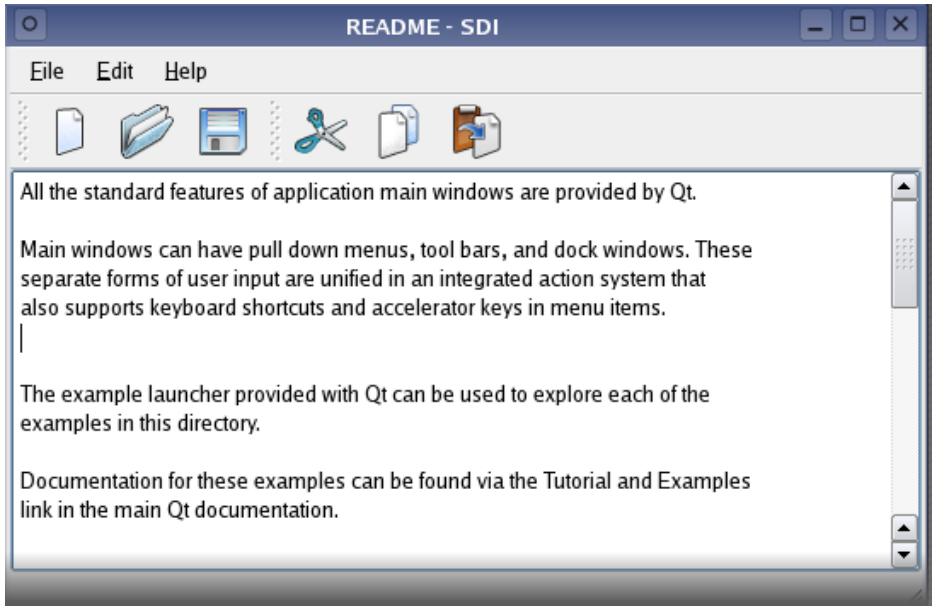
QTabWidget



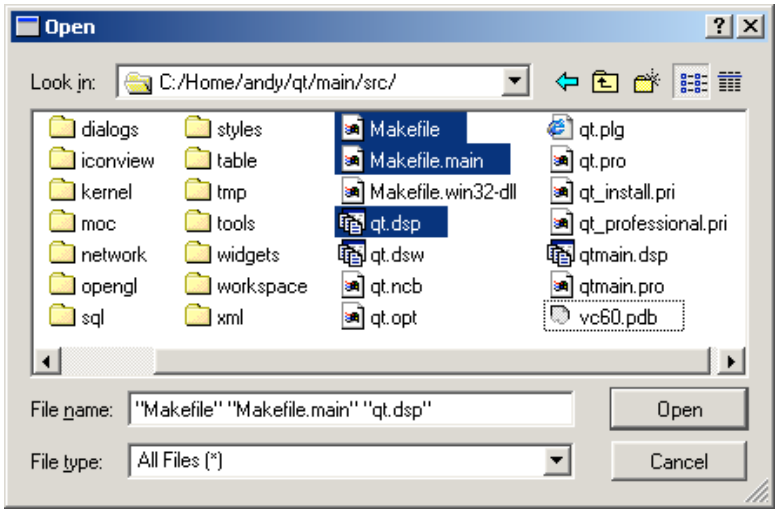
QToolbox

Principaux widgets

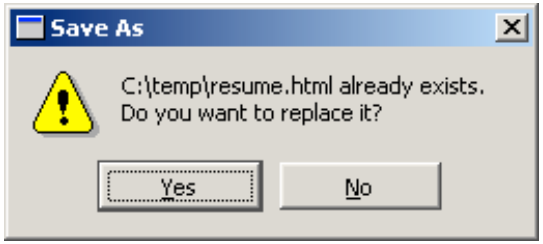
Fenêtre principale, boîtes de dialogue, menus



QMainWidget



QFileDialog



QMessageBox

Hello Word!

```
#include <QApplication>
#include <QLabel>

int main( int argc, char **argv ) {
    QApplication * app = new QApplication(argc, argv);
    QLabel * hello = new QLabel("Hello Qt!");
    hello->show( );
    return app->exec( );
}
```

- **pointeurs** C++ : attention aux **->** et aux *
- hello->**show**( ) => le **QLabel** devient le widget **principal**
- **exec**( ) lance la boucle de gestion des **événements**

Variante (objets dans la pile)

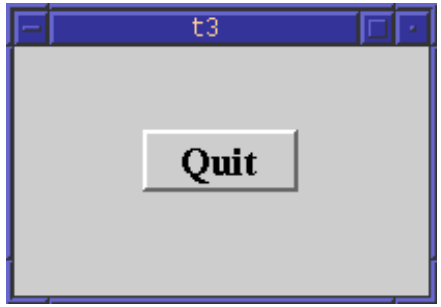
```
#include <QApplication>
#include <QLabel>

int main( int argc, char **argv ) {
    QApplication app(argc, argv);
    QLabel hello("Hello Qt!");
    hello.show( );
    return app.exec( );
}
```

la variable **contient** l'objet => . pour accéder aux champs et non ->

attention : objets créés dans la **pile** => **détruits** en fin de fonction

Avec un conteneur

```
#include <QApplication>       // inclure headers adéquats !
#include <QPushButton>
#include <QWidget>
#include <QFont>

int main( int argc, char **argv ) {
    QApplication * app = new QApplication(argc, argv);

    QWidget * box = new QWidget();
    box->resize(200, 120);

    QPushButton * quitBtn = new QPushButton("Quit", box);      // box sera le parent de quitBtn
    quitBtn->resize(100, 50);
    quitBtn->move(50, 35);
    quitBtn->setFont( QFont("Times", 18, QFont::Bold) );

    box->show( );
    return app->exec( );
}
```

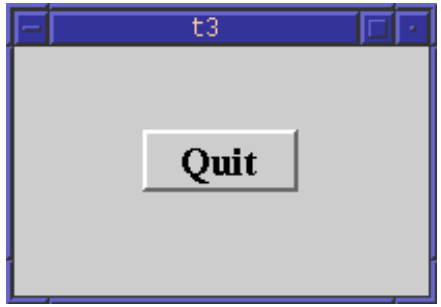


Le **parent** est passé en argument du **constructeur**

pas d'argument pour les "**top-level**" widgets

Signaux et slots

```
#include <QApplication>
#include <QPushButton>
#include <QWidget>
#include <QFont>

int main( int argc, char **argv ) {
    QApplication * app = new QApplication(argc, argv);

    QWidget * box = new QWidget();
    box->resize(200, 120);

    QPushButton * quitBtn = new QPushButton("Quit", box);
    quitBtn->resize(100, 50);
    quitBtn->move(50, 35);
    quitBtn->setFont( QFont("Times", 18, QFont::Bold) );

    QObject::connect( quitBtn, SIGNAL(clicked( )), app, SLOT(quit( )) );      // connexion

    box->show( );
    return app->exec( );
}
```



Signaux et slots

Un signal est émis vers le monde extérieur

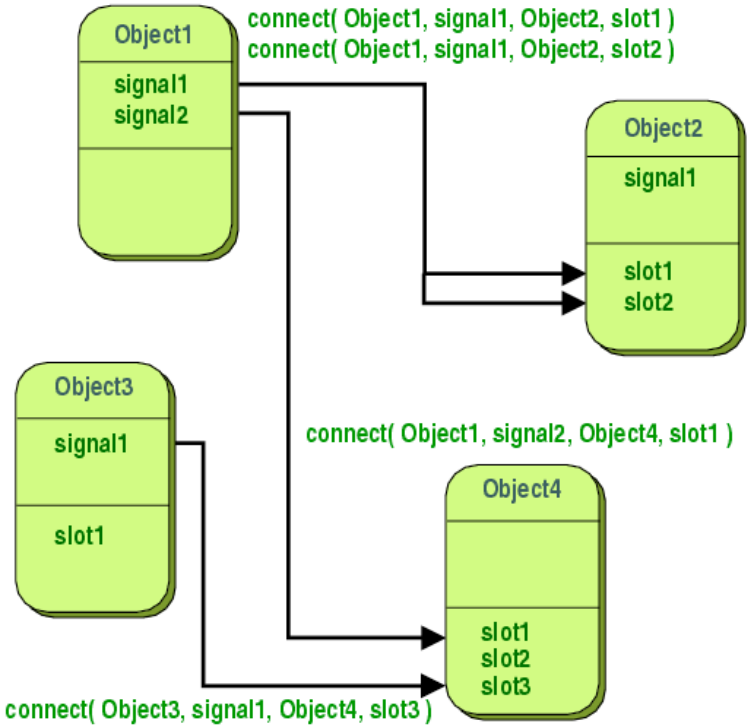
- par un objet quand il change d'état
- on n'indique pas à qui il s'adresse

Un slot est un récepteur

- en pratique c'est une méthode

Les signaux sont connectés à des slots

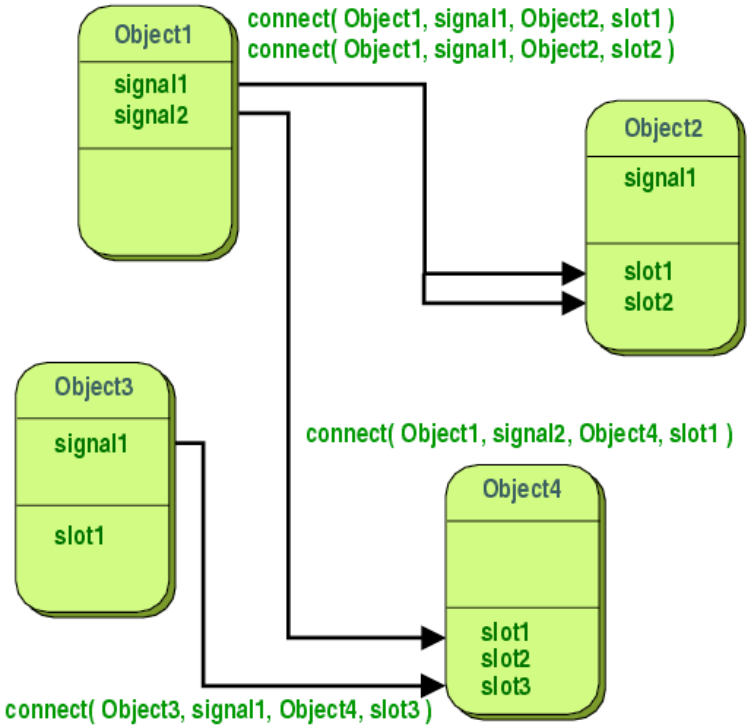
- les slots sont alors appelés automatiquement



Signaux et slots

Modularité, flexibilité

- le **même** **signal** peut être connecté à **plusieurs** **slots**
- **plusieurs** **signaux** peuvent être connectés à un **même** **slot**



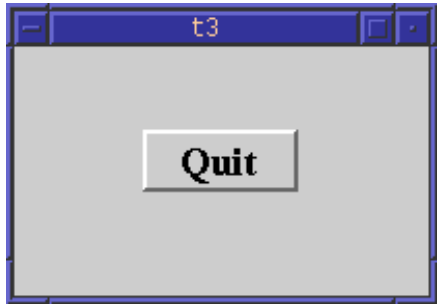
Remarques

- **l'émetteur** ne **connaît pas** le(s) **récepteur**(s)
- il ne sait pas si le signal est **reçu**
- le récepteur ne **connaît pas** l'émetteur

➜ Programmation **modulaire** par composants

Signaux et slots

```
#include <QApplication>
#include <QPushButton>
#include <QWidget>
#include <QFont>

int main( int argc, char **argv ) {
    QApplication * app = new QApplication(argc, argv);

    QWidget * box = new QWidget();
    box->resize(200, 120);

    QPushButton * quitBtn = new QPushButton("Quit", box);
    quitBtn->resize(100, 50);
    quitBtn->move(50, 35);
    quitBtn->setFont( QFont("Times", 18, QFont::Bold) );

    QObject::connect( quitBtn, SIGNAL(clicked( )), app, SLOT(quit( )) );      // connexion

    box->show( );
    return app->exec( );
}
```



Connexion

Règles

- les **signaux** et les **slots** peuvent avoir des **paramètres**
- leurs **types** doivent être les **mêmes**
- un **slot** peut avoir **moins** de paramètres qu'un **signal**

```
QObject::connect( quitBtn, SIGNAL(clicked( )), app, SLOT(quit( )) );
QObject::connect( x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)) );
QObject::connect( x, SIGNAL(balanceChanged(int)), app, SLOT(quit()) );
```

Aspect central de Qt

- diffère de l'habituel mécanisme des **callbacks** ou **listeners**

Inconvénient

- **SLOT** et **SIGNAL** sont des macros, nécessite une **pré-compilation**

Déclaration de slot

Dans le fichier header (.h)

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int curBalance;
public:
    BankAccount( ) { curBalance = 0; }
    int getBalance( ) const { return curBalance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

- sous classe de **QObject**
- mot-clés **Q_OBJECT**, **slots** et **signals** pour le **pré-compilateur**
- **signals** **pas** implémentés, **slots** **doivent** être implémentés

Définition de slot

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int curBalance;
public:
    BankAccount( ) { curBalance = 0; }
    int getBalance( ) const { return curBalance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

Dans le fichier d'implémentation (.cpp)

```
void BankAccount::setBalance(int newBalance)
{
    if (newBalance != curBalance) {
        curBalance = newBalance;
        emit balanceChanged(curBalance);
    }
}
```

- **emit** provoque l'**émission :**
  - du signal **balanceChanged**()
  - avec la nouvelle valeur de **curBalance**

Connexion

Connexion simple

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int curBalance;
public:
    BankAccount( ) { curBalance = 0; }
    int getBalance( ) const { return curBalance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

```
BankAccount * x = new BankAccount;
BankAccount * y = new BankAccount;
connect(x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)));
x->setBalance(10);
```

```
void BankAccount::setBalance(int newBalance)
{
    if (newBalance != curBalance) {
        curBalance = newBalance;
        emit balanceChanged(curBalance);
    }
}
```

Connexion

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int curBalance;
public:
    BankAccount( ) { curBalance = 0; }
    int getBalance( ) const { return curBalance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

Connexion simple

```
BankAccount * x = new BankAccount;
BankAccount * y = new BankAccount;
connect(x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)));
x->setBalance(10);
```

- **x** est mis à 10 ➜ le signal **balanceChanged**() est émis
- il est reçu par le slot **setBalance**() de **y** ➜ **y** est mis à 10

Connexion

Connexion dans les deux sens

```
connect(x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)));
connect(y, SIGNAL(balanceChanged(int)), x, SLOT(setBalance(int)));
x->setBalance(10);
```

- **OK ?**

```
void BankAccount::setBalance(int newBalance)
{
    if (newBalance != curBalance) {
        curBalance = newBalance;
        emit balanceChanged(curBalance);
    }
}
```

Connexion

Connexion dans les deux sens

```
connect(x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)));
connect(y, SIGNAL(balanceChanged(int)), x, SLOT(setBalance(int)));
x->setBalance(10);
```

- OK : car **test** dans **setBalance()**

vérifier que la valeur a **changé**

pour éviter les **boucles infinies !**

```
void BankAccount::setBalance(int newBalance)
{
    if (newBalance != curBalance) {
        curBalance = newBalance;
        emit balanceChanged(curBalance);
    }
}
```

Connexion

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int curBalance;
public:
    BankAccount( ) { curBalance = 0; }
    int getBalance( ) const { return curBalance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

Nouvelle syntaxe :
Pointeurs de méthodes

```
connect(x, &BankAccount::balanceChanged, y, &BankAccount::setBalance);
```

- utilise les **pointeurs de méthodes** de **C++**
- **avantage** : vérifie la **validité** de la connexion à la **compilation**
  - avec l'ancienne syntaxe c'est fait à **l'exécution**
- **inconvénient** : pas compatible avec **QtQuick**

Connexion

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int curBalance;
public:
    BankAccount( ) { curBalance = 0; }
    int getBalance( ) const { return curBalance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

Nouvelle syntaxe :

Lambdas de C++11

```
int value = 10;
connect(x, &BankAccount::balanceChanged,  [ = ] {setBalance (value);} );
```

[=] {setBalance (value);} est une **lambda**

- permet d'appeler la fonction avec les **arguments que l'on veut** (ici **toujours** 10)
- le **=** signifie que la **lambda** peut **capturer** les variables de la fonction où elle se trouve

Connexion

```
class Controller : public QObject {
    Q_OBJECT
signals:
    void changed(int);
    ....
};
```

Propagation de signal

```
connect(x, SIGNAL(balanceChanged(int)), controller, SIGNAL(changed(int)) );
```

Déconnexion

```
disconnect(x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)) );
```

Compilation

Meta Object Compiler (MOC)

- **pré-processeur** de Qt
- génère du **code supplémentaire** (tables de signaux / slots)
- **attention**: ne pas oublier le mot-clé **Q_OBJECT**

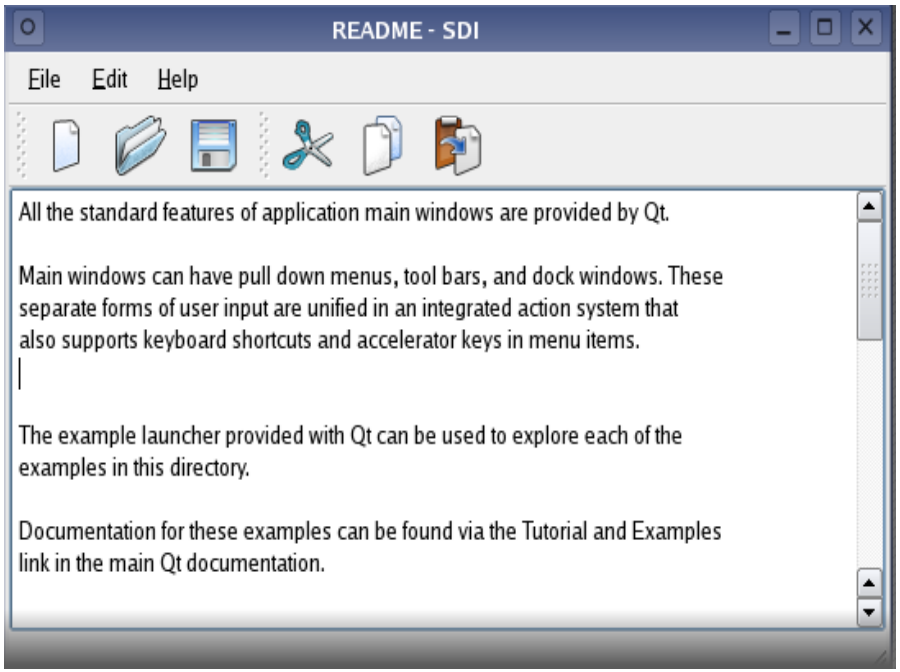
Commande qmake

- dans le répertoire contenant les fichiers sources faire:
  - **qmake** -project // crée le fichier **xxx.pro** (décrit le projet)
  - **qmake** // crée le fichier **Makefile** (ou équivalent si IDE)
  - **make** // crée les fichiers **.moc** (un par fichier ayant des slots),
    // et les fichiers **binaires** (*.o et exécutable)

Fenêtre principale (QMainWindow)

Zones prédéfinies pour

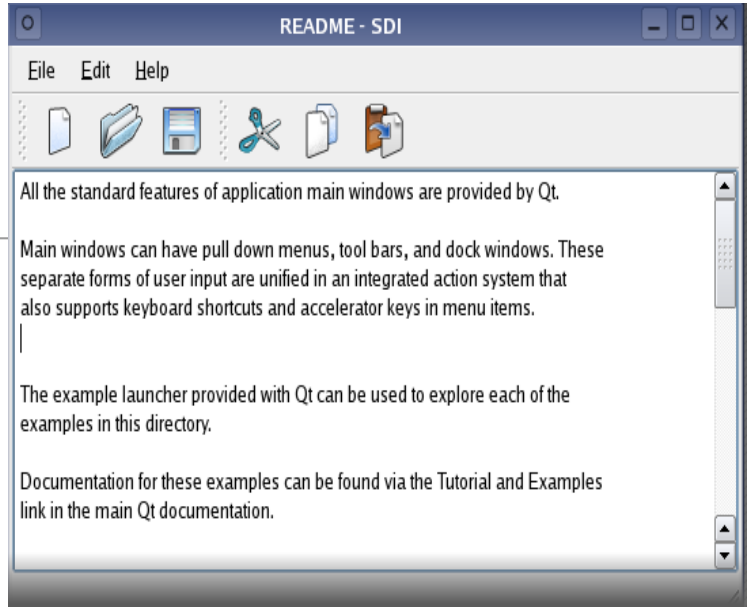
- barre de **menu**
- barre **d'outils**
- barre de **statut**
- **zone centrale**
- (et d'autres fonctionnalités…)



Utilisation

- créer une **sous-classe** de **QMainWindow**
- son **constructeur** lui ajoute les objets graphiques

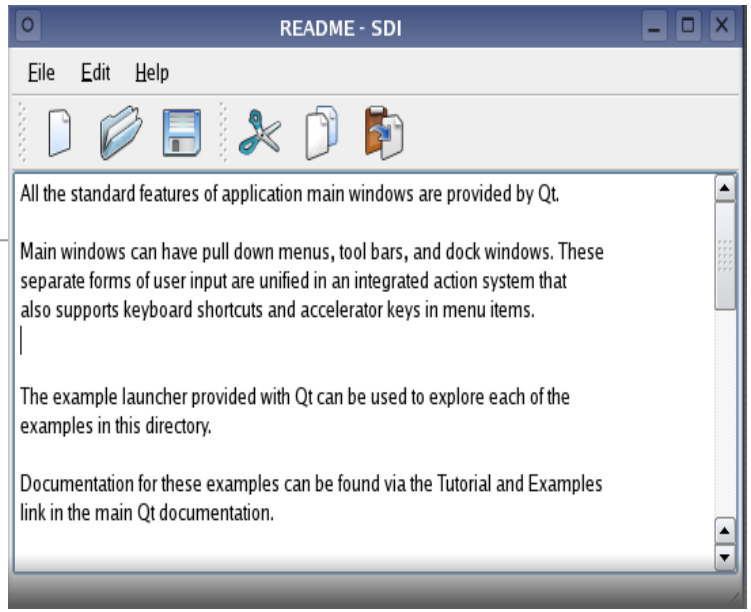
Fenêtre principale



Dans le constructeur d'une classe dérivant de QMainWindow

```
// menuBar() est une method de QMainWindow
QMenuBar * menuBar = this->menuBar( );
QMenu * fileMenu = menuBar->addMenu( tr ("&File") );

// new.png est un fichier qui sera spécifié dans un fichier de ressources .qrc
QAction * newAction = new QAction( QIcon(":new.png"), tr("&New..."), this);

newAction->setShortcut( tr("Ctrl+N"));          // accélérateur clavier
newAction->setToolTip( tr("New file"));         // bulle d'aide
newAction->setStatusTip( tr("New file"));       // barre de statut
fileMenu->addAction(newAction);                 // rajouter l'action au menu déroulant

connect(newAction, SIGNAL(triggered( )), this, SLOT(open( )));      // connexion
```

Fenêtre principale



Actions

- les actions peuvent s'ajouter à la fois dans les **menus** et les **toolbars**

```
QToolBar * toolBar = this->addToolBar( tr("File") );
toolBar->addAction(newAction);
```

Zone centrale

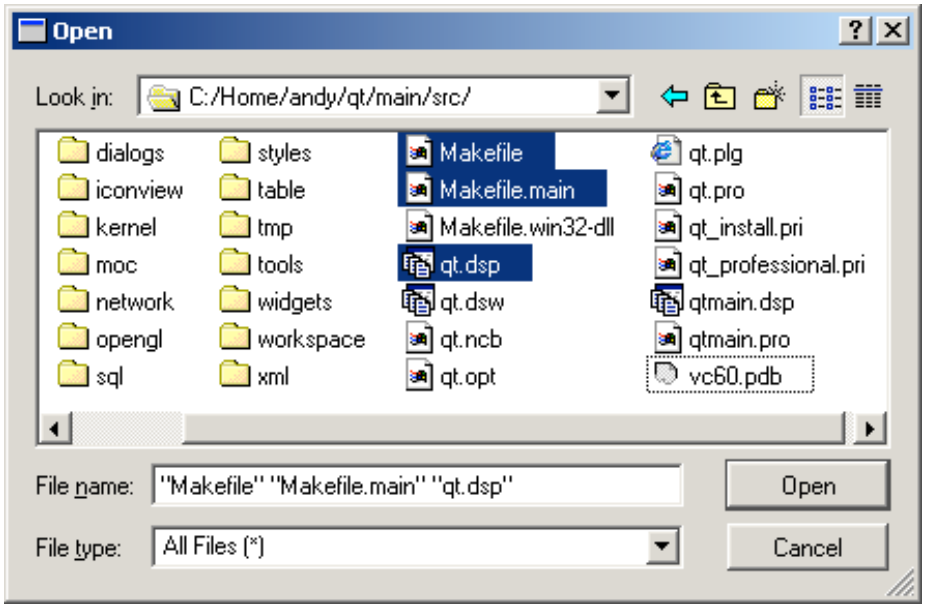
```
QTextEdit * text = new QTextEdit(this);
setCentralWidget(text);
```

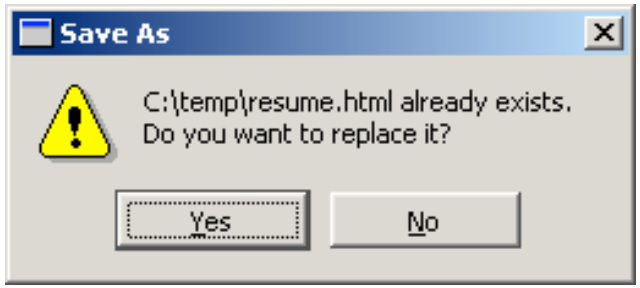
Traduction: tr( )

- **localisation** : permet de **traduire** le texte

Boîtes de dialogue

QFileDialog, QMessageBox

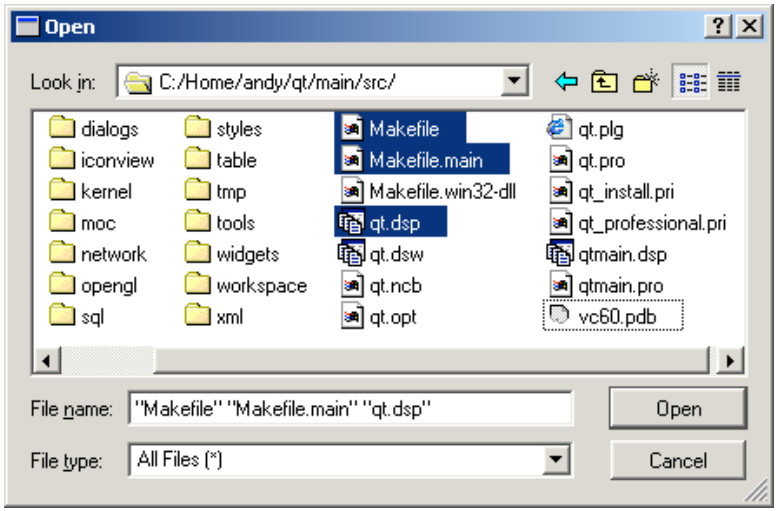




Boîte de dialogue **modale**

Solution générale

```
QFileDialog dialog (parent);
dialog.setFilter("Text files (*.txt)");
QStringList fileNames;

if (dialog.exec() == QDialog::Accepted) {
    fileNames = dialog.selectedFiles();
    QString firstName = fileNames[0];
    ...
}
```



dialog.exec() lance une boucle de gestion des événements **secondaire**

Solution simplifiée

```
QString fileName =
    QFileDialog::getOpenFileName( this,
                                  tr("Open Image"),                    // titre
                                  "/home/jana",                        // répertoire initial
                                  tr("Image Files (*.png *.jpg *.bmp)") // filtre
                                  );
```

QString, QFile, QTextStream

QString : Codage Unicode 16 bits

- **Suite de QChars**
  - 1 caractère = 1 **QChar** de 16 bits (cas usuel)
  - 1 caractère = 2 **QChars** de 16 bits (pour valeurs > 65535)

- **Conversions d'une QString :**
  - **toAscii**( ) : **ASCII** 8 bits
  - **toLatin1**( ) : **Latin-1** (ISO 8859-1) 8 bits
  - **toUtf8**( ) : **UTF-8** Unicode multibyte (1 caractère = 1 à 4 octets)
  - **toLocal8Bit**( ) : codage local 8 bits
  - **qPrintable**(const QString & str) équivalent à : str.**toLocal8Bit**( ).**constData**( )

QFile

- **lecture** et **écriture** de fichiers
- **exemple** :

```
QFile file( fileName );
if ( file.open( QIODevice::ReadOnly | QIODevice::Text) )  { etc.... }
if ( file.open( QIODevice::WriteOnly ) )  { etc.... }
```

QTextStream

- pour **lire** ou **écrire** du texte depuis un **QFile** :

```
QTextStream stream( &file );
```

- compatible avec **QString**

- supporte les opérateurs **<<** et **>>** :

```
output_stream << arg;
input_stream >> arg;
```

```
QFile file("output.txt");
if (file.open(QFile::WriteOnly)) {
    QTextStream out(&data);
    out << "Result: " << 10 << left << 3.14 << 2.7;
}
```

QTextStream

- lit un **mot** :

```
input_stream >> arg;          // s'arrête au premier espace si arg est une string
```

- lit une **ligne** :

```
QString readLine( taillemax = 0 );          // pas de limite de taille si = 0
```

- lit **tout** le fichier :

```
QString readAll( );          // à n'utiliser que pour de petits fichiers
```

- codecs :
  - **setCodec**(codec), **setAutoDetectUnicode**(bool);

qDebug

Afficher des messages

```
int val;
QString s;
qDebug() << "value: "  <<  s << " / " << val;

// ou encore
std::cout << "value: "  <<  qPrintable(s) << " / " << val << std::endl;
```

Ressources

Dans le programme

```
QAction * myaction = new QAction (QIcon(":images/new.png"), tr("&New..."), this);
```

- noter le **:**
  - signifie : chemin **relatif à l'application**
    - ici : le fichier "new.png" dans le sous-répertoire "images"
  - **attention** : le "**prefix**" doit être **/**
    - dans **QtCreator** (et dans le fichier .**qrc**)

- **tr()** pour **traduire** :

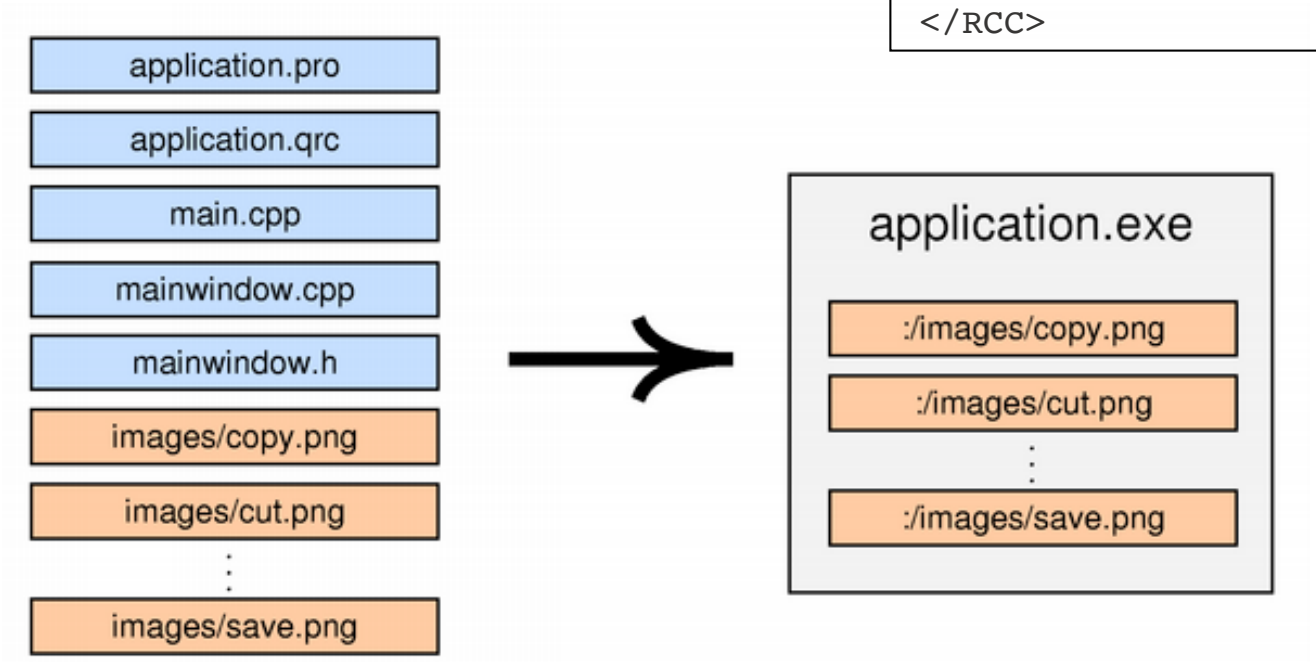
```
tr("&New...")
myaction->setShortcut(tr("Ctrl+N"));
```

Ressources

Fichier .qrc

- créé à la main ou par **QtCreator**
- attention au **prefix** !

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
    <file>images/copy.png</file>
    <file>images/cut.png</file>
    <file>images/new.png</file>
    <file>images/open.png</file>
    <file>images/paste.png</file>
    <file>images/save.png</file>
</qresource>
</RCC>
```



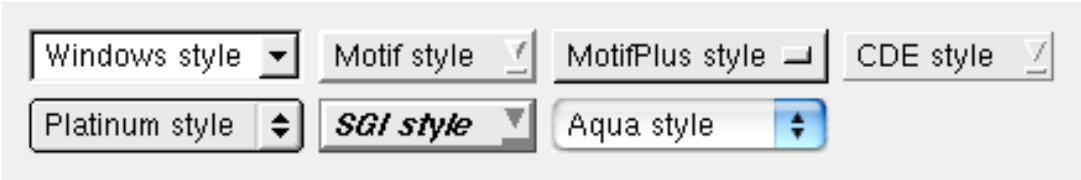
Styles

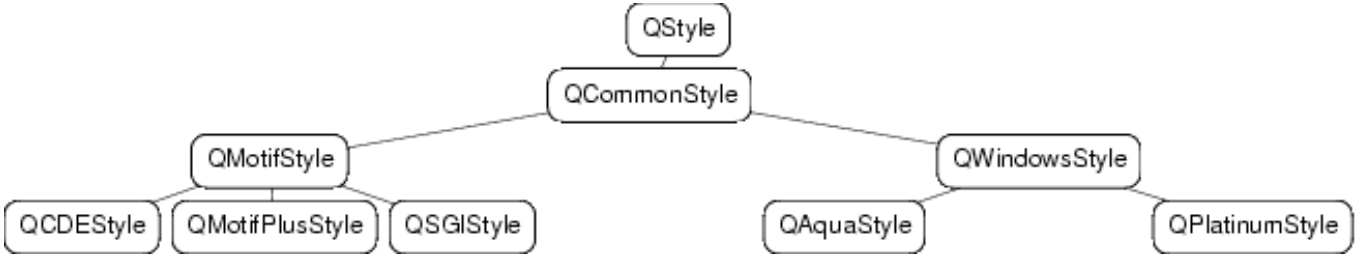
Emulation du "Look and Feel"

- Look and feel simulé et paramétrable (comme Swing)
- rapidité, flexibilité, extensibilité,
- pas restreint à un "dénominateur commun »

QStyle

QApplication::**setStyle**(new MyCustomStyle);

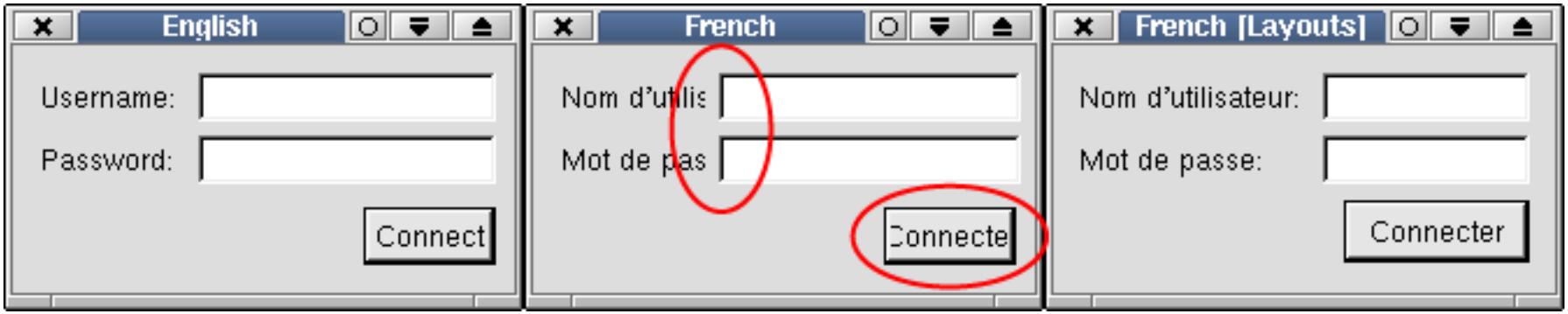




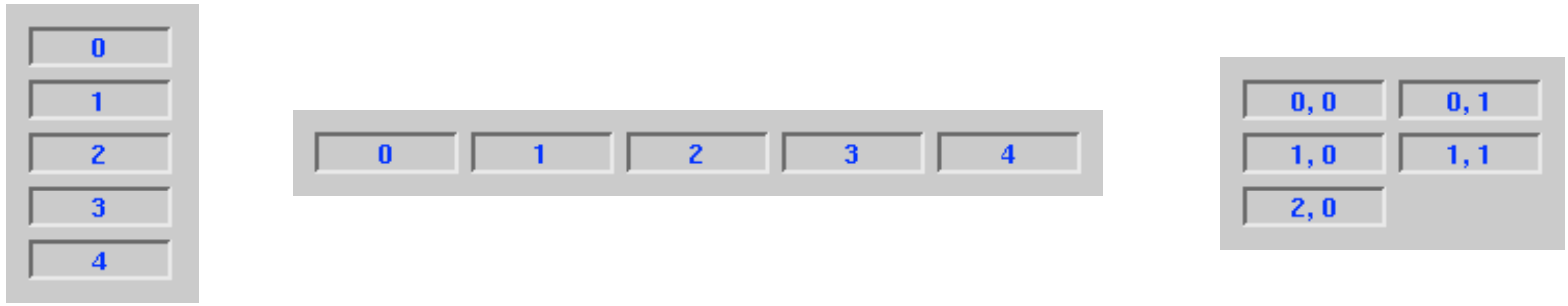
Layout

Buts

- **internationalisation**
- **retailler** interactivement
- éviter d'avoir à faire des calculs de taille compliqués

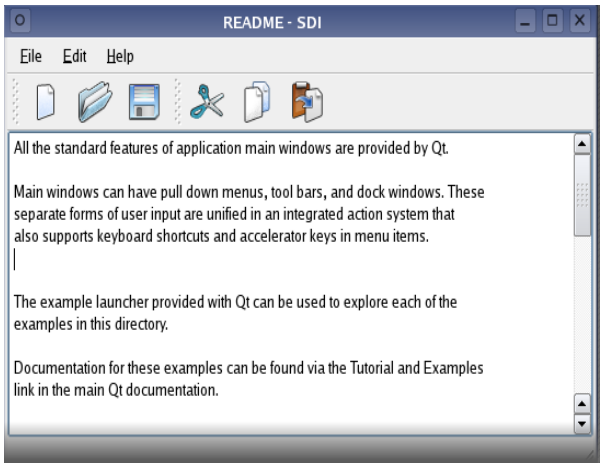


Layout



QHBoxLayout, QVBoxLayout, QGridLayout

QFormLayout



QMainWindow

Layout : exemple

```
QVBoxLayout * v_layout = new QVBoxLayout( );
v_layout->addWidget( new QPushButton( "OK" ) );
v_layout->addWidget( new QPushButton( "Cancel" ) );
v_layout->addStretch( );
v_layout->addWidget( new QPushButton( "Help" ) );
```

Les Layouts

- peuvent être **emboîtés**
- ne sont **pas** liés à une **hiérarchie** de conteneurs comme en **Java**
- cf. le « **stretch** »

Layout : exemple

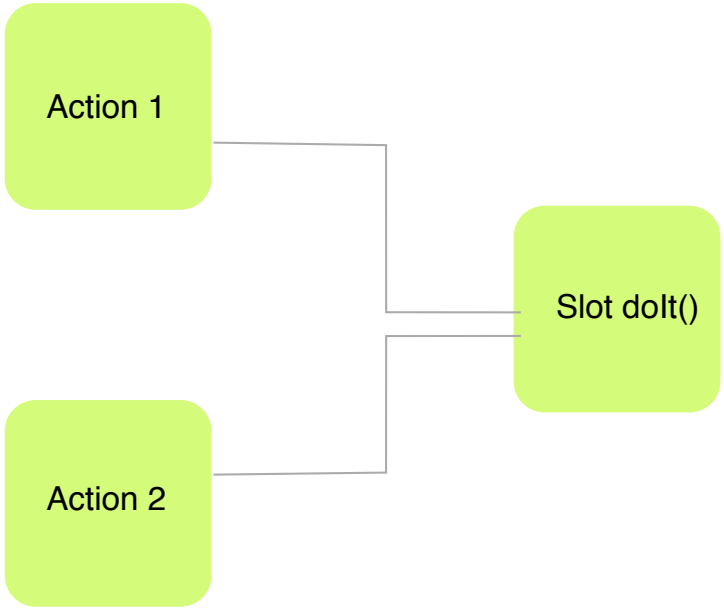
```
QVBoxLayout * v_layout = new QVBoxLayout( );
v_layout->addWidget( new QPushButton( "OK" ) );
v_layout->addWidget( new QPushButton( "Cancel" ) );
v_layout->addStretch( );
v_layout->addWidget( new QPushButton( "Help" ) );

QListBox * country_list = new QListBox( this );
countryList->insertItem( "Canada" );
...etc...

QHBoxLayout * h_layout = new QHBoxLayout( );
h_layout->addWidget( country_list );
h_layout->addLayout( v_layout );
```

Layout : exemple

```
QVBoxLayout * v_layout = new QVBoxLayout( );
v_layout->addWidget( new QPushButton( "OK" ) );
v_layout->addWidget( new QPushButton( "Cancel" ) );
v_layout->addStretch( );
v_layout->addWidget( new QPushButton( "Help" ) );

QListBox * country_list = new QListBox( this );
countryList->insertItem( "Canada" );
...etc...

QHBoxLayout * h_layout = new QHBoxLayout( );
h_layout->addWidget( country_list );
h_layout->addLayout( v_layout );

QVBoxLayout * top_layout = new QVBoxLayout( );
top_layout->addWidget( new QLabel( "Select a country", this ) );
top_layout->addLayout( h_layout );

window->setLayout( top_layout );
window->show( );
```

Retour sur les signaux et les slots

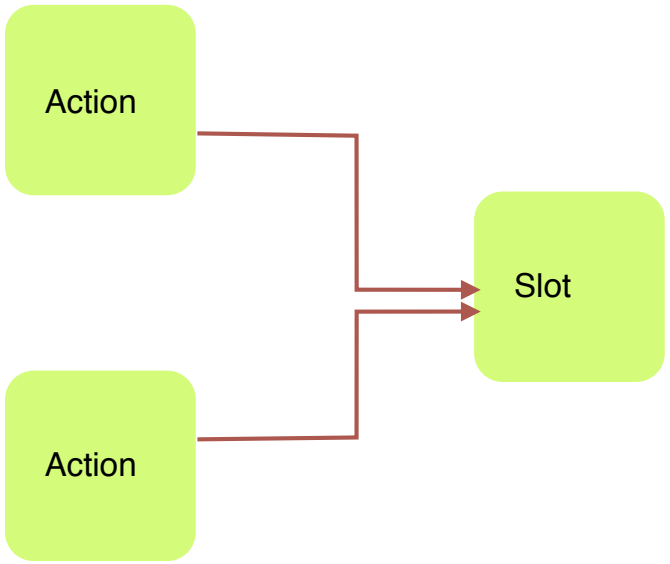
Comment différencier des actions dans un même slot ?



- modèle **signal / slot** => le récepteur ne **connaît pas** l'émetteur
- mais **parfois c'est utile !**

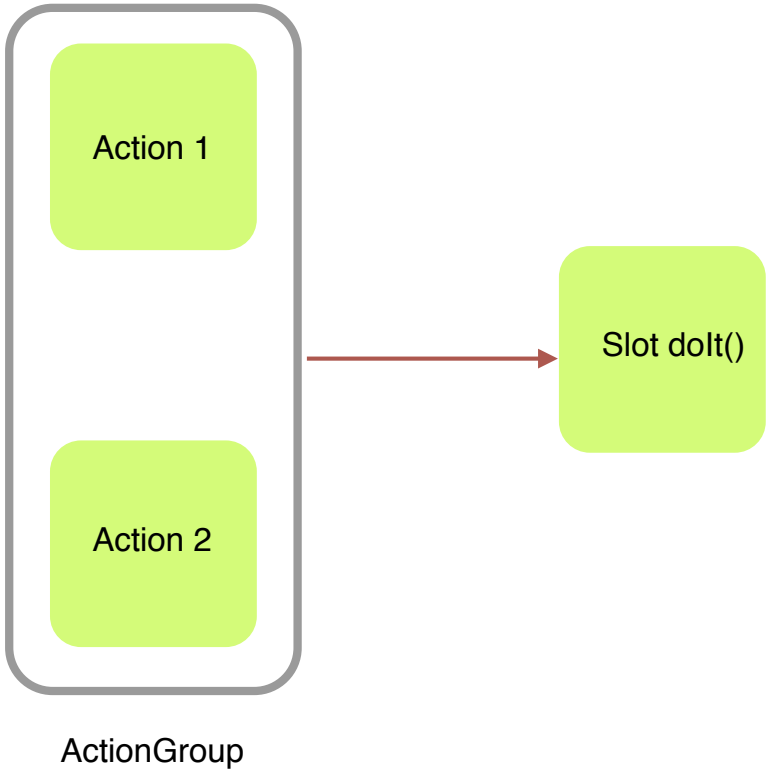
Solution 1: QObject::sender()

```
// Dans le .h en variables d'instance de MaClasse

    QAction * action1, * action2, …;

// Dans le .cpp

void MaClasse::createGUI( ) {
      action1 = new QAction(tr("Action 1"), this);
      connect(action1, SIGNAL(triggered()), this, SLOT(doIt()));

      action2 = new QAction(tr("Action 2"), this);
      connect(action2, SIGNAL(triggered()), this, SLOT(doIt()));
         …
}

void MaClasse::doIt( ) {
       QObject * sender = QObject::sender( );          // un peu comme getSource() de Java/Swing
       if (sender == action1) ....;
       else  if (sender == action2) .... ;
     ....
}
```



Solution 2: QActionGroup

```
// Dans le .h en variables d'instance de MaClasse

    QAction * action1, * action2, ...;

// Dans le .cpp

void MaClasse::createGUI( ) {
      QActionGroup *group = new QActionGroup(this);
      // un seul connect !
      connect(group, SIGNAL(triggered(QAction *)),
                  this, SLOT(doIt(QAction *)));
      action1 = group->addAction(tr("Action 1"));
      action2 = group->addAction(tr("Action 2"));
      ...
}

void MaClasse::doIt(QAction * sender ) {          // l'action est récupérée via le paramètre
     if (sender == action1) ....;                 // transmis au slot
     else  if (sender == action2) .... ;
     ....
}
```



Solution 2: QActionGroup

Par défaut le groupe est exclusif,

sinon faire :

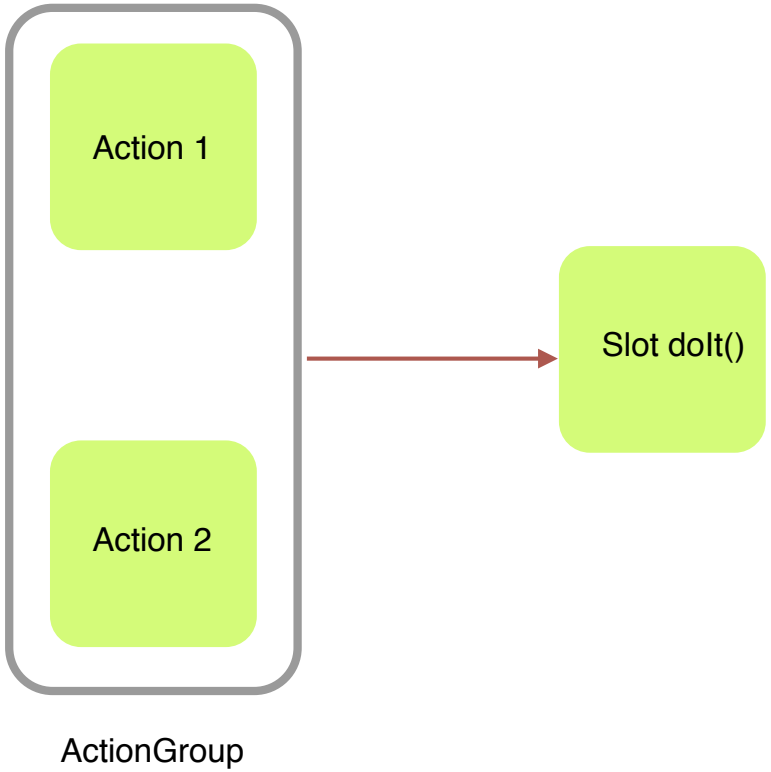
```
group->setExclusive(false);
```

On peut faire de même pour les boutons
(QPushButton, QRadioButton, QCheckBox …)

```
QButtonGroup * group = new QButtonGroup(this);
```

avec les signaux de **QButtonGroup** :

```
buttonClicked(QAbstractButton * button)
buttonClicked(int id)
```



Solution 3: QSignalMapper

```
void MaClasse::createGUI( ) {
    QSignalMapper* mapper = new QSignalMapper (this) ;
    connect(mapper, SIGNAL(mapped(int)), this, SLOT(doIt(int))) ;

    QPushButton * btn1 = new QPushButton("Action 1"), this);
    connect (btn1, SIGNAL(clicked( )), mapper, SLOT(map( ))) ;
    mapper->setMapping (btn1, 1) ;

    QPushButton * btn2 = new QPushButton("Action 1"), this);
    connect (btn2, SIGNAL(clicked( )), mapper, SLOT(map( ))) ;
    mapper->setMapping (btn2, 2) ;
     ...
}

void MaClasse::doIt(int value) {          // l'action est récupérée via le paramètre
    ....
}
```

Possible pour les types : int, QString&, QWidget* et QObject*

Graphique 2D illustré en Qt

Dessiner dans un widget

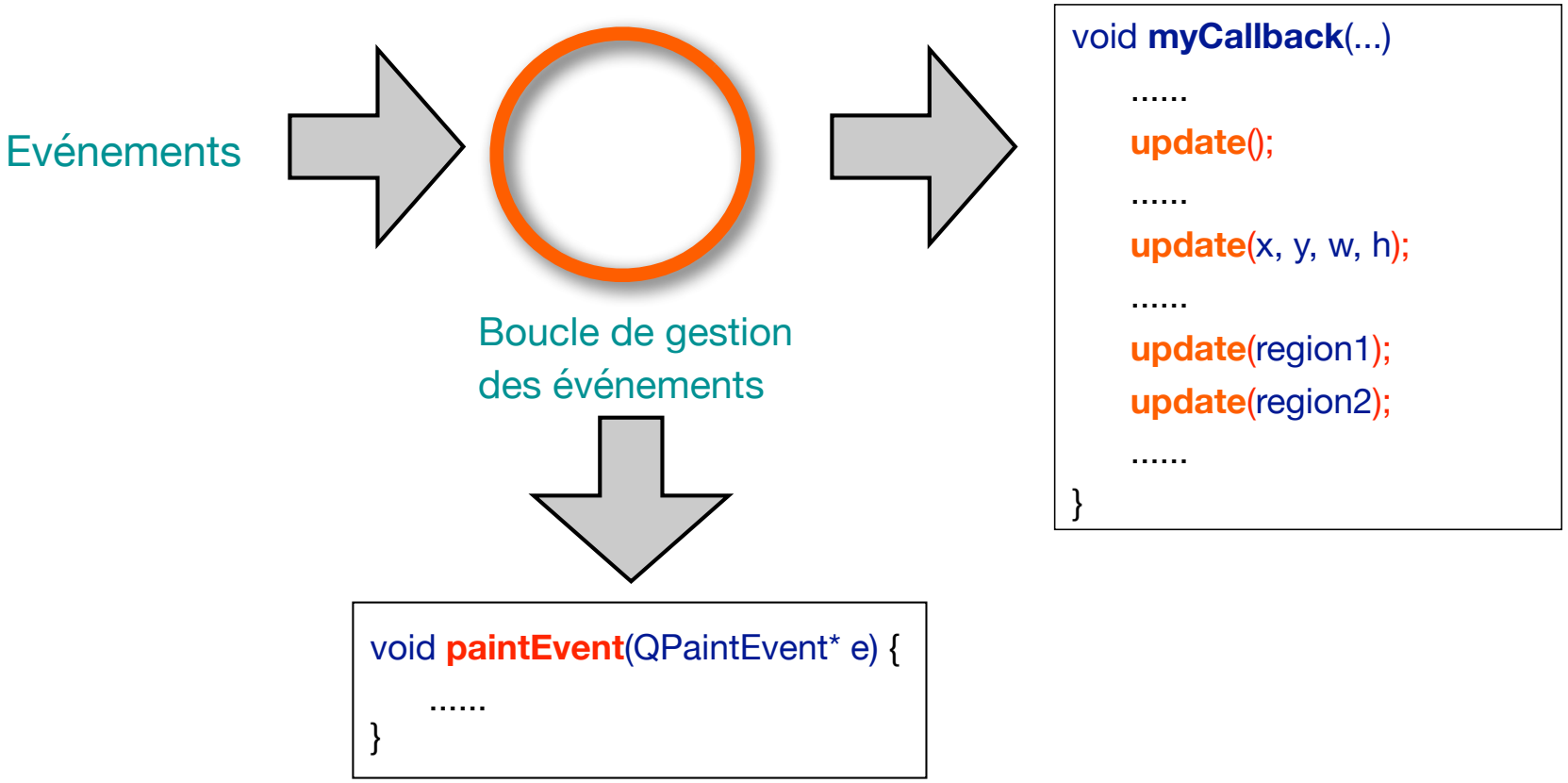
Un widget n'est repeint que lorsque c'est nécessaire

- l'application est **lancée** ou **dé-iconifiée**
- l'application est **déplacée** (selon l'OS)
- une fenêtre qui **cachait** une partie du widget est déplacée
- on le demande **explicitement** via la fonction **update**( )

= Modèle « damaged / repaint »

- contrairement aux **applications 3D** où généralement on **réaffiche en boucle**

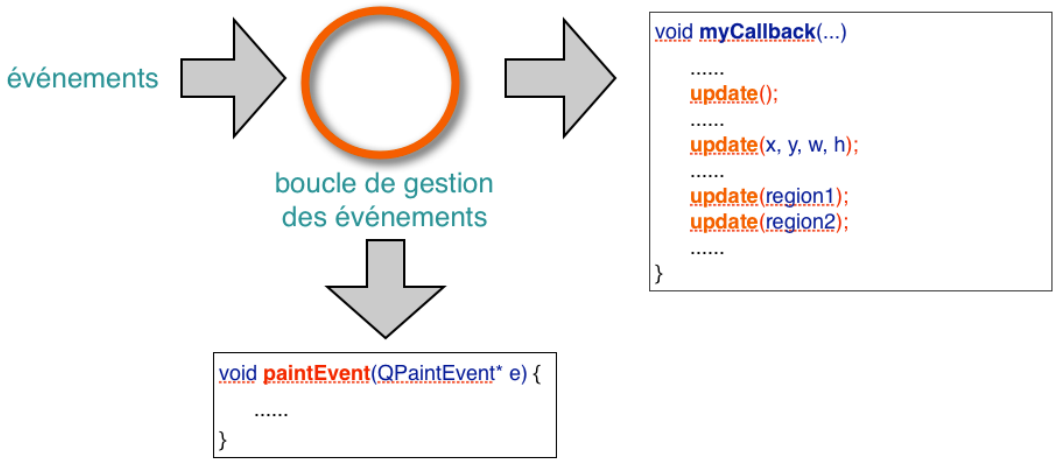
Modèle "damaged / repaint"



update() indique qu'une zone est **endommagée** => il faut **repeindre**

paintEvent() est alors **automatiquement** appelée

Modèle "damaged / repaint"



update() = zone **endommagée**

paintEvent() = **réaffiche**

met les demandes dans une **file d'attente**

zone = **tout le widget** sauf si on précise

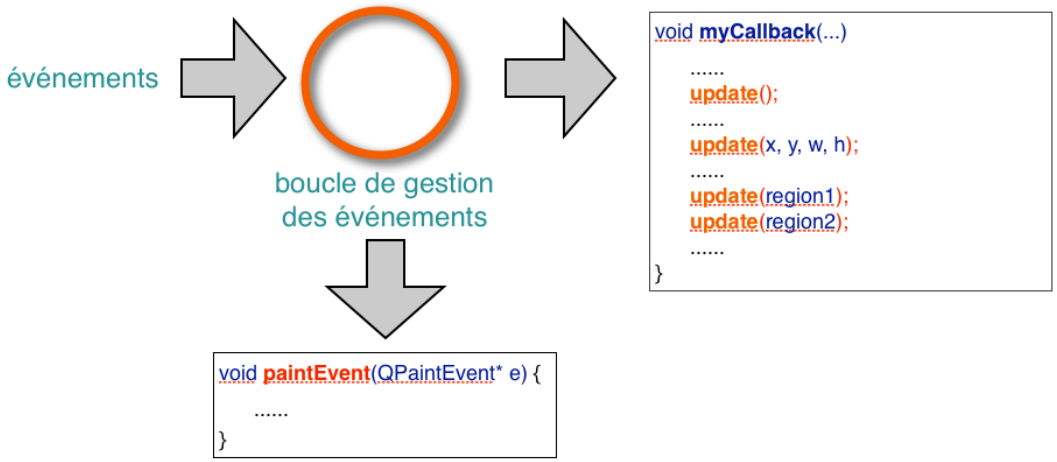
n'est appelée qu'**une seule fois,**

au retour dans la boucle

Attention

- afficher **uniquement** dans **paintEvent**()
- ne **pas** appeler **paintEvent**() directement

Modèle "damaged / repaint"



Compléments

repaint() entraîne un réaffichage **immédiat** (contrairement à **update**())

ne **pas** utiliser sauf cas particuliers (animations)

Java fonctionne de manière **similaire** mais noms **différents** !

update() Qt == **repaint**() Java

paintEvent() Qt == **paint**() Java

Redessiner

Créer une sous-classe de QWidget qui redéfinit paintEvent()

Header Canvas.h

```
#include <QWidget>

class Canvas : public QWidget {
public:
    Canvas(QWidget* parent) : QWidget(parent) {}

protected:
    virtual void paintEvent(QPaintEvent*);
};
```

Implémentation Canvas.cpp

```
#include <QPainter>
#include "Canvas.h"

void Canvas::paintEvent(QPaintEvent* e) {
    // comportement standard (affiche le fond)
    QWidget::paintEvent(e);

    // crée un Painter pour ce Canvas
    QPainter painter(this);
    painter.drawLine(50, 10, 100, 20);
}
```

N'utiliser **QPainter que** dans **paintEvent**() (à cause du double buffering)

Détecter les événements

Méthodes de QWidget appelées quand :

- on **presse/relâche** un bouton
- on **double-clique**
- on **déplace** la souris
  - en appuyant (ou pas) sur le bouton souris selon mouseTracking()

Remarque

- **pas** de signals / slots !

```
void mousePressEvent(QMouseEvent*);

void mouseReleaseEvent(QMouseEvent*);

void mouseDoubleClickEvent(QMouseEvent*);

void mouseMoveEvent(QMouseEvent*);

void setMouseTracking(bool)

bool hasMouseTracking()
```

Détecter les événements

Canvas.h

```
#include <QWidget>
#include <QMouseEvent>

class Canvas : public QWidget {
public:
    Canvas(QWidget* p) : QWidget(p) {}

protected:
   void mousePressEvent(QMouseEvent*);
};
```

Canvas.cpp

```
#include "Canvas.h"

void Canvas::mousePressEvent(QMouseEvent* e) {
     if (e->button() == Qt::LeftButton) {
            ......
            .....
       update();              // demande de réaffichage
    }
}
```

QMouseEvent permet de récupérer :

- **button**() **bouton** souris qui a déclenché l'événement. ex: **Qt::LeftButton**
- **buttons**() état des **autres** boutons. ex: **Qt::LeftButton** | **Qt::MidButton**
- **modifiers**() modificateurs **clavier**. ex: **Qt::ControlModifier** | **Qt:: ShiftModifier**

Récupérer la position du curseur

QMouseEvent permet de récupérer :

- la position **locale** (relative au widget) :
  pos( )
  localPos( )

- la position **relative** à un référentiel :
  globalPos( )
  screenPos( )
  windowPos( )
  => utile si on **déplace** le widget interactivement

Canvas.cpp

```
#include "Canvas.h"

void Canvas::mousePressEvent(QMouseEvent* e) {

    if (e->button() == Qt::LeftButton) {
            ......
            ......
      update();
    }
}
```

Alternative

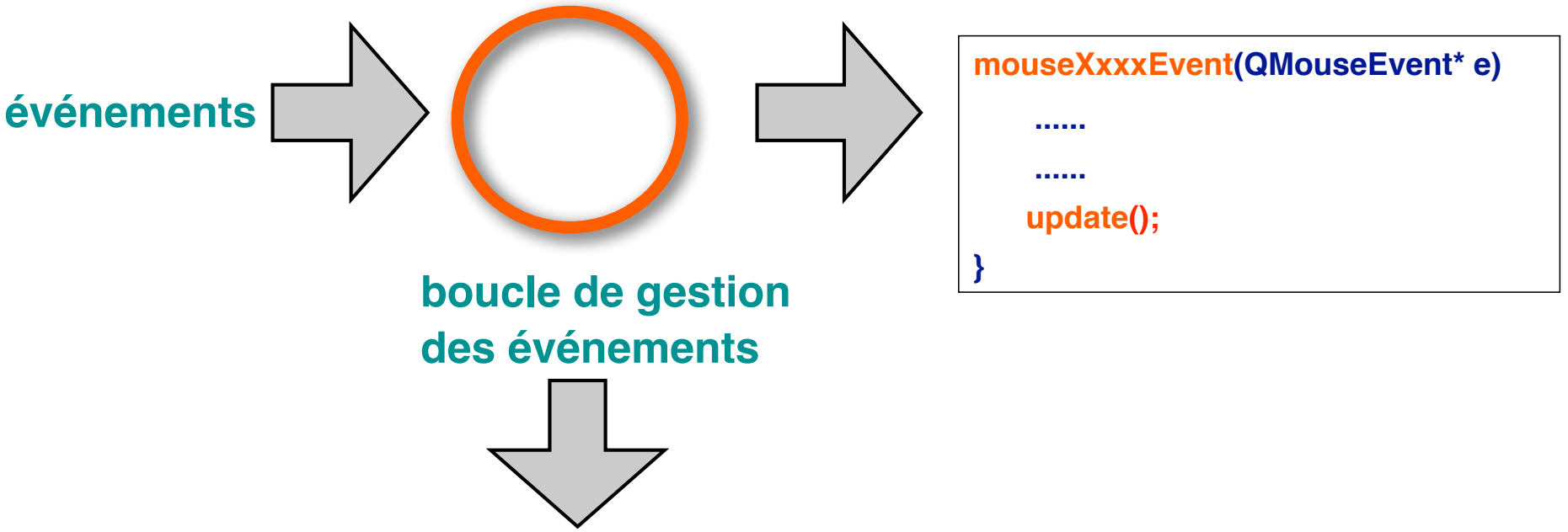
- **QCursor::**pos( ) : position du curseur sur **l'écran**
- **QWidget::**mapToGlobal( ), mapFromGlobal( ), etc. : **conversion** de coordonnées

Ignorer les événements

Ignorer les événements

- **QEvent::ignore( ) :** signifie que le receveur **ne veut pas l'événement**
  - ex : dans méthode closeEvent() de la MainWindow pour ne pas quitter l'appli

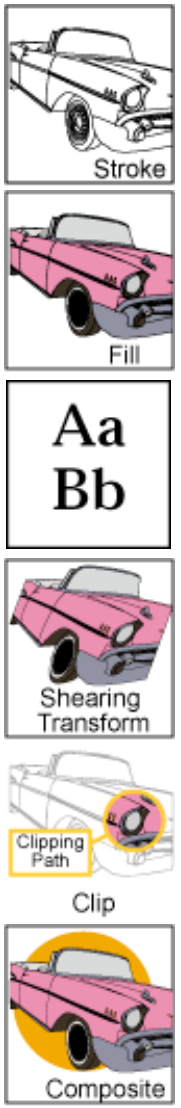
Synthèse


événements
boucle de gestion
des événements
mouseXxxxEvent(QMouseEvent* e)
.....
.....
update();
}
paintEvent(QPaintEvent* e) {
QWidget::paintEvent(e);
QPainter.painter(this);
.....
}

QPainter

Attributs

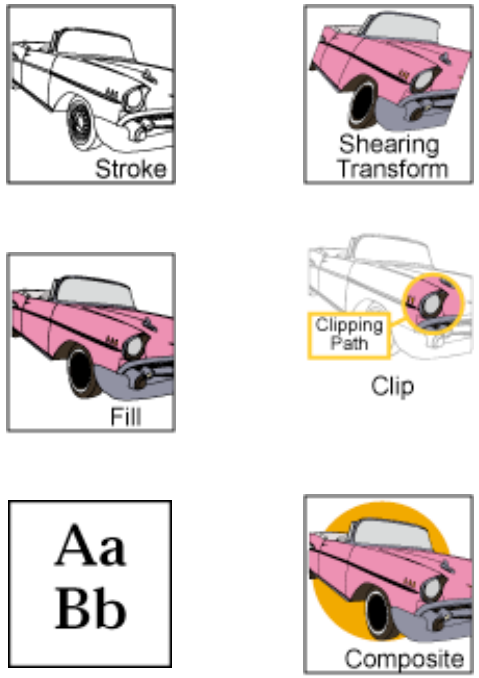
- setPen( ) : lignes et contours
- setBrush( ) : remplissage
- setFont( ) : texte
- setTransform( ), etc. : transformations affines
- setClipRect/Path/Region( ) : clipping (découpage)
- setCompositionMode( ) : composition



QPainter

Lignes et contours

- drawPoint(), drawPoints()
- drawLine(), drawLines()
- drawRect(), drawRects()
- drawArc(), drawEllipse()
- drawPolygon(), drawPolyline(), etc...
- **drawPath**() : **path** = chemin complexe



Remplissage

- fillRect(), fillPath()

Divers

- drawText()
- drawPixmap(), drawImage(), drawPicture(), etc.

QPainter

Classes utiles

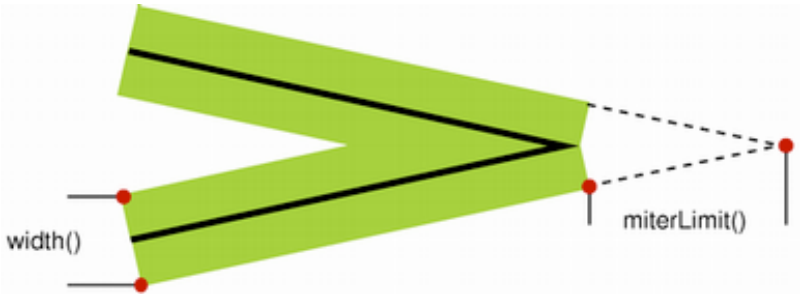
- entiers: QPoint, QLine, QRect, QPolygon
- flottants: QPointF, QLineF, ...
- chemin complexe: QPainterPath
- zone d'affichage: QRegion

Pinceau: QPen

Attributs

- style : type de ligne
- width : épaisseur **(0 = «cosmétique»)**
- brush : attributs du pinceau (couleur...)
- capStyle : terminaisons
- joinStyle : jointures

Qt::PenStyle



Join Style

Cap Style

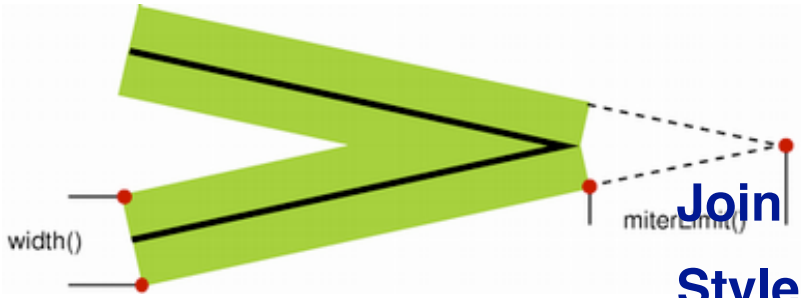
Pinceau: QPen

Exemple

```
// dans méthode paintEvent()
QPen pen;        // default pen
pen.setStyle(Qt::DashDotLine);
pen.setWidth(3);
pen.setBrush(Qt::green);
pen.setCapStyle(Qt::RoundCap);
pen.setJoinStyle(Qt::RoundJoin);

QPainter painter(this);
painter.setPen(pen);
```

Qt::PenStyle

Cap Style



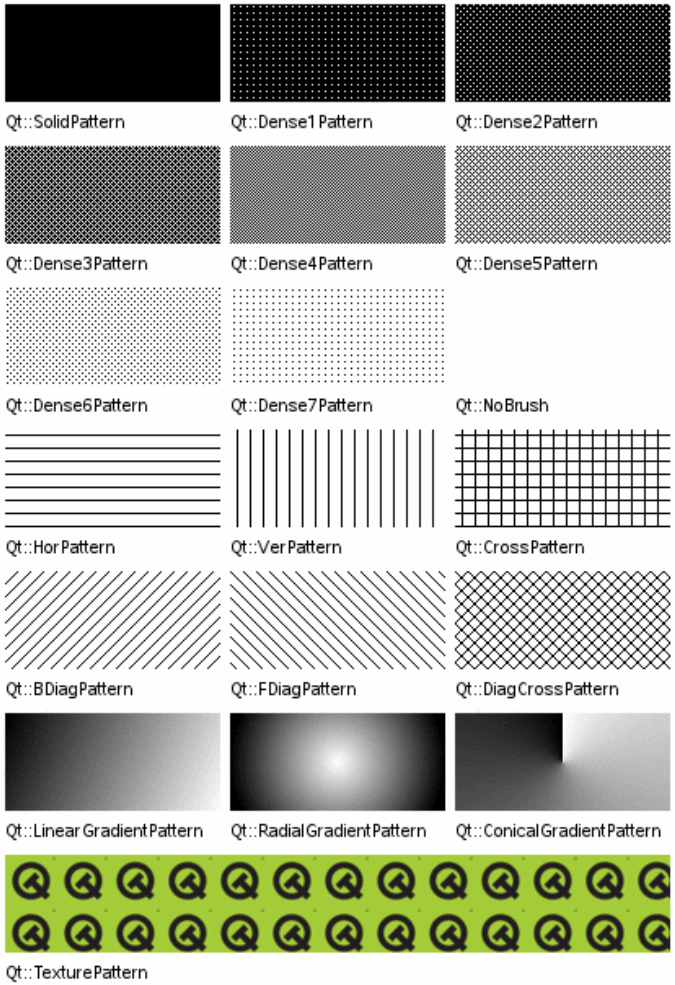
Join Style

Remplissage: QBrush

Attributs

- style
- color
- gradient
- texture

```
QBrush brush( ... );
.....
QPainter painter(this);
painter.setBrush(brush);
```



Qt::BrushStyle

Remplissage: QBrush

Attributs

- style
- color
- gradient
- texture

Qt::GlobalColor

QColor

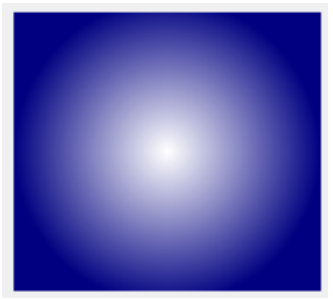
- modèles **RGB**, **HSV** or **CMYK**
- composante alpha (transparence) : alpha blending
- couleurs prédéfinies: **Qt::GlobalColor**

Remplissage: gradients

Type de gradients

- lineaire
- radial
- conique

QLinearGradient

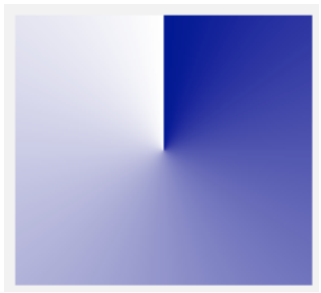


QRadialGradient

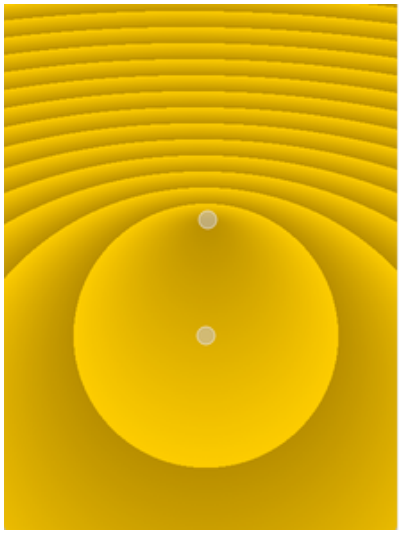
Type de coordonnées

- défini par QGradient::CoordinateMode

```
QLinearGradient
    gradient( QPointF(0, 0), QPointF(100, 100) );

gradient.setColorAt(0, Qt::white);
gradient.setColorAt(1, Qt::blue);

QBrush brush(gradient);
```



QConicalGradient

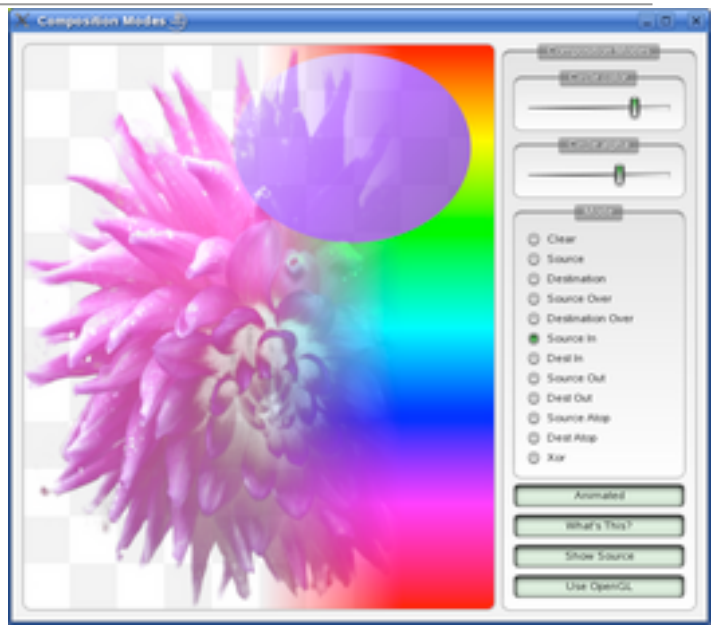


répétition: setSpread()

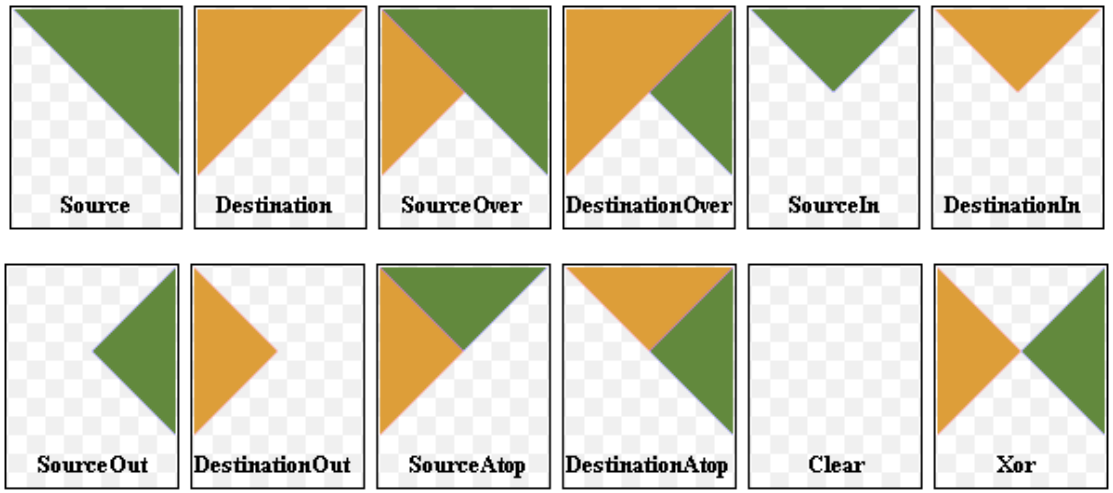
Composition

Modes de composition

- opérateurs de **Porter Duff:**
- définissent : **F(source, destination)**
- défaut : **SourceOver**
  - avec alpha blending
  - dst <= a_{src} * src + (1-a_{src}) * a_{dst} * dst



QPainter::setCompositionMode( )



Limitations

- selon implémentation et **Paint Device**

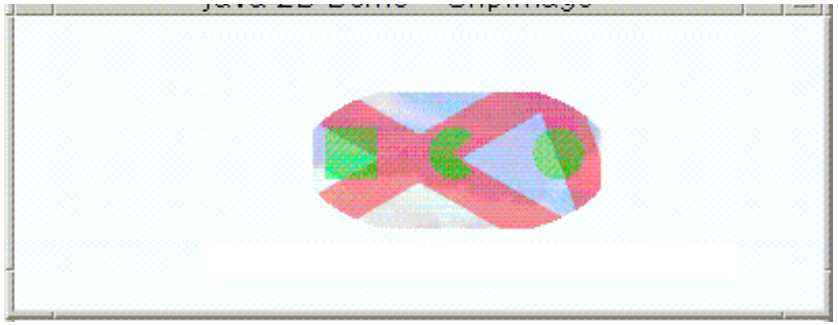
Découpage (clipping)

Découpage

- selon un rectangle, une région ou un path
- QPainter::setClipping(), setClipRect(), setClipRegion(), setClipPath()

QRegion

- united(), intersected(), subtracted(), xored()

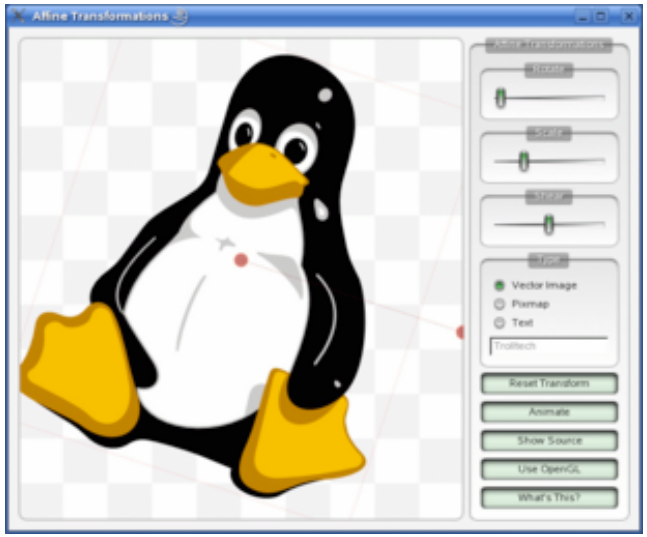


```
QRegion r1(QRect(100, 100, 200, 80), QRegion::Ellipse);     // r1: elliptic region
QRegion r2(QRect(100, 120, 90, 30));                        // r2: rectangular region
QRegion r3 = r1.intersected(r2);                            // r3: intersection

QPainter painter(this);
painter.setClipRegion(r3);
...etc...
```

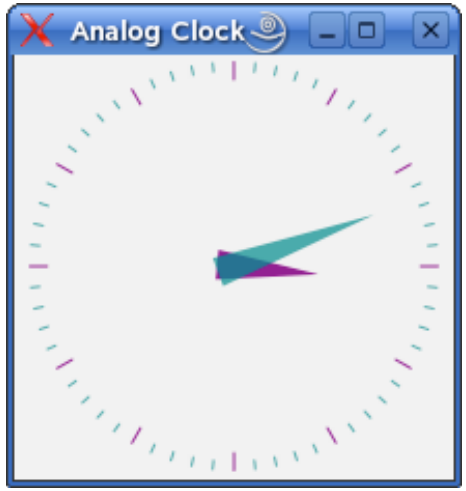
Transformations affines

Transformations

- translate()
- rotate()
- scale()
- shear()
- setTransform()




```
QPainter painter( this );
painter.translate( width( )/2, height( )/2 );
painter.scale( side/200.0, side/200.0 );

painter.save( );                    // empile l'état courant
painter.rotate( 30.0 * ((time.hour() + time.minute() / 60.0)) );
painter.drawConvexPolygon( hourHand, 3 );
painter.restore( );                 // dépile
```



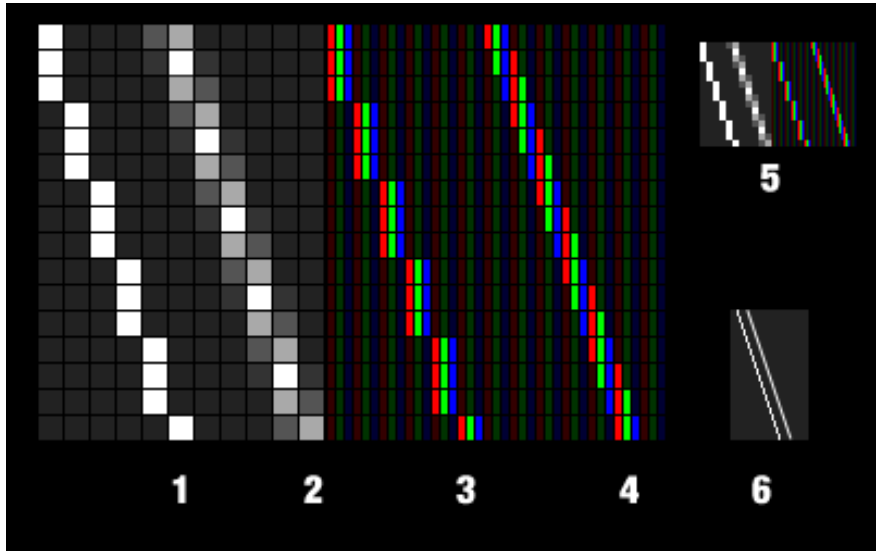
Antialiasing

Anti-aliasing

- éviter l'effet d'escalier
- particulièrement utile pour les polices de caractères

Subpixel rendering

- exemples : **ClearType**, texte sous MacOSX



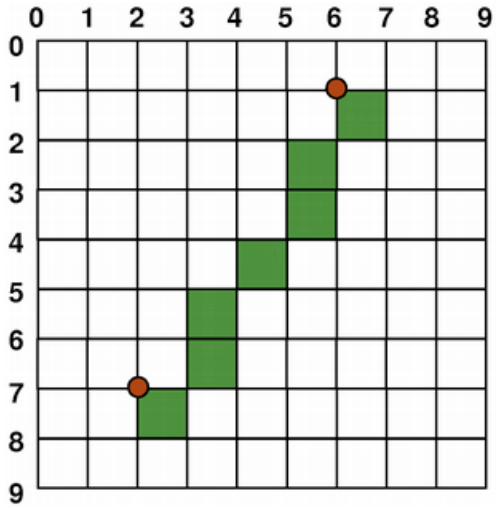
ClearType (Wikipedia)

MacOSX

Antialiasing

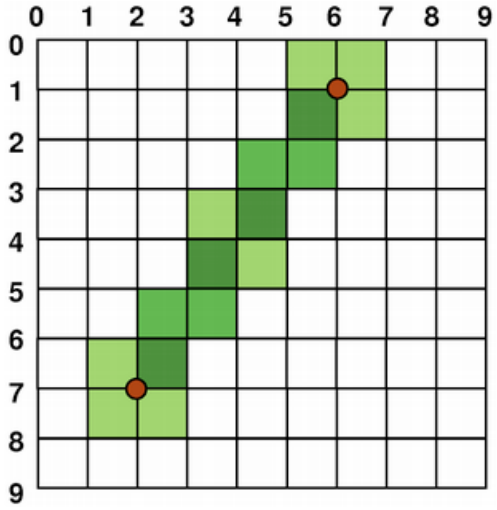
Anti-aliasing sous Qt

```
QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing);
painter.setPen(Qt::darkGreen);
painter.drawLine(2, 7, 6, 1);
```



Rendering hints

- “hint” = option de rendu
  - effet non garanti
  - dépend de l’implémentation et du matériel
- QPainter::setRenderingHints( )



Antialiasing et cordonnées

Epaisseurs impaire

- pixels dessinés à droite et en dessous

Dessin anti-aliasé

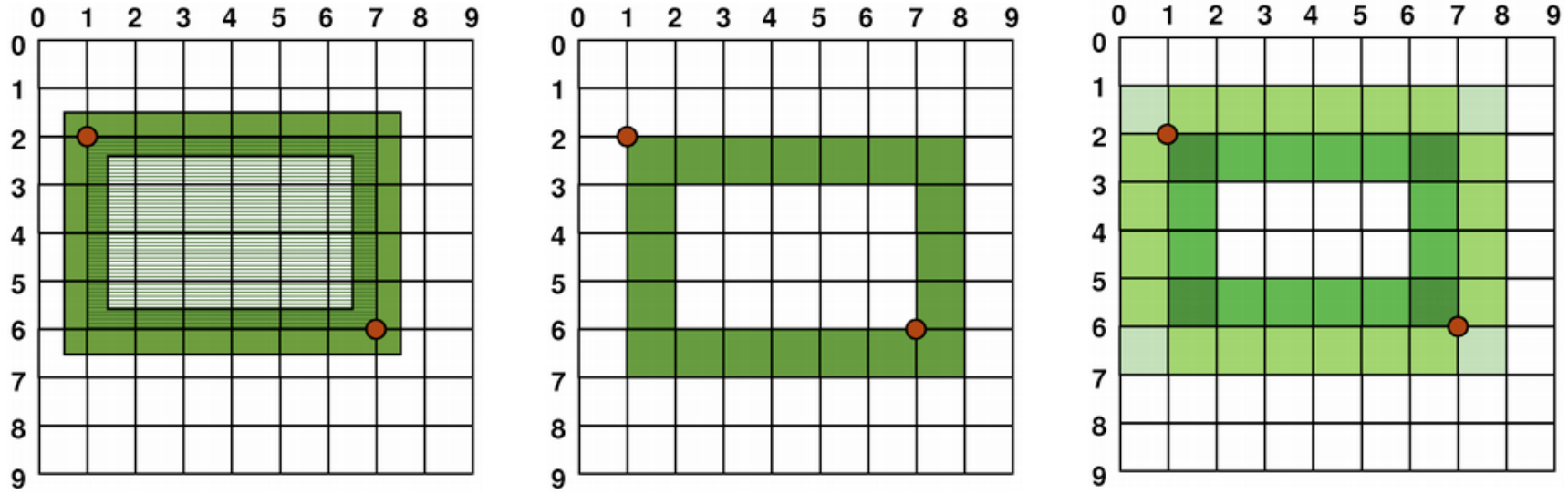
- pixels répartis autour de la ligne idéale

Note

QRect::right() = left() + width() -1

QRect::bottom() = top() + height() -1

Mieux : QRectF (en flottant)



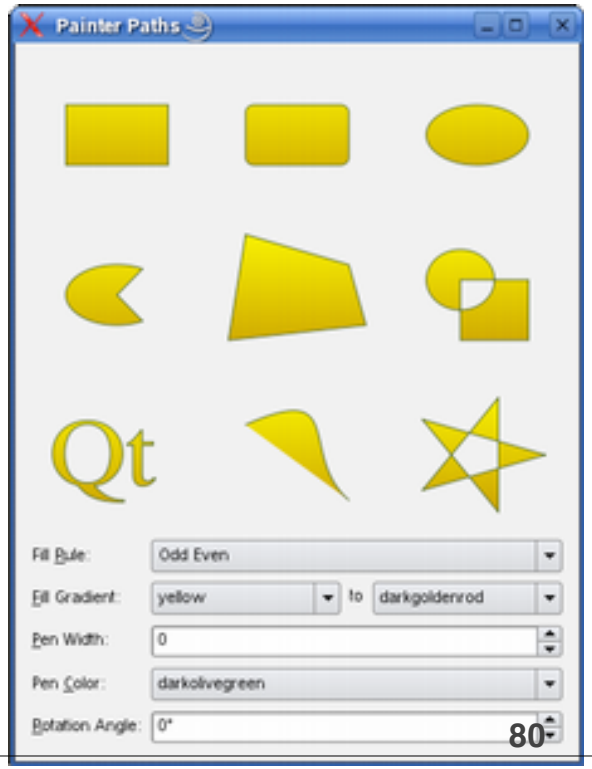
Paths

QPainterPath

- **suite arbitraire** de lignes et courbes
  - affichée par: **QPainter::drawPath**()
  - sert aussi pour remplissage, profilage, découpage

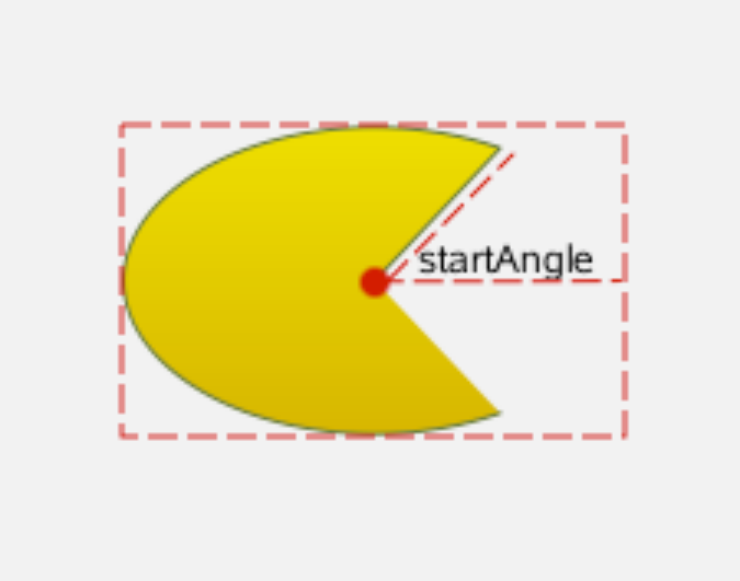
Méthodes

- **déplacements**: moveTo(), arcMoveTo()
- **dessin**: lineTo(), arcTo()
- **courbes** de Bezier: quadTo(), cubicTo()
- addRect(), addEllipse(), addPolygon(), addPath() ...
- addText()
- translate(), union, addition, soustraction...
- et d'autres encore ...



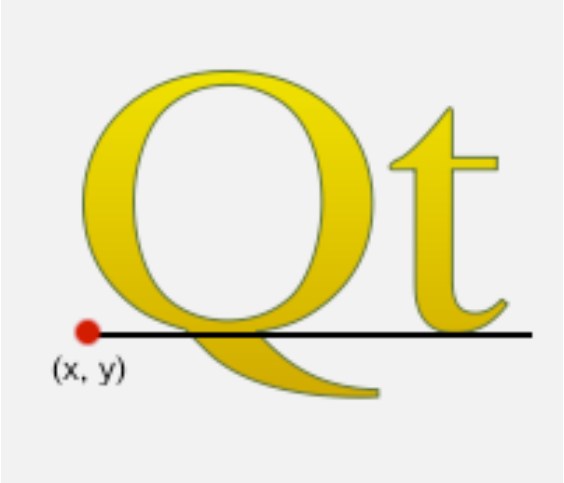
Paths

```
QPointF center, startPoint;
QPainterPath myPath;

myPath.moveTo( center );
myPath.arcTo( boundingRect, startAngle, sweepAngle );

QPainter painter( this );
painter.setBrush( myGradient );
painter.setPen( myPen );
painter.drawPath( myPath );
```



Paths

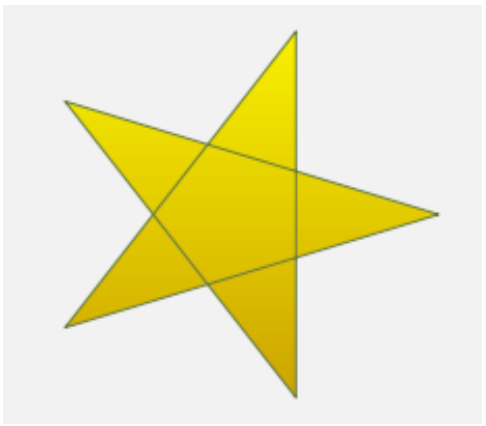
```
QPointF baseline(x, y);
QPainterPath myPath;

myPath.addText( baseline, myFont, "Qt" );

QPainter painter( this );
 ... etc....
painter.drawPath( myPath );
```



Paths

```
QPainterPath path;
path.addRect(20, 20, 60, 60);
path.moveTo(0, 0);
path.cubicTo(99, 0,  50, 50,  99, 99);
path.cubicTo(0, 99,  50, 50,  0, 0);

QPainter painter(this);
painter.fillRect(0, 0, 100, 100, Qt::white);
painter.setPen(QPen(QColor(79, 106, 25), 1, Qt::SolidLine, Qt::FlatCap, Qt::MiterJoin));
painter.setBrush(QColor(122, 163, 39));
painter.drawPath(path);
```

Qt::OddEvenFill (défaut)



Qt::WindingFill

Images

Format RGB

```
QImage image(3, 3, QImage::Format_RGB32);
QRgb value;
value = qRgb(122, 163, 39); // 0xff7aa327
image.setPixel(0, 1, value);
image.setPixel(1, 0, value)
```

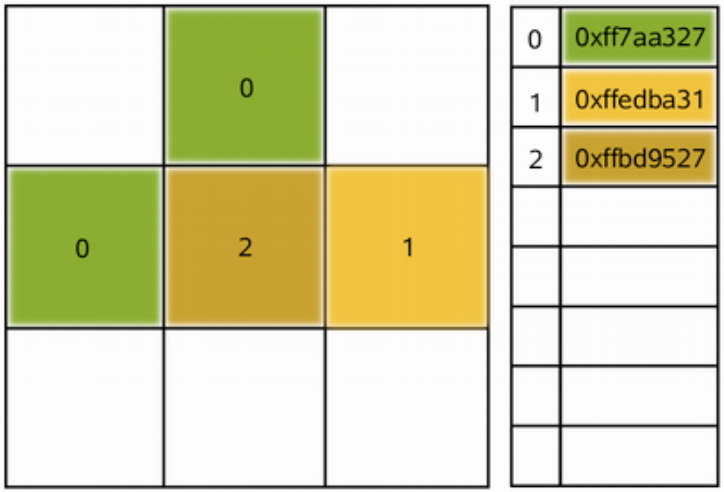
Entrées/sorties

- load() / save() : depuis/vers un fichier (principaux formats supportés)
- loadFromData() : depuis la mémoire
- également QPixmap, QBitmap : optimisé pour **affichage**

Images

Format Indexé

```
QImage image(3, 3, QImage::Format_Indexed8);
QRgb value;

value = qRgb(122, 163, 39); // 0xff7aa327
image.setColor(0, value);

value = qRgb(237, 187, 51); // 0xffedba31
image.setColor(1, value);

image.setPixel(0, 1, 0);
image.setPixel(1, 0, 1);
```



QPicture

Enregistrer et rejouer les commandes d'un QPainter

Enregistrer :

```
QPicture picture;
QPainter painter;
painter.begin( &picture );                // paint in picture
painter.drawEllipse( 10,20, 80,70 );      // draw an ellipse
painter.end( );                           // painting done
picture.save( "drawing.pic" );            // save picture
```

Rejouer :

```
QPicture picture;
picture.load( "drawing.pic" );            // load picture
QPainter painter;
painter.begin( &myImage );                // paint in myImage
painter.drawPicture( 0, 0, picture );     // draw the picture at (0,0)
painter.end( );                           // painting done
```

Picking / Interaction

Picking avec QRect, QRectF

- intersects()
- contains()

Picking avec QPainterPath

- intersects(const QRectF & rectangle)
- intersects(const QPainterPath & path)
- contains(const QPointF & point)
- contains(const QRectF & rectangle)
- contains(const QPainterPath & path)

Retourne l'intersection

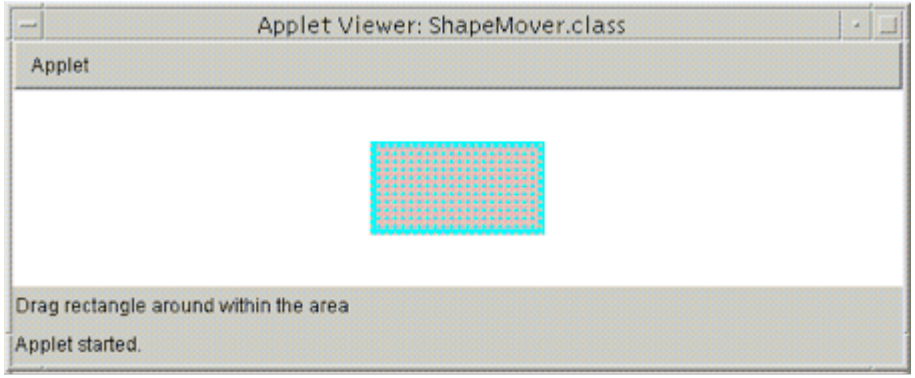
- QPainterPath intersected(const QPainterPath & path)

Picking / Interaction

Exemple

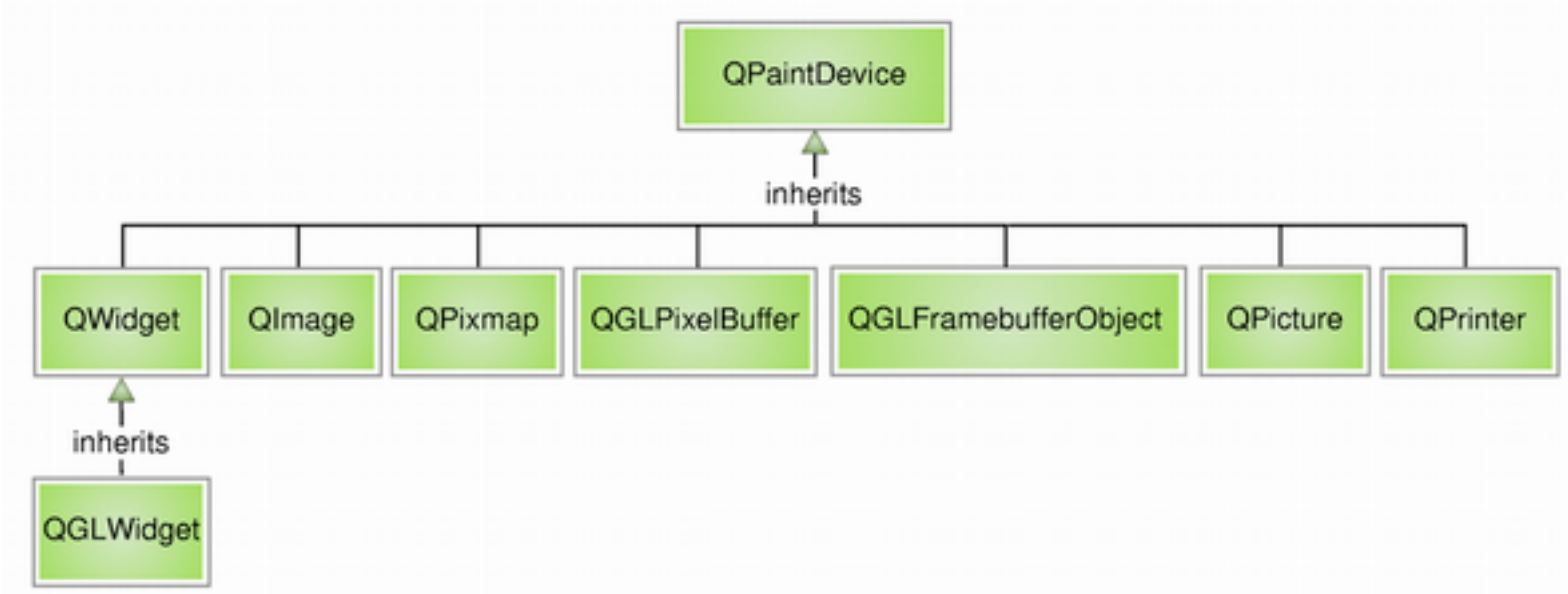
- teste si la souris est dans le rectangle quand on appuie sur le bouton de la souris
- met à jour la position du rectangle pour qu'il soit centré

```
QRect rect;       // variable d'instance de mon Widget de dessin
void mousePressEvent(QMouseEvent* e) {
    if (rect.contains( e->pos() )) {
       rect.moveCenter( e->pos() );
       update( );
}
...
void paintEvent(QPaintEvent* e ) {
     QPainter painter( this );
     .....            // specifier attributs graphiques
     painter.drawRect( rect );
}
```



Surfaces d'affichage

– QPainter : outil de dessin
– QPaintDevice : objet sur lequel on peut dessiner (QWidget en dérive)
– QPaintEngine : moteur de rendu



Graphique structuré 2D

Graphics View

- graphique structuré
  - représentation objet du dessin
  - réaffichage, picking automatiques
- **vues d'un graphe de scène :**
  - QGraphicsScene
  - QGraphicsView
  - QGraphicsItem, etc.

```
QGraphicsScene scene;
QGraphicsRectItem * rect =
      scene.addRect( QRectF(0,0,100,100) );
QGraphicsItem *item = scene.itemAt(50, 50);
      .....
QGraphicsView view( &scene );
view.show( );
```

Performance de l'affichage

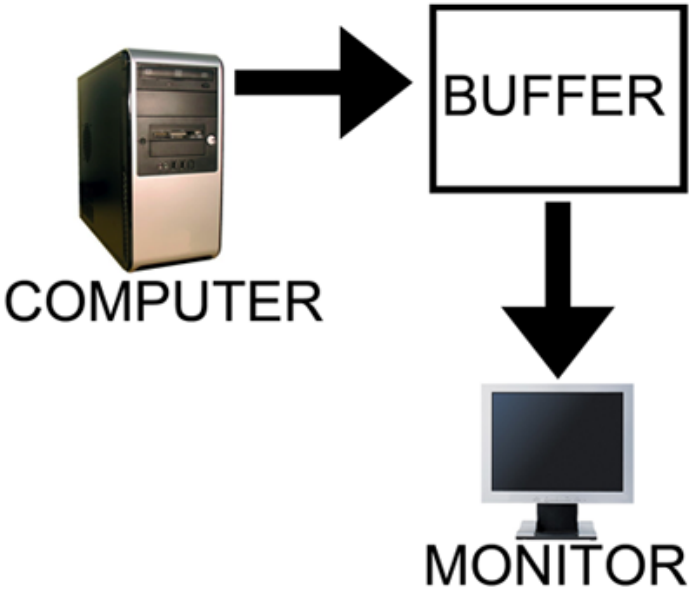
Problèmes classiques

- **Flickering** et **Tearing** (scintillement et déchirement)
- **Lag** (latence)

Flicker

Flicker

- scintillement de l'affichage
- exemple :
  https://www.youtube.com/watch?v=QoZ8L1quWnc



Problème

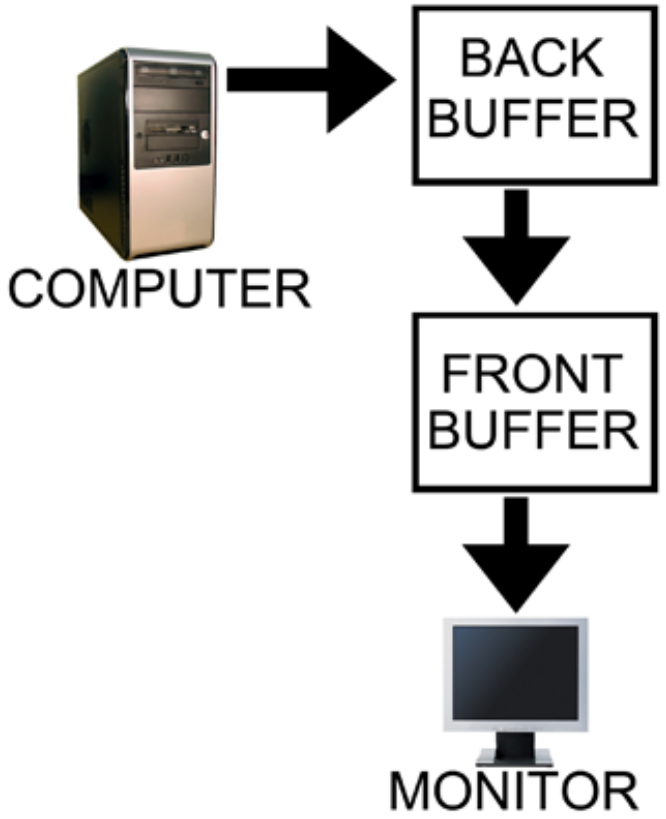
- l'oeil perçoit les étapes intermédiaires
- se produit (normalement) en « simple buffering »
  - dessin directement sur le buffer de l'écran

source: AnandTech

Double buffering

Solution au flicker

- dessin dans le back buffer (caché)
- recopie dans le front buffer (écran)
- par défaut avec **Qt** ou **Java Swing**



source: AnandTech

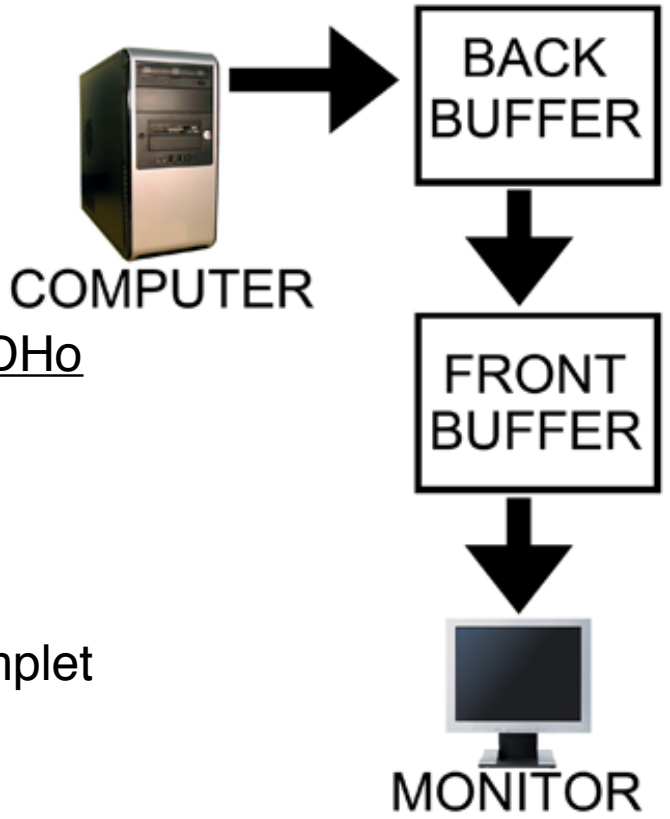
Tearing

Tearing

– l'image apparait en 2 (ou 3...) parties horizontales

– exemple :

• https://www.youtube.com/watch?v=1nFBtTq0DHo



Problème

– recopie du back buffer avant que le dessin soit complet

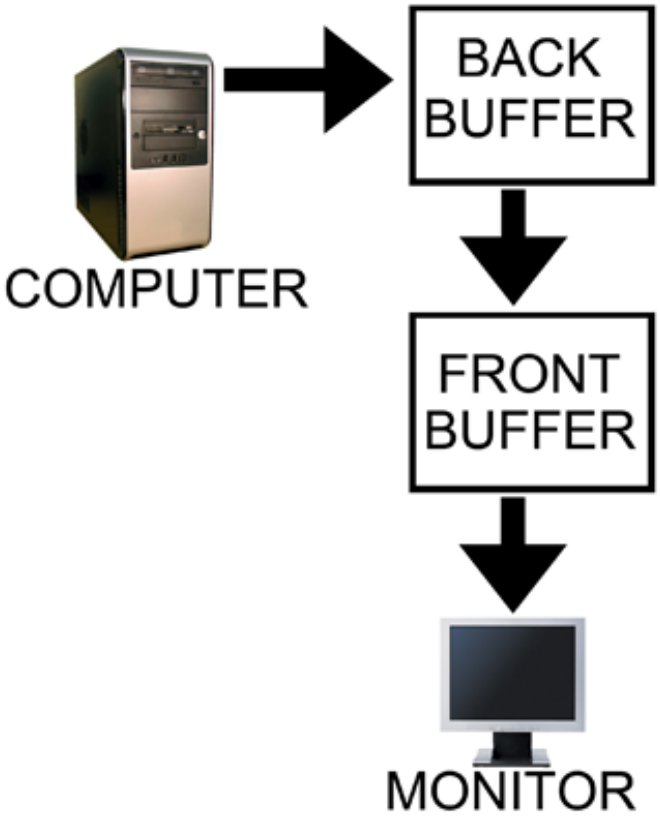
• mélange de plusieurs "frames" vidéo

• en particulier avec jeux vidéo

Tearing

Solution

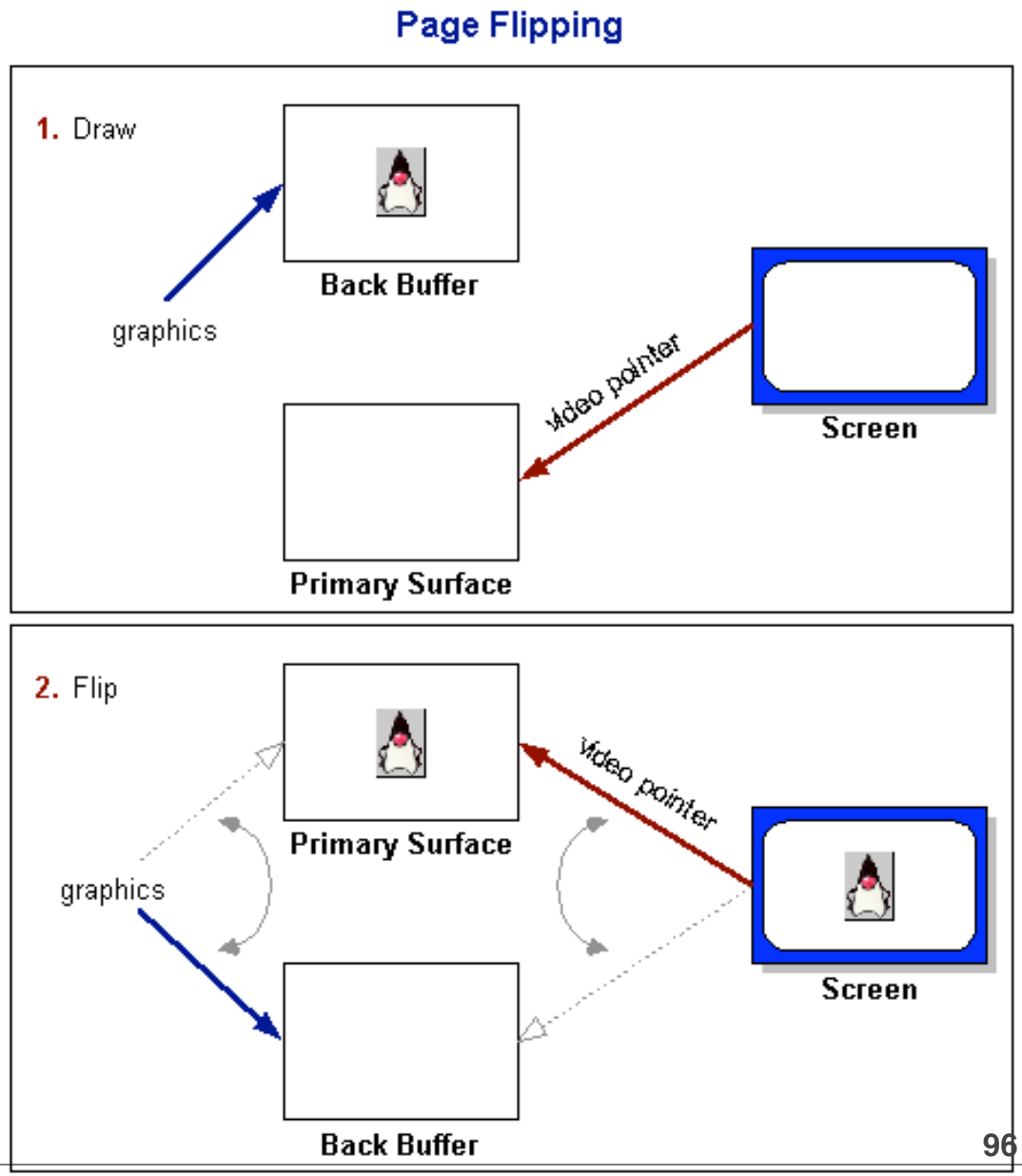
- **VSync** (Vertical synchronization)
- rendu synchronisé avec l'affichage
- inconvénient : ralentit l'affichage



Page flipping

Page flipping

- alternative au double buffering
- pas de recopie des pixels, on change juste de buffer
- plus rapide (mais tout de même contraint par VSync)

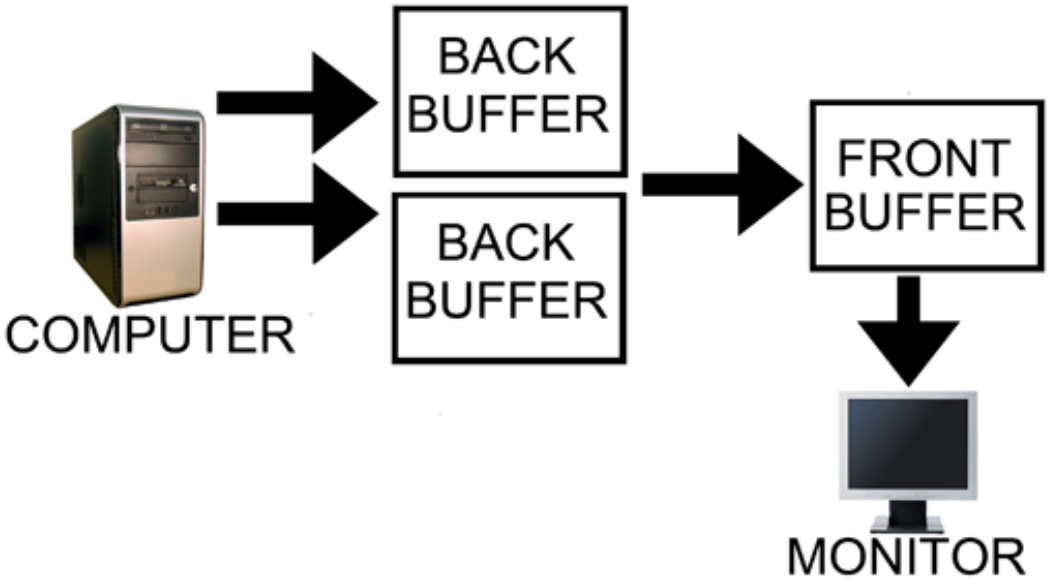


src: Oracle/JavaSE

Triple buffering

Evite le ralentissement dû à la VSync

- un des deux back buffers est toujours complet
- le front buffer (écran) peut être mis à jour sans attendre

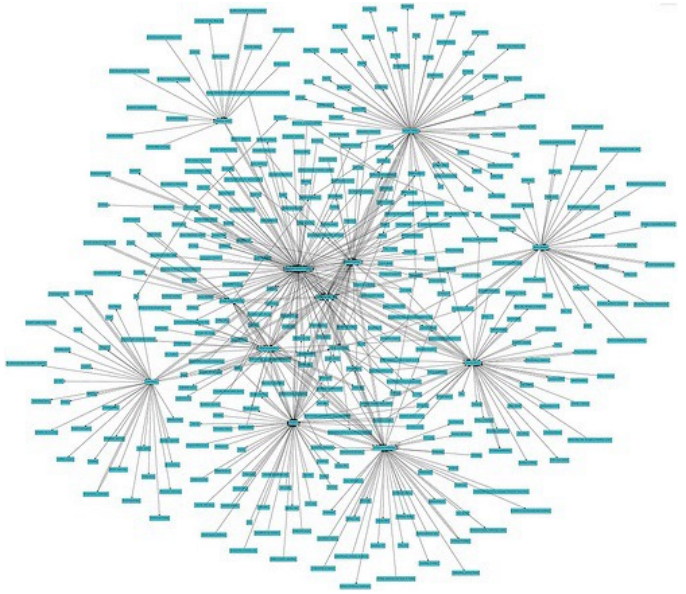


source: AnandTech

Latence (dessin interactif)

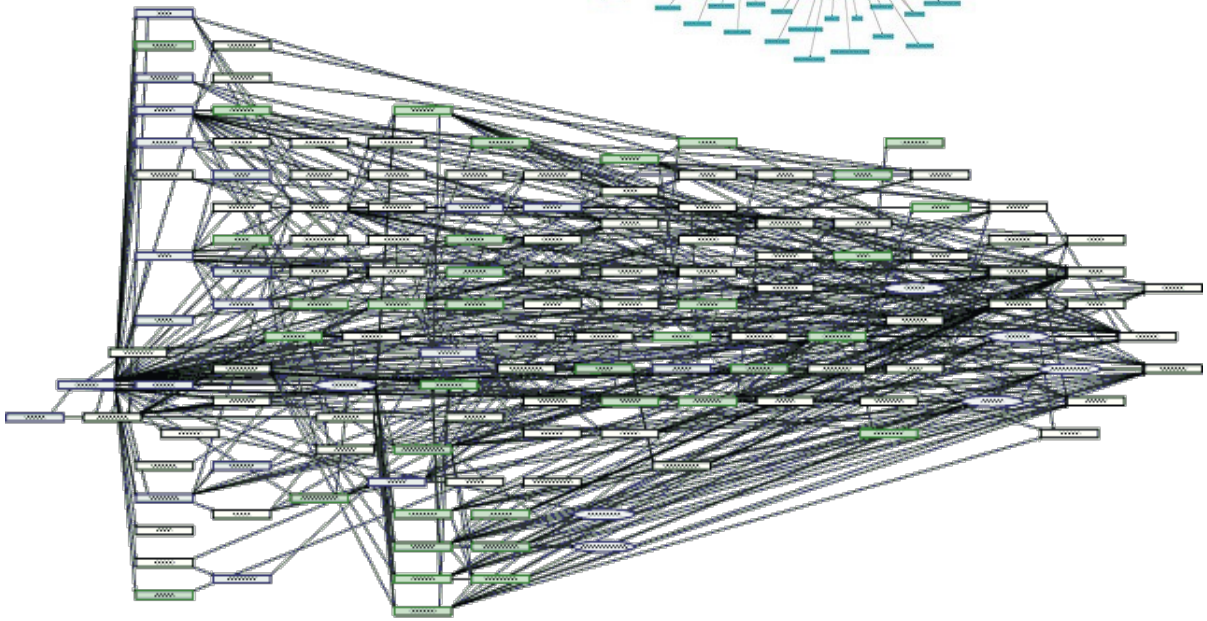
Diagnostic

- l'affichage «ne suit pas » l'interaction
  - ex : gros graphe dont on veut déplacer un noeud



Problème

- trop d'éléments à réafficher à chaque interaction



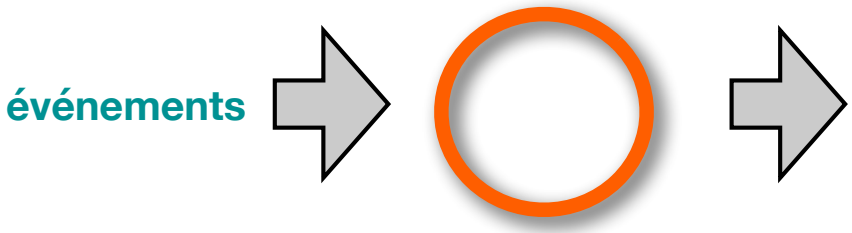
Solution

- réafficher moins d'éléments

Latence

1) Afficher moins de choses dans le temps

- sauter les images intermédiaires : réafficher une fois sur N ou selon un délai
- inconvénient : affichage moins fluide, plus saccadé




```
void mouseMoveEvent(QMouseEvent* e)
    .....
    if (delay < MAX_DELAY) update();
}
```

Ne pas réafficher si le délai est trop long

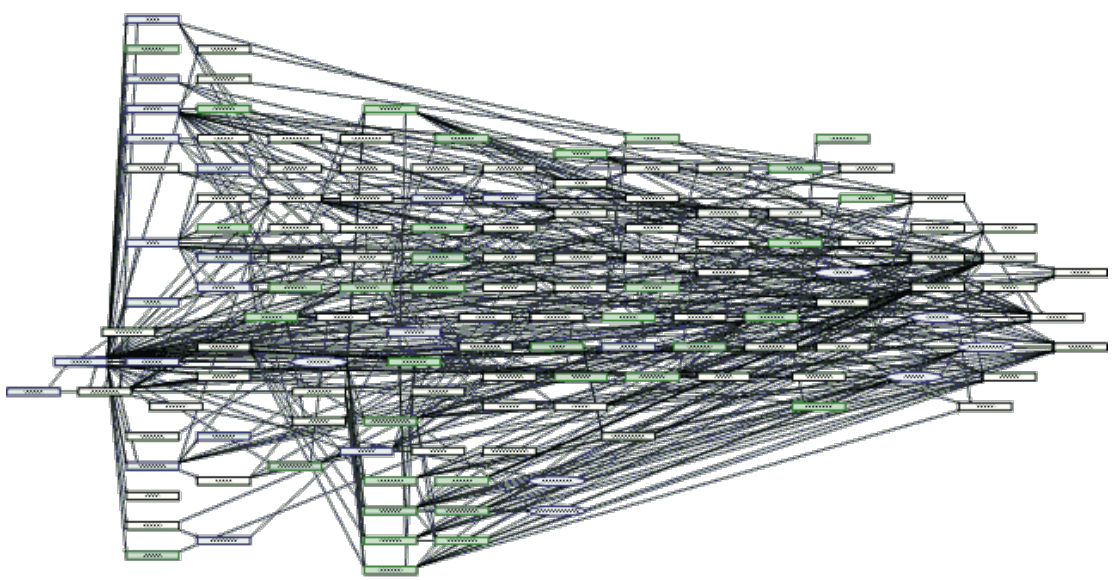
mieux : utiliser des **QTimer** pour demander de réafficher

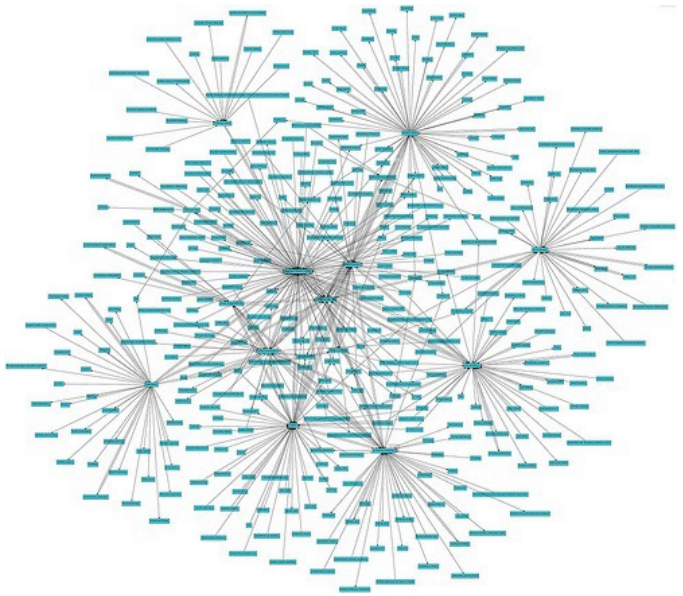
```
void paintEvent(QPaintEvent* e) {
    .....
}
```

Latence

2) Afficher moins de choses dans l'espace

- Ne réafficher que la partie qui change
  - sauver ce qui ne change pas dans une image avant l'interaction
  - à chaque interaction réafficher l'image puis ce qui change par dessus





XOR

Principe

- affichage en mode XOR
  - très efficace pour déplacements interactifs

Afficher 2 fois pour effacer

- 1er affichage: pixel = bg ^ fg
- 2eme affichage: pixel = (bg ^ fg) ^ fg = bg

Problèmes

- couleurs « aléatoires »

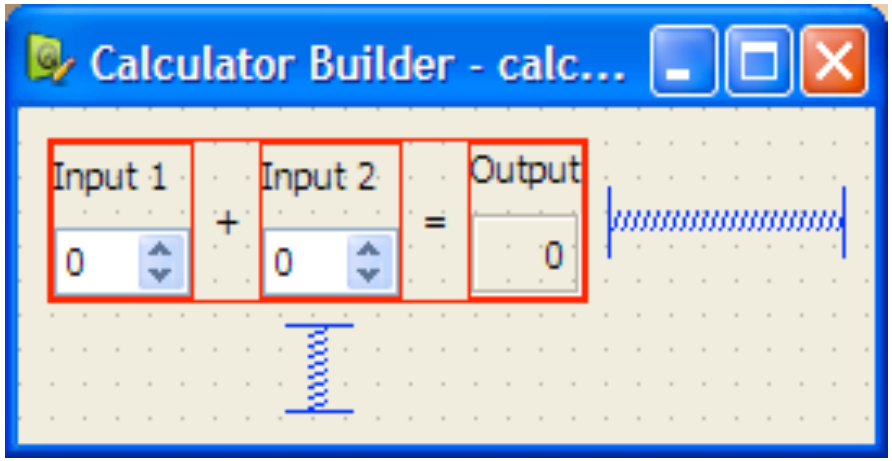
Exemple Java Graphics :

- setColor(Color c1)
- setPaintMode() : dessine avec c1
- setXORMode(Color c2) : échange c1 et c2

Qt Designer

Fichiers

- calculator.pro
- calculator.ui
- calculator.h
- calculator.cpp
- main.cpp



calculator.ui

- fichier XML

Exemple

- vidéo et code .ui

calculator.pro

```
TEMPLATE = app
FORMS = calculator.ui
SOURCES = main.cpp
HEADERS = calculator.h
```

Qt Designer

main.cpp

```
#include "calculator.h"


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    Calculator widget;
    widget.show();
    return app.exec();
}
```

Qt Designer

calculator.h

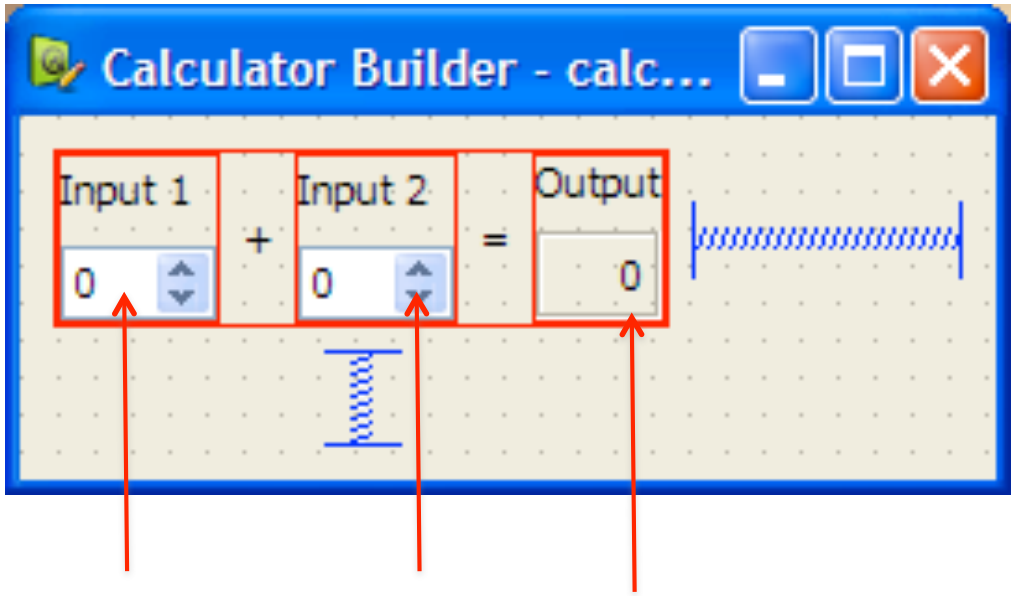
```
#include "ui_calculator.h"


class Calculator : public QWidget {
    Q_OBJECT
public:
    Calculator(QWidget * parent = 0);
private :
    Ui::Calculator * ui;
};
```

calculator.cpp

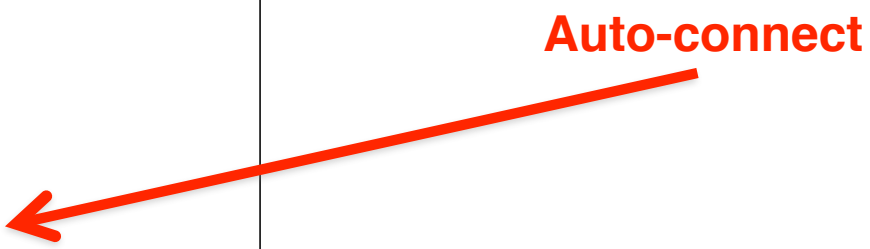
```
#include "calculator.h"

Calculator::Calculator(QWidget * parent) :
    QWidget(parent),
    ui (new Ui::Calculator)
{
    ui->setupUi(this);
    .....
}
```

Lien avec le code source

Comment afficher le résultat du calcul dans le **QLabel** result ?



```
#include "ui_calculator.h"

class Calculator : public QWidget {
    Q_OBJECT
public:
    Calculator(QWidget *parent =0);
private :
    Ui::Calculator * ui;
private slots :
    void on_spinBox1_valueChanged(int value);
    void on_spinBox2_valueChanged(int value);
};
```

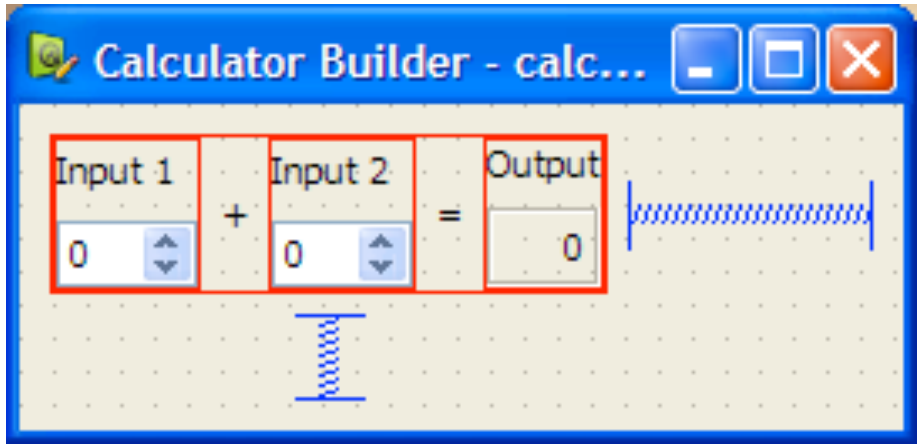



```
#include "calculator.h

void Calculator::on_spinBox1_valueChanged(int value) {
    QString res = QString::number(val)ue;
    // ou: QString res = QString::number(ui->spinBox1->value());

    ui->result->setText(res);        // la variable result a le nom donné au widget dans Qt Designer
                                     // elle est définie dans ui.calculator.h
}
```

Lien avec le code source

On peut lier les signaux des objets créés interactivement avec n'importe quel slot :

- par **auto-connexion** :
  - void on_spinBox1_valueChanged(int)
- ou via le mode Edition Signaux/Slots de **QtCreator**



Variante

Même principe mais en utilisant l'**héritage multiple**

```
#include "ui_calculatorform.h"

Class Calculator : public QWidget, public Ui::CalculatorForm {
    Q_OBJECT
public:
    Calculator(QWidget * parent = 0);
};
```

```
Calculator::Calculator(Qwidget *parent) : Qwidget(parent)  {
    setupUi(this);
}

void Calculator::on_spinBox1_valueChanged(int val) {
    QString res = QString::number(val);
    result->setText(res);      // au lieu de : ui->label3->setText(res);
}
```

Chargement dynamique

```
#include <QtUiTools>

Calculator::Calculator(QWidget *parent) : QWidget(parent) {
    QUiLoader loader;
    QFile file(":/forms/calculator.ui");
    file.open(QFile::ReadOnly);
    QWidget * widget = loader.load(&file, this);
    file.close();
}
```

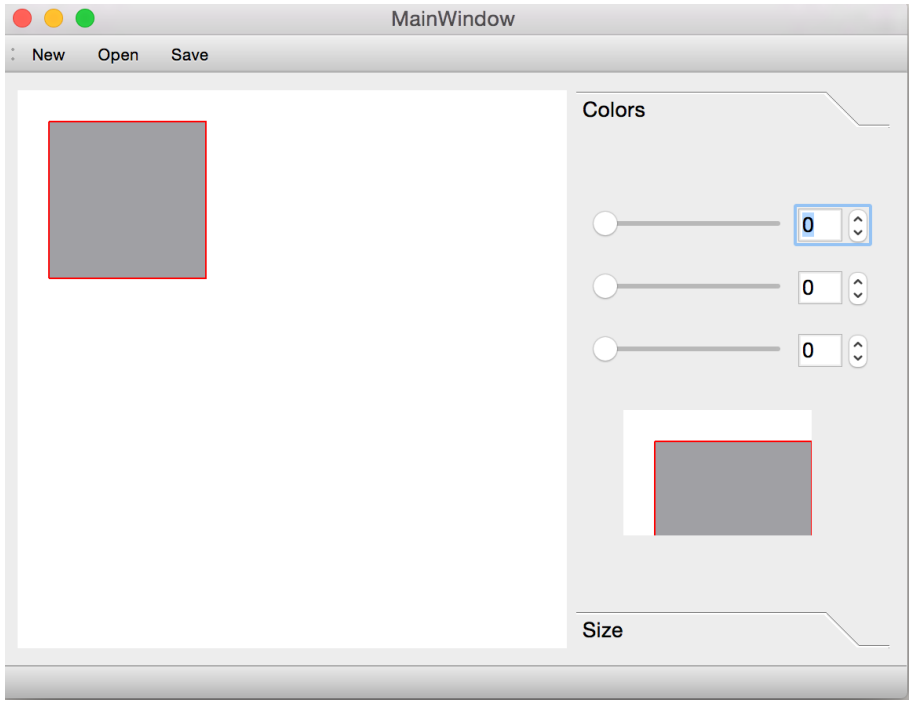
Chargement dynamique

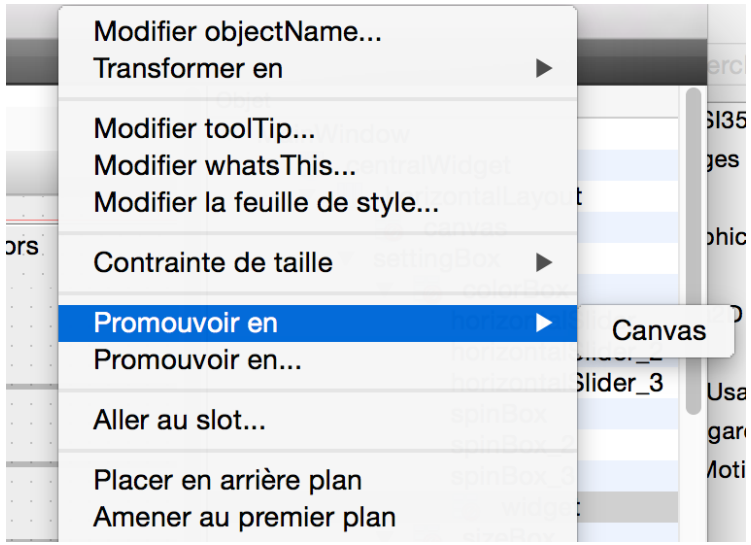
```
class Calculator : public QWidget {
    Q_OBJECT
public:
    Calculator(QWidget *parent = 0);
private:
    QSpinBox * ui_spinBox1;
    QSpinBox * ui_spinBox2;
    QLabel * ui_result;
};

   .....
  ui_spinBox1 = qFindChild<QSpinBox*>(this,  "spinBox1");
  ui_spinBox2 = qFindChild<QSpinBox*>(this, "spinBox2");
  ui_result = qFindChild<QLabel*>(this, "result");
```

« Promotion »

Promote

- permet d'inclure des instances de ses **propres classes** dans **Designer**





Contrôle du layout

Pour chaque QWidget

- **setMinimumSize**(), **setMaximumSize**()
- **setSizePolicy**(QSizePolicy)
- **setSizePolicy**(QSizePolicy::Policy horizontal,
  QSizePolicy::Policy vertical)
- **setHorizontalPolicy**()
- **setVerticalPolicy**()

canvas : Canvas

Propriété	Valeur
Hauteur	373
▼ **sizePolicy**	[Expanding,...
Politique hori...	Expanding
Politique verti...	Expanding
Étirement hori...	0
Étirement vert...	0
▼ **minimumSize**	300 x 0
Largeur	300
Hauteur	0
▼ maximumSize	16777215 x ..

settingBox : QToolBox

Propriété	Valeur
▼ **sizePolicy**	[Fixed, Pref...
Politique hori...	Fixed
Politique verti...	Preferred
Étirement hori...	0
Étirement vert...	0
▼ **minimumSize**	200 x 0
Largeur	200
Hauteur	0
▼ maximumSize	16777215 x ...
Largeur	16777215

Contrôle du layout

QSizePolicy

- **Fixed**: = sizeHint() (taille recommandée)
- **Minimum**: >= sizeHint()
- **Maximum**: <= sizeHint()
- **Preferred**: ~ sizeHint(),
  can be shrunk, can be expanded
- **Expanding**: get as much space as possible,
  can be shrunk
- **MinimumExpanding**: idem, but can't be shrunk
- **Ignored**: idem, ignoring sizeHint()

canvas : Canvas

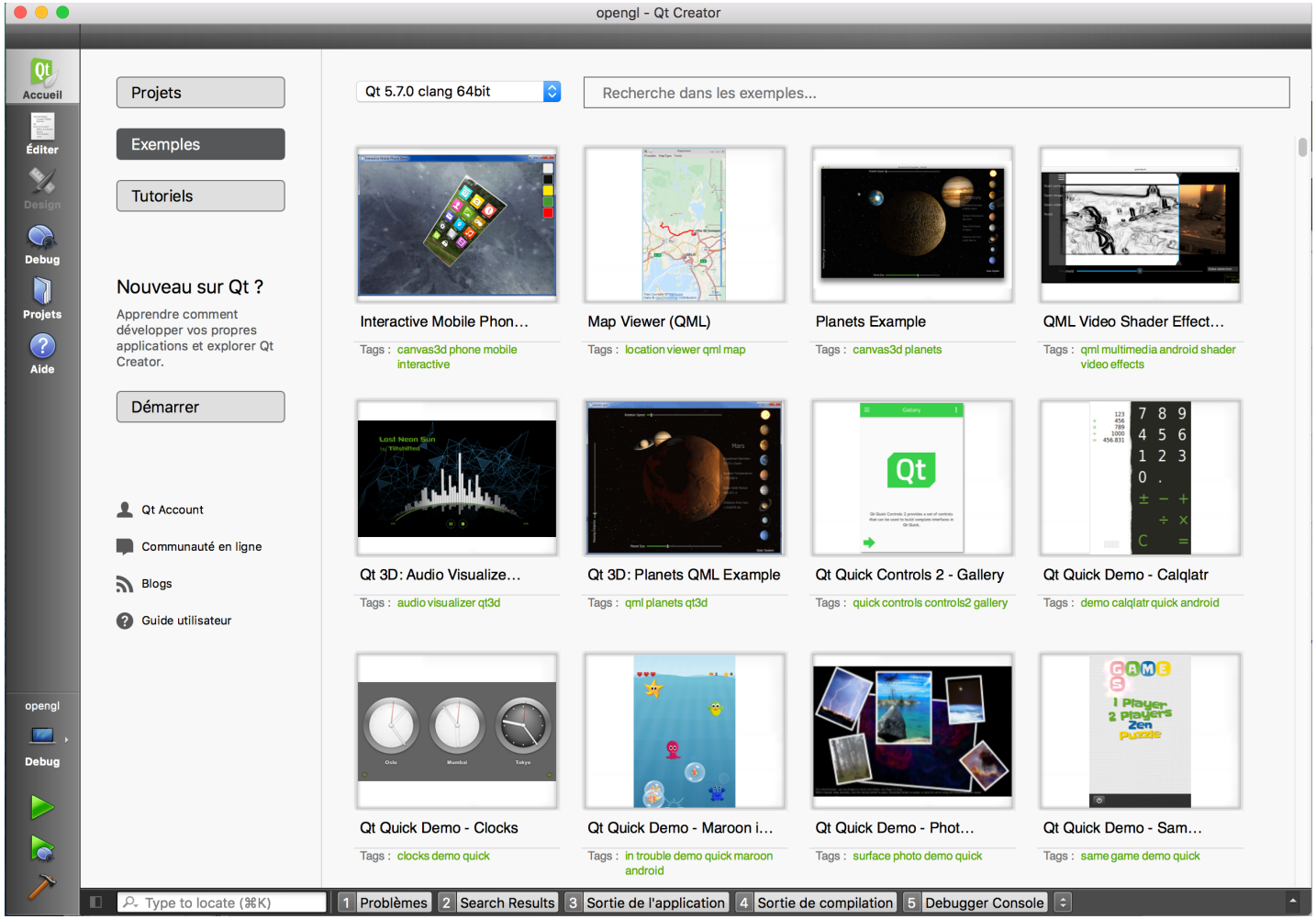
Propriété	Valeur
Hauteur	373
▼ **sizePolicy**	[Expanding,...
Politique hori...	Expanding
Politique verti...	Expanding
Étirement hori...	0
Étirement vert...	0
▼ **minimumSize**	300 x 0
Largeur	300
Hauteur	0
▼ maximumSize	16777215 x ..

settingBox : QToolBox

Propriété	Valeur
▼ **sizePolicy**	[Fixed, Pref...
Politique hori...	Fixed
Politique verti...	Preferred
Étirement hori...	0
Étirement vert...	0
▼ **minimumSize**	200 x 0
Largeur	200
Hauteur	0
▼ maximumSize	16777215 x ...
Largeur	16777215

Exemples et tutoriels

Dans QtCreator > Accueil



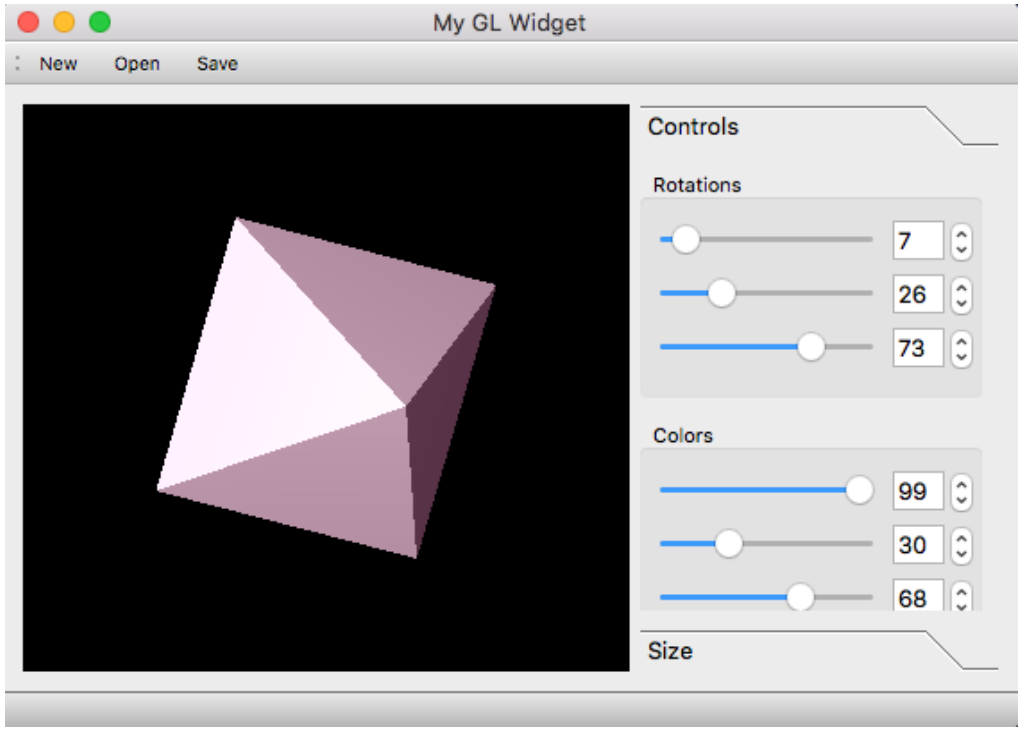
Graphique 3D

Dessin 3D

- via **OpenGL**
- principe : créer une sous-classe de **QOpenGLWidget** et redéfinir :
  - virtual void initializeGL( )
  - virtual void resizeGL(int w, int h)
  - virtual void paintGL( )
- pour demander un réaffichage il faut appeler :
  - update( )
  - (effectué après terminaison d'une fonction de callback)
- Note: **QtGraphicsView** est également compatible avec **OpenGL**

OpenGL : Box3D.h

```
#include <QOpenGLWidget>
#include <QOpenFunctions>

class Box3D : public QOpenGLWidget,  protected QOpenGLFunctions {
    Q_OBJECT
public:
    Box3D(QWidget * parent);

public slots:
    void setXRot(int angle);
    void setYRot(int angle);
    void setZRot(int angle);

signals:
    void xRotChanged(int angle);
    void yRotChanged(int angle);
    void zRotChanged(int angle);

private:
    void initializeGL( );
    void paintGL( );
    void resizeGL(int w, int h);
    int xRot, yRot, zRot;
    GLfloat red, green, blue;
};
```



OpenGL : Box3D.cpp

```
#include "Box3D.h"

Box3D::Box3D(QWidget * parent) : QOpenGLWidget(parent) {
    rotX = rotY = rotZ = 0.;
    red = green = blue = 0.;
}

void Box3D::initializeGL( ) {
   initializeOpenGLFunctions( );     // ne pas oublier !
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    GLfloat lightPosition[4] = {0, 0, 10, 1};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
    GLfloat lightColor[4] = {red, green, blue, 1};
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightColor);
}
```

```
void Box3D::setXRot(int angle) {
    if (angle != xRot) {
        xRot = angle;
        emit xRotChanged(angle);
        update( );
    }
}

void Box3D::setYRot(int angle) {
    if (angle != yRot) {
        yRot = angle;
        emit yRotChanged(angle);
        update( );
    }
}

void Box3D::setZRot(int angle) {
    if (angle != zRot) {
        zRot = angle;
        emit zRotChanged(angle);
        update( );
    }
}
```

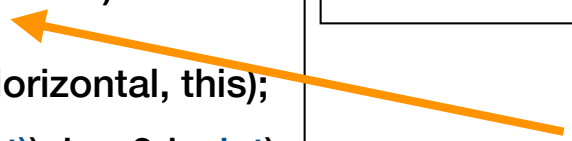
OpenGL : Box3D.cpp

```
void Box3D::paintGL( ) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLoadIdentity();
    glTranslatef(0.0, 0.0, -10.0);
    glRotatef(xRot, 1.0, 0.0, 0.0);
    glRotatef(yRot, 0.0, 1.0, 0.0);
    glRotatef(zRot, 0.0, 0.0, 1.0);

    glBegin(GL_QUADS);
    glNormal3f(0,0,-1);
    glVertex3f(-1,-1,0);
    glVertex3f(-1,1,0);
    glVertex3f(1,1,0);
    glVertex3f(1,-1,0);
    glEnd();

    glBegin(GL_TRIANGLES);
    glNormal3f(0,-1,0.707);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(-1,-1,0);
    glColor3f(1.0f, 1.0f, 0.0f);
    glVertex3f(1,-1,0);
    glColor3f(1.0f, 0.0f, 1.0f);
    glVertex3f(0,0,1.2);
    glEnd();
     ....
    glBegin(GL_TRIANGLES);
     glNormal3f(1,0, 0.707);
     glVertex3f(1,-1,0);
     glVertex3f(1,1,0);
     glVertex3f(0,0,1.2);
     glEnd();
     glBegin(GL_TRIANGLES);
     glNormal3f(0,1,0.707);
     glVertex3f(1,1,0);
     glVertex3f(-1,1,0);
     glVertex3f(0,0,1.2);
     glEnd();
     glBegin(GL_TRIANGLES);
     glNormal3f(-1,0,0.707);
     glVertex3f(-1,1,0);
     glVertex3f(-1,-1,0);
     glVertex3f(0,0,1.2);
     glEnd();
}

void Box3D::resizeGL(int w, int h) {
    int side = qMin(w, h);
    glViewport((w - side) / 2, (h - side) / 2, side, side);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2, +2, -2, +2, 1.0, 15.0);
    glMatrixMode(GL_MODELVIEW);
}
```

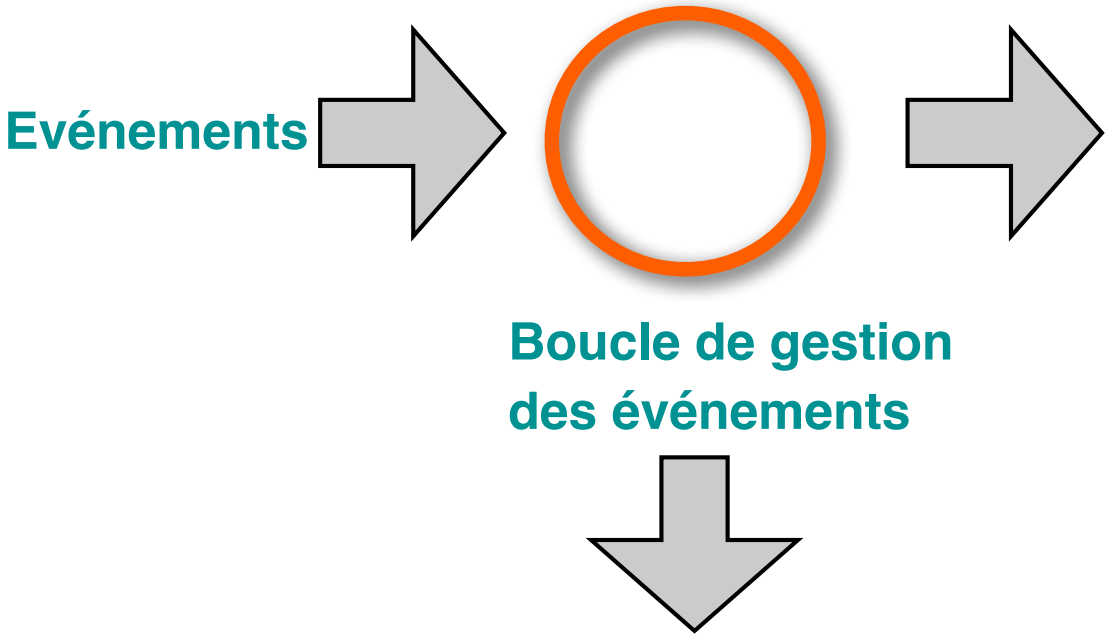
OpenGL : main et MainWindow

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent)
{
    ...
    Box3D box3d = new Box3D(parent);
    setCentralWidget(box3d);

    createSlider(box3d, SLOT(setXRot(int)));
    createSlider(box3d, SLOT(setYRot(int)));
    createSlider(box3d, SLOT(setZRot(int)));
    ...
}

void MainWindow::createSlider(Box3D * box3d,
                              const char * slot)
{
    QSlider * slider = new QSlider(QSlider::Horizontal, this);
    connect(slider, SIGNAL(valueChanged(int)), box3d, slot);
}
```

```
int main(int argc, char **argv)  {
    QApplication app(argc, argv);

    QSurfaceFormat format;
    format.setDepthBufferSize(24);
    format.setStencilBufferSize(8);
format.setProfile(QSurfaceFormat::CoreProfile);
    QSurfaceFormat::setDefaultFormat(format);

    MainWindow mainwin;
    mainwin.setWindowTitle("OpenGL Box");
    mainwin.show( );
    return app.exec( );
}
```



Concurrence




```
void mouseXxxxEvent(QMouseEvent*
e)
    ......
    update();
}
```

ou

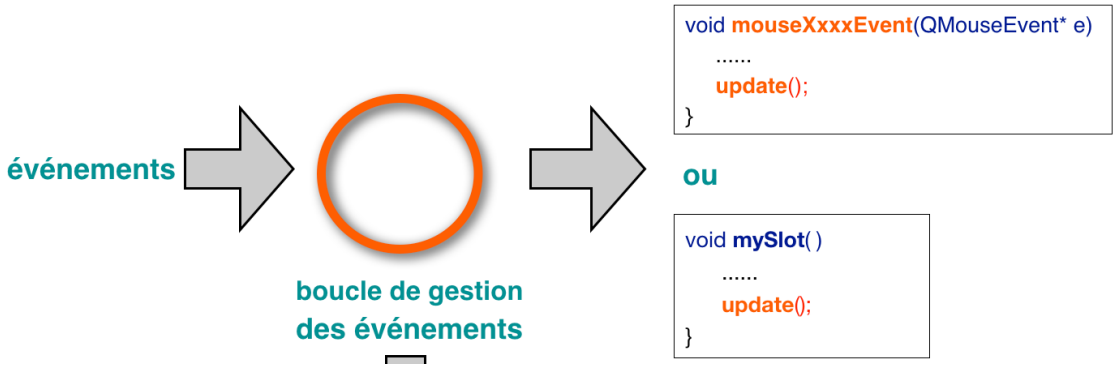
```
void mySlot( )
    ......
    update();
}
```

```
void paintEvent(QPaintEvent* e) {
    QWidget::paintEvent(e);
    QPainter.painter(this);
     ......
}
```

Concurrence

Boucle de gestion des événements

- les événements sont traités **séquentiellement**

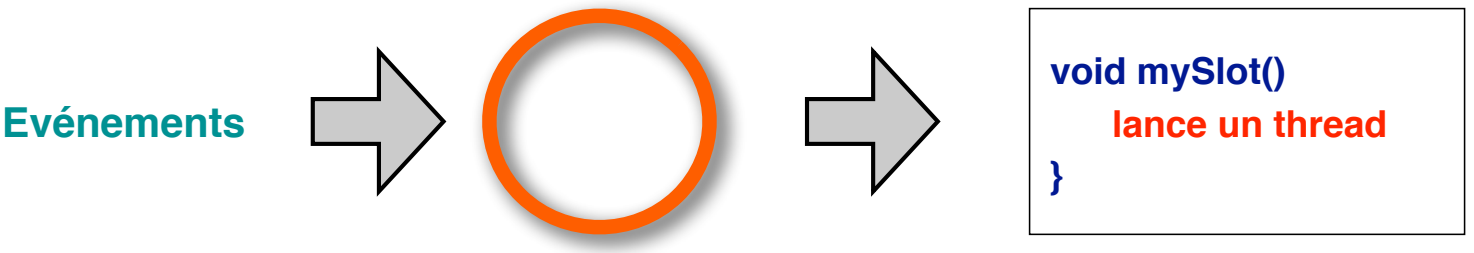


Problème

- boucle **bloquée** si un slot attend longtemps des données :
  - les événements sont **mis en attente** dans une pile
  - typique avec les **applications réseau** (sockets, etc.)
  - => **blocage** de l'interface, **plus de rafraîchissement**

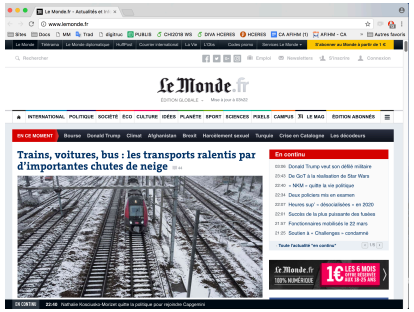
Solution ?

Threads



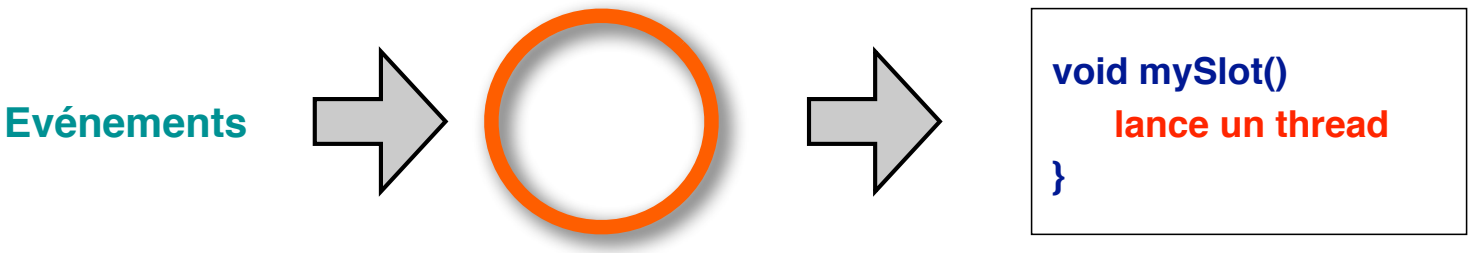
Solutions

- 1) le slot **lance un thread et retourne immédiatement**
  - le thread se termine une fois l'action réalisée
  - exple : charger une page web



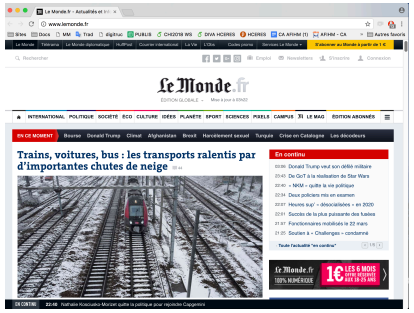
- 2) un thread **tourne en permanence**
  - et **notifie** régulièrement l'application
  - exple : récupérer des données d'une Kinect

Threads



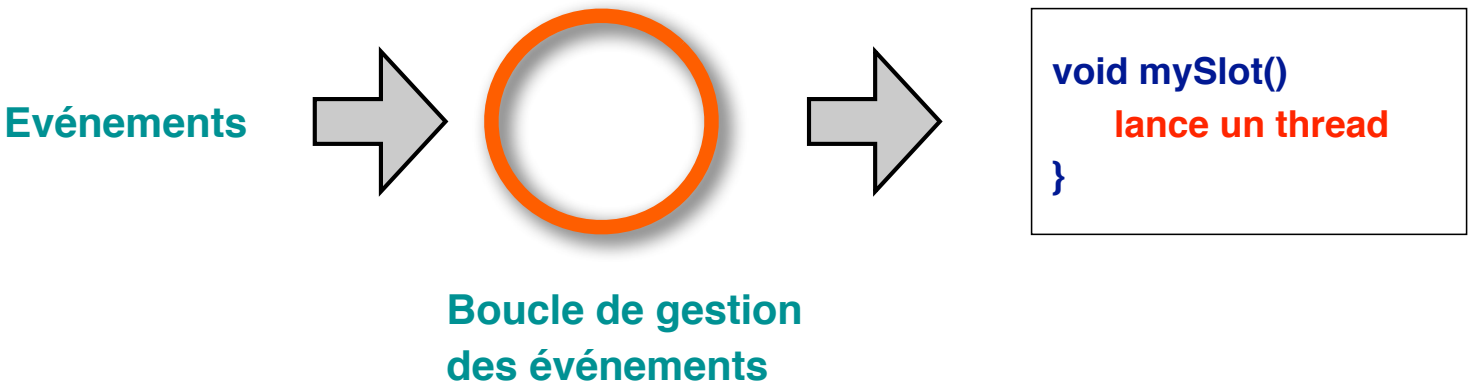
Dans les deux cas

- le thread doit **notifier l'interface**
  - il ne doit **pas réafficher directement !**
  - ni modifier les widgets



- car l'interface graphique "vit" dans le thread principal
- même problème en Java Swing et autres toolkits

Threads



QThread

- on peut s'en servir diverses façons (ici la plus simple)
- principe : sa méthode **run**( ) s'exécute **dans un nouveau thread**
  - elle lance sa **propre boucle** de gestion des événements
  - on peut la **redéfinir**

Threads

```
class DTracker : public QThread {
    Q_OBJECT
public:
    DTracker(const std::string& hostname, int port);

signals:
    void dataReady(const QString& data);

private:
    /// run() is executed in a new thread and gets data from a socket.
    void run( ) {
        bool ok = true;
        QString data;

        while (ok) {
            data = getData();
            emit dataReady(data);
        }
    }
    ....
};
```

Pour lancer le traitement

- appeler la méthode **start( )**

Threads

Pour notifier l'interface : connect()

- **connect**() peut gérer une **queue de connexions** (voir sa doc)
- exemple : dans le **thread principal** (ex : dans le constructeur de **MainWindow**)

```
// dans le thread principal
    slot  showData()
// faire :
    dtracker = new DTracker(hostname, port);
    connect( dtracker, SIGNAL(dataReady(QString)), this, SLOT(showData(QString)) );
```

=> le slot **showData**() sera exécuté dans le **thread principal**

Alternatives

- **QTimer**
- **QSocketNotifier** etc.
- QEventLoop::processEvents()
- framework **QtConcurrent**

Timers

```
QTimer * timer = new QTimer( );
connect(timer, SIGNAL(timeout( )), widget, SLOT(doSomething( )));
timer->start(delay)
…
timer->stop();
```

Le signal **timeout**() sera activé **indéfiniment**, chaque fois **après ce délai**

Remarques

- **setSingleShot**(true) : **une seule** activation (le timer est auto-détruit)
- délai **nul** => slot activé **dès que possible**
- également utile pour les **animations**

Propriétés et animation

Propriétés

```
s1->assignProperty( label, "text", "In state s1" );
s1->assignProperty( button, "geometry", QRectF(0, 0, 50, 50) );
```

Base des animations

```
s1->assignProperty( button, "geometry", QRectF(0, 0, 50, 50) );
s2->assignProperty( button, "geometry", QRectF(0, 0, 100, 100) );
s1->addTransition( button, SIGNAL(clicked( )), s2)
                    ->addAnimation( new QPropertyAnimation( button, "geometry" ) );
```

Voir aussi QML

- langage **déclaratif** en **JavaScript**
- permet davantage de possibilités pour **l'animation**

Gestes

Créer de nouveaux gestes et les reconnaître

C’est possible en sous-classant :

- **Questure**
- **QGestureRecognizer**

Voir cours et TP suivants

Machines à états et Statecharts

Machines à états finis

Example : rubber banding (repris de Scott Hudson - CMU)

```
Accept the press for endpoint p1;
P2 = P1;
Draw line P1-P2;
Repeat
     Erase line P1-P2;
     P2 = current_position();
     Draw line P1-P2;
Until release event;
Act on line input;
```

Problème ?

Machines à états finis

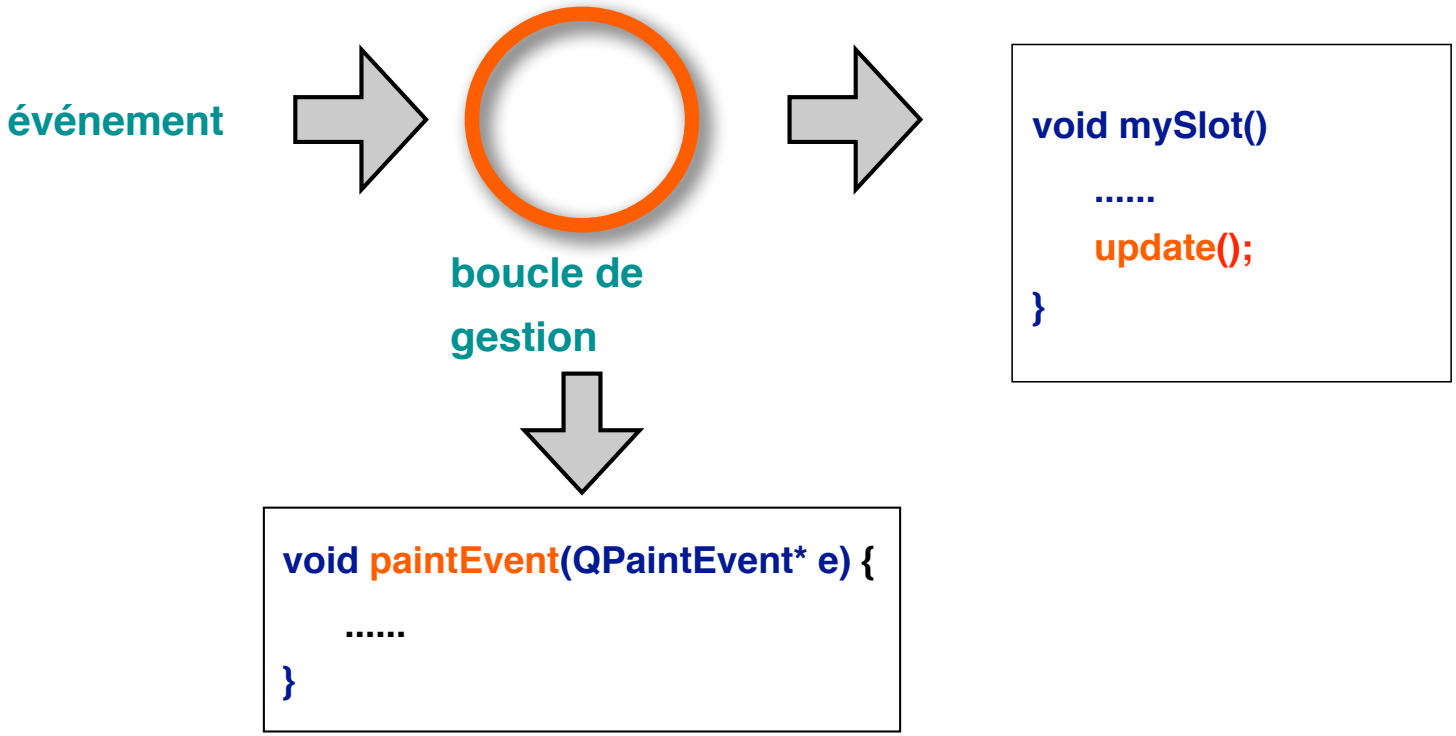
Rubber banding

```
Accept the press for endpoint p1;
P2 = P1;
Draw line P1-P2;
Repeat
      Erase line P1-P2;
      P2 = current_position();
      Draw line P1-P2;
Until release event;
Act on line input;
```

Problème

- pas compatible avec la gestion événementielle

Gestion des événements



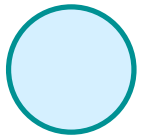
Cycle événement / réaffichage

- on ne doit **pas bloquer** ce cycle **ni ignorer** les (autres) événements

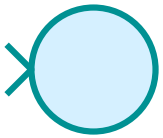
Machines à états

Solution appropriée pour « maintenir l'état » !

- simple et efficace pour **modéliser les comportements**
- évite les «**spaghettis de callbacks**», la multiplication des **variables d'état** et ... **les erreurs !**
- passage facile d'une **représentation visuelle** au **code source**
- divers outils : **UML**, **QStateMachine**, etc.



Etat



Etat de départ

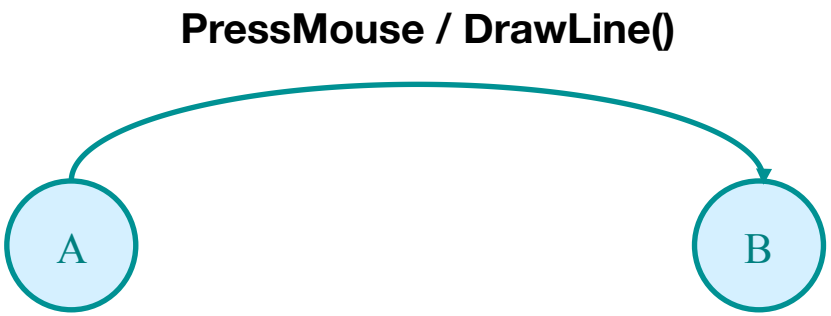
Etat final

En IHM généralement pas d'état **final** : on revient à **l'état initial**

Machines à états

Transitions

- Représentées par des **arcs** avec un label : **événement / action**
- Si on est dans l'**état A** et cet **événement** se produit :
  - l'**action** est effectuée
  - puis on passe dans l'**état B**



Remarque

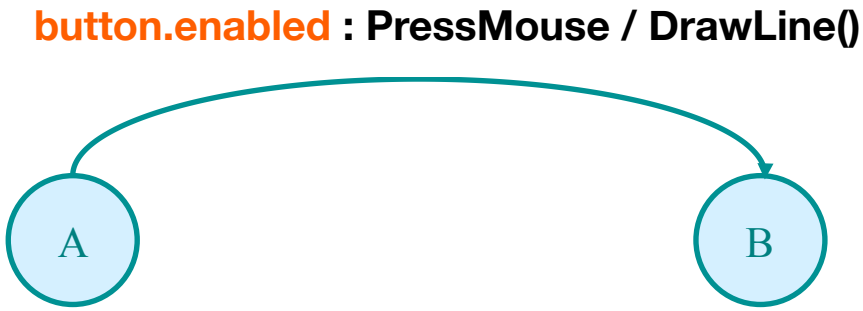
– par défaut avec **Qt** les **actions** sont sur les **états** :
  - quand on **entre** ou on **sort** de l'état

- on peut aussi mettre les **actions** sur les **transitions**

Machines à états

Gardes

- **conditions** de déclenchement
- notation : **prédicat** **:** **événement** **/** **action**



Evénements

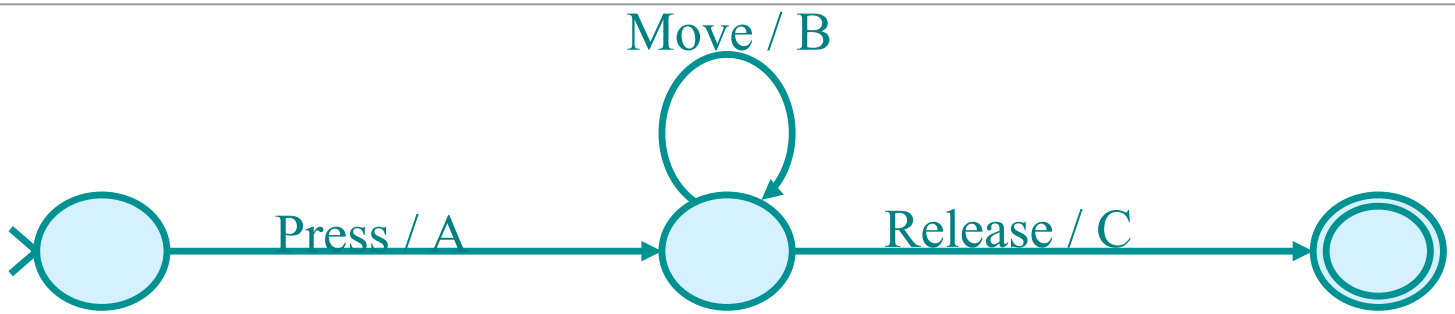
- peuvent être des **timeouts** (cf. **QTimer**)
- ou être **générés par le programme**

Machines à états

Retour à notre «bande élastique»

```
Accept the press for endpoint p1;
  A  P2 = P1;
     Draw line P1-P2;
Repeat
     Erase line P1-P2;
  B  P2 = current_position();
     Draw line P1-P2;
Until release event;
  C  Act on line input;
```

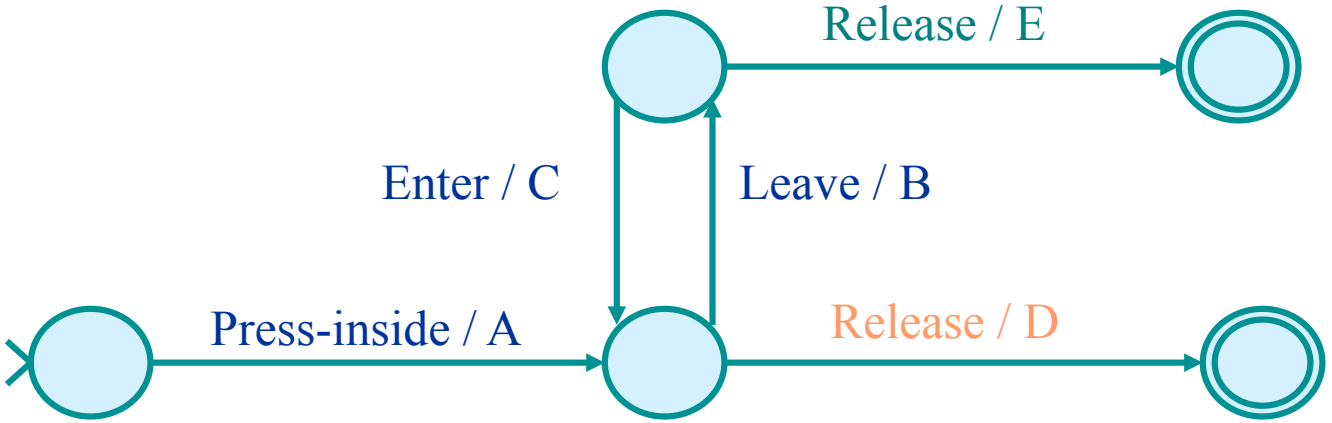
Machines à états




```
Accept the press for endpoint p1;
  A  P2 = P1;
     Draw line P1-P2;
Repeat
     Erase line P1-P2;
  B  P2 = current_position();
     Draw line P1-P2;
Until release event;
  C  Act on line input;
```

Machines à états

Autre exemple : bouton



A: highlight button
B: unhighlight button
C: highlight button
D: do button action
E: do nothing

Machines à états

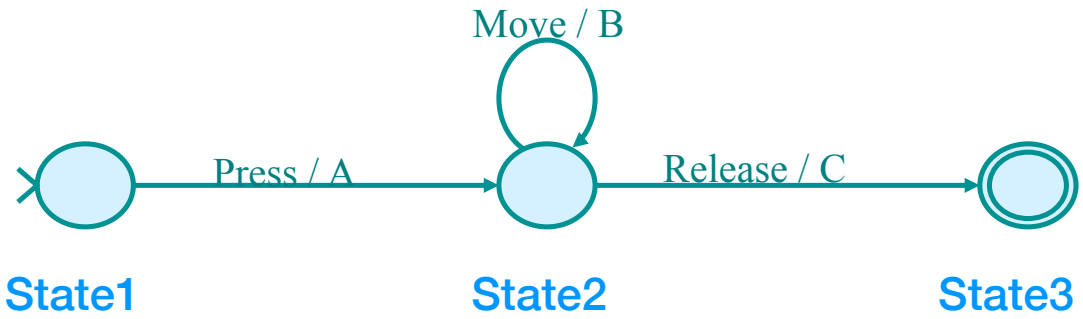
Implémentation

- doit être compatible avec la **boucle de gestion des événements**
- codage « en dur » ou via des **tables d'états / transitions**
- préférer l'utilisation d'**outils existants**

```
state = start_state;

while (true) {
    raw_event = wait_for_event();
    event = transform_event(raw_event);
    state = fsm_transition(state, event);
}
```

Machines à états

Implémentation «en dur»



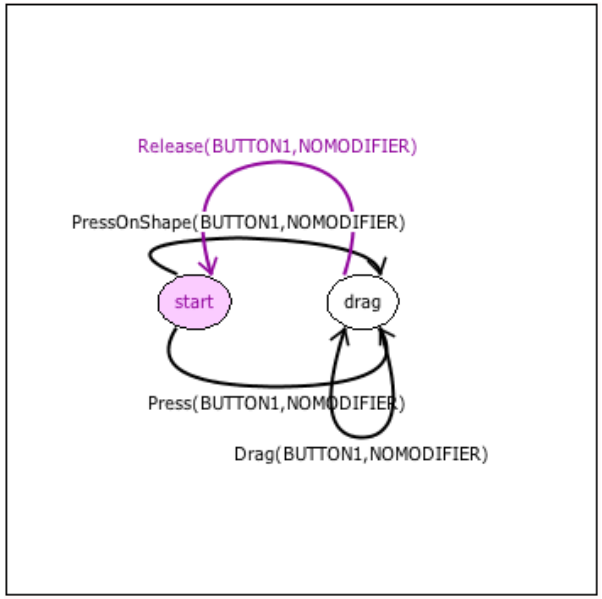

```
state = start_state;

while (true) {
      raw_event = wait_for_event();
      event = transform_event(raw_event);
      state = fsm_transition(state, event);
}
```

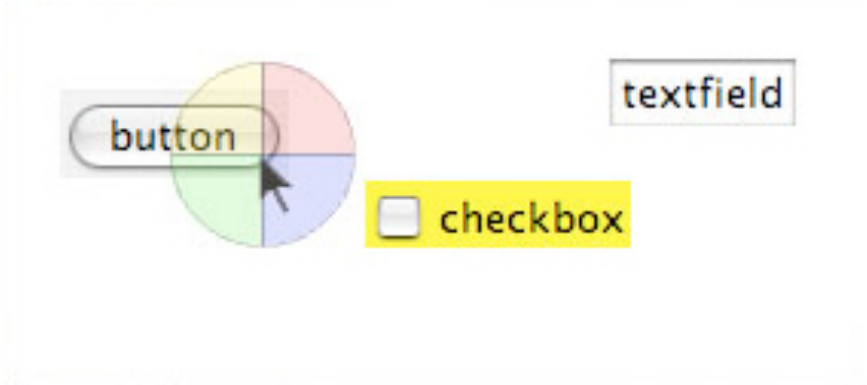
```
State fsm_transition (state, event) {

   switch (state) {
    case State2:
      switch (event.kind) {
        case MouseMove:
            actionB ( );
            state = State2;

        case MouseRelease:
            actionC ( )
           state = State3;
       }
       break;

    case State1:
      switch (event.kind) {
        case ...
      }
   }

  return state;
}
```

SwingStates

Une implementation Java

- Caroline Appert, LRI
- http://swingstates.sourceforge.net/
- classe **Canvas** qui facilite le dessin interactif
- exemples d'interactions avancées





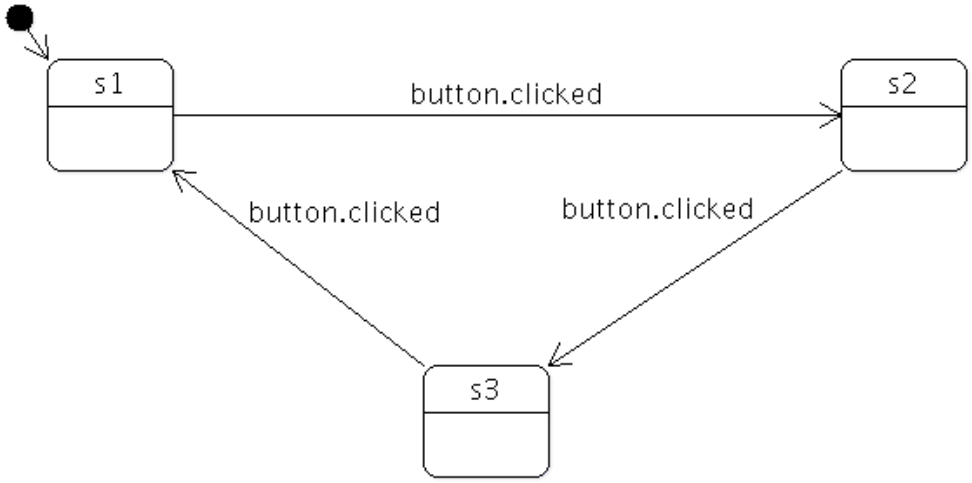
Qt State Machine

Qt : State Machine Framework

- modèle **hiérarchique** : **Statecharts**
- basés sur **SCXML**
- permet entre autres :
  - **groupes d'états** (hiérarchies)
  - états **parallèles** : pour éviter l'explosion combinatoire
  - états **historiques** : pour sauver et restaurer l'état courant
  - d'injecter ses **propres** événements
- sert aussi pour les **animations**

Bouton à trois états

```
// créer la machine à états
QStateMachine * mac = new QStateMachine( );

QState *s1 = new QState( );
QState *s2 = new QState( );
QState *s3 = new QState( );
mac->addState(s1);
mac->addState(s2);
mac->addState(s3);
s1->addTransition(button, SIGNAL(clicked( )), s2);      // transition de s1 à s2 si on clique le bouton
s2->addTransition(button, SIGNAL(clicked( )), s3);
s3->addTransition(button, SIGNAL(clicked( )), s1);

                                                        // les actions sont sur les états
QObject::connect(s3, SIGNAL(entered( )), window, SLOT(showMaximized( )));
QObject::connect(s3, SIGNAL(exited( )), window, SLOT(showMinimized( )));

mac->setInitialState(s1);          // ne pas oublier !
mac->start( );                     // lance la machine
```



Hiérarchies

Principe

- les états enfants **héritent des transitions** des états parents

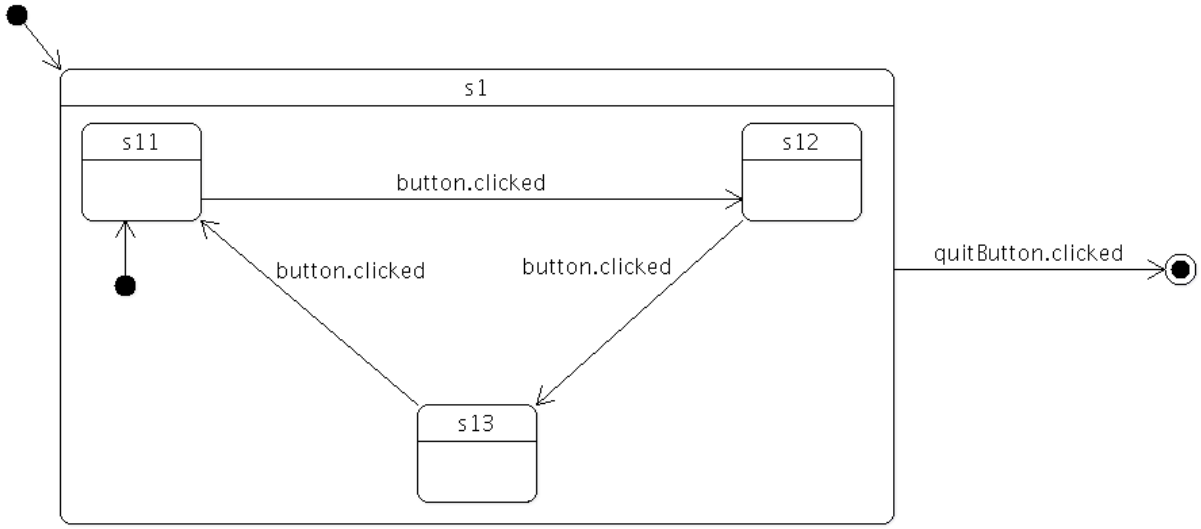
Avantages

- permettent de **simplifier** (grandement !) les schémas

Exemple (page suivante)

- on rajoute à l'exemple précédent un bouton **quitButton** qui **ferme** l'application
- les états **héritent** de cette transition

Hiérarchies

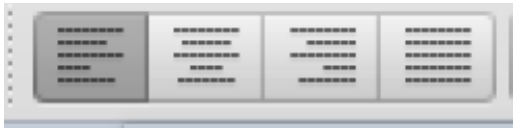



```
// s11, s12, s13 sont groupés dans s1
QState *s1 = new QState( );
mac->addState(s1);
QState *s11 = new QState(s1);
QState *s12 = new QState(s1);
QState *s13 = new QState(s1);

s11->addTransition(button, SIGNAL(clicked( )), s12);
s12->addTransition(button, SIGNAL(clicked( )), s13);
s13->addTransition(button, SIGNAL(clicked( )), s11);
s1->setInitialState(s11);

QFinalState *s2 = new QFinalState( );       // s2 est un étal final
mac->addState(s2);
                                            // les enfants de s1 héritent de la transition s1 -> s2
s1->addTransition(quitButton, SIGNAL(clicked( )), s2);
                                            // le signal finished() est émis quand on atteint l'état final
connect(mac, SIGNAL(finished( )), QApplication::instance( ), SLOT(quit( )));
mac->setInitialState(s1);
mac->start( );
```

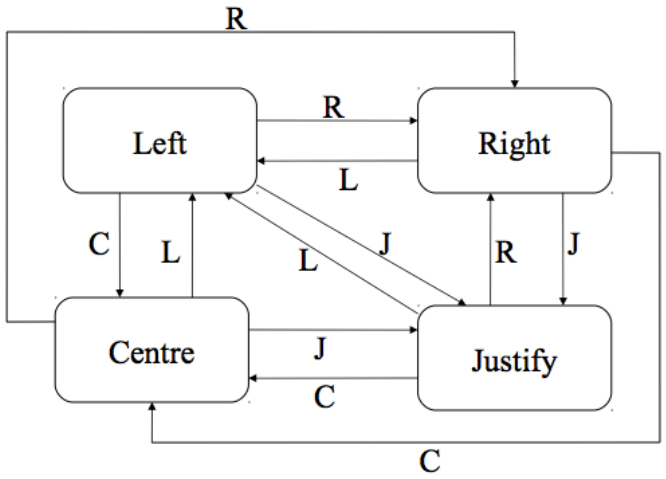
Hiérarchies

Second exemple

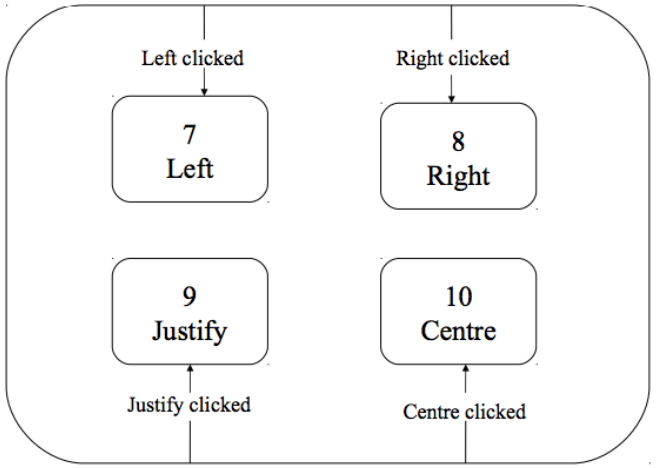
- 4 radio boutons => 4 **états exclusifs**
- noter :
  - transitions de l'état parent **héritées** par les enfants
  - le **parent** est **toujours actif**
- l'**héritage** **réduit fortement** le nombre de transitions à écrire



aligner le texte



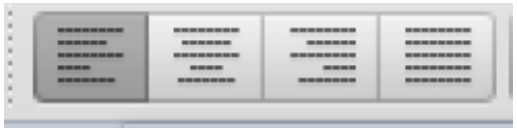
solution « plate »



solution hiérarchique

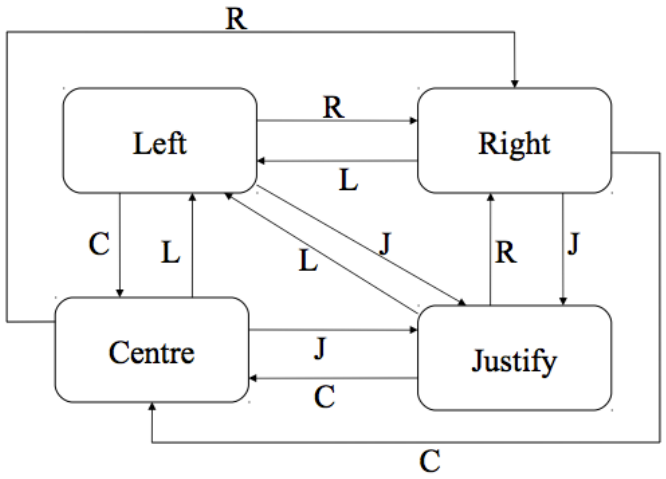
Hiérarchies

Second exemple

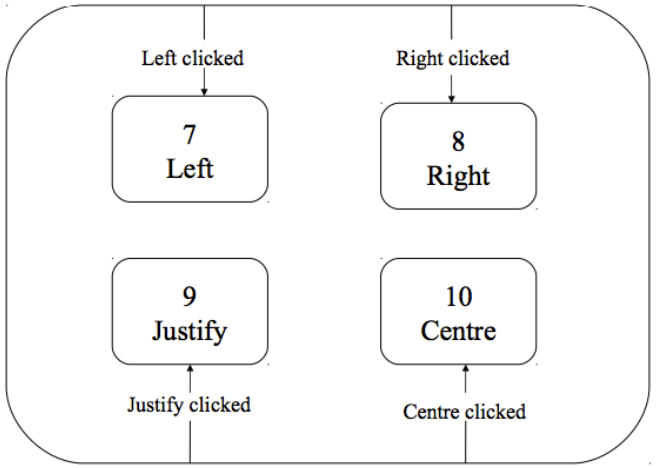


aligner le texte

- 4 radio boutons => 4 **états exclusifs**
- noter :
  - transitions de l'état parent **héritées** par les enfants
  - le **parent** est **toujours actif**
- l'**héritage** **réduit fortement** le nombre de transitions à écrire



solution « plate »

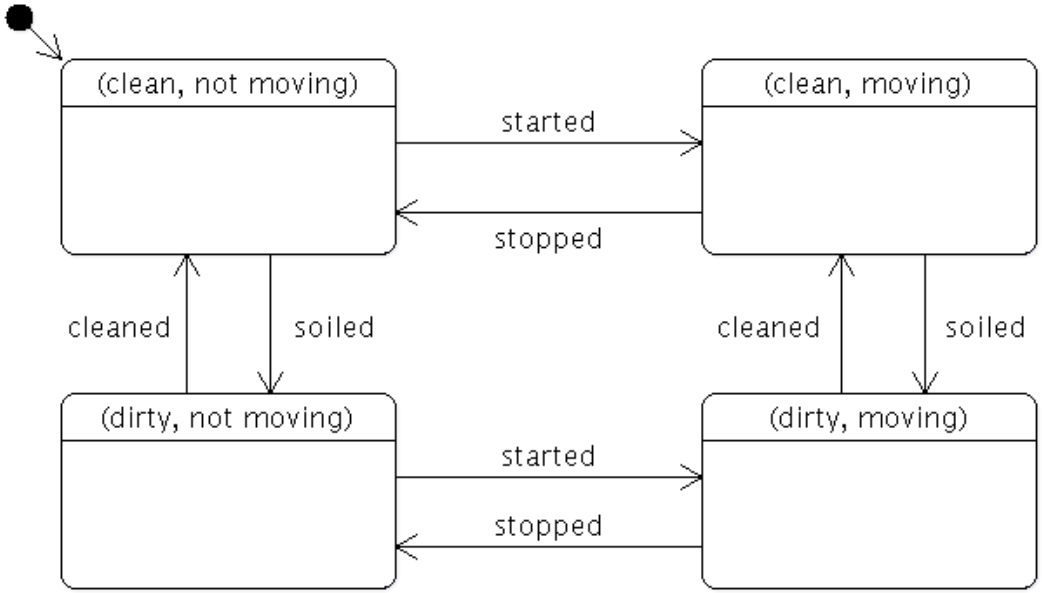


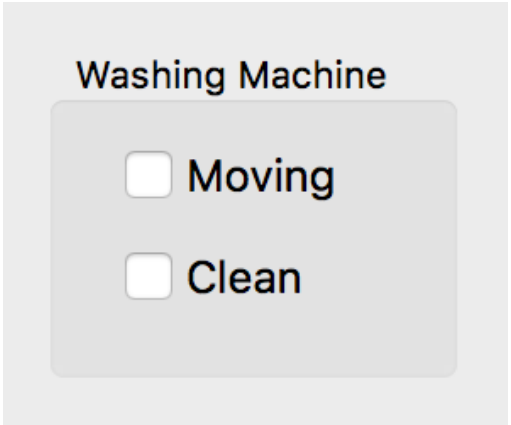
solution hiérarchique

Etats parallèles

Exemple

- 2 check boxes **indépendantes**
- chaque check box a **2 états** mais il faut considérer **toutes les combinaisons**
- => ca "**explose**" rapidement !

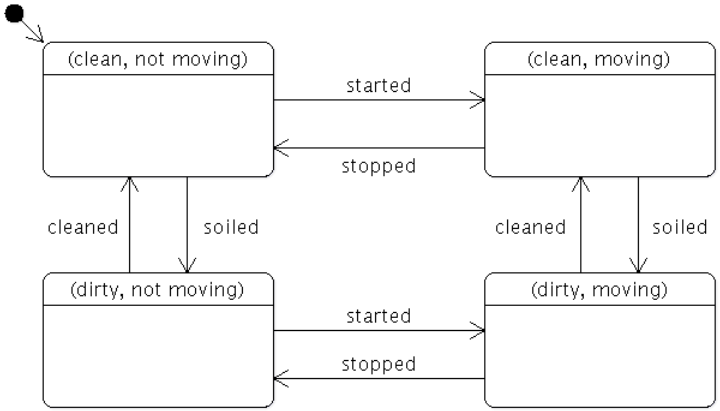




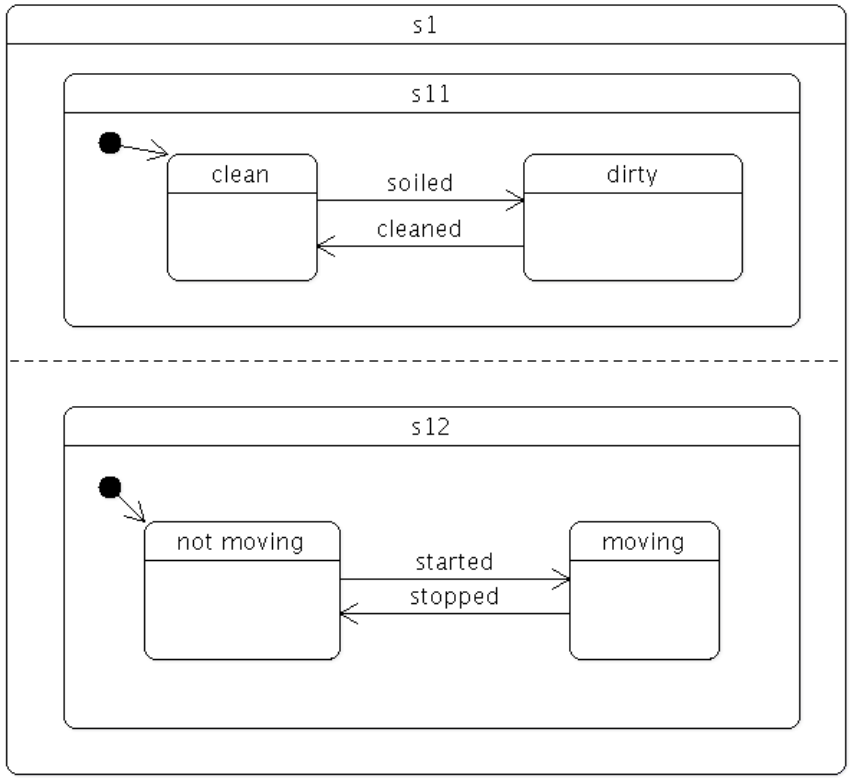
Etats parallèles

Solution : états parallèles

- évitent l'**explosion combinatoire**
- **plusieurs états** sont actifs **en même temps**



solution « plate »




```
QState *s1 = new QState(QState::ParallelStates);

// on peut être à la fois dans s11 et dans s12
QState *s11 = new QState(s1);
QState *s12 = new QState(s1);
```

solution hiérarchique

Types de transitions

Pour les Signaux

- **QSignalTransition** : c'est ce qu'on a utilisé jusqu'à présent via :

```
s1->addTransition (button, SIGNAL(clicked( )), s2);
```

Pour les Evénements

- **QEventTransition, QKeyEventTransition, QMouseEventTransition**
- **pas** de fonctions utilitaires, il faut instancier ces objets

On peut aussi créer ses *propres* transitions

- en dérivant **QAbstractTransition**
- pour gérer des **événements utilisateur :**
  - définis par le **programmeur** et envoyés par le **programme**

Transitions.h

Header « maison »

- **fonctions simplifiées** pour les cas courants
- http://www.telecom-paristech.fr/~elc/qt/Transitions.h

Avantages

- l'action est sur la **transition**, pas sur l'**état** (parfois plus intuitif)
- fonctions simplifiées pour les **événements** (clavier, souris, utilisateur, etc.)
- chaque fonction a une **variante** qui permet d'ajouter une **action** (= un slot)

Transitions sur les **signaux**

Qt


```
state1->addTransition (sender, signal, state2);
```

exemple :

```
state1->addTransition (button, SIGNAL(clicked( )), state2);
QObject::connect (state2, SIGNAL(entered( )), this, SLOT(dolt( )));    // action sur l'état
```

Transitions.h

```
addTrans (state1, state2, sender, signal);
```

avec une **action** sur la **transition**:

```
addTrans (state1, state2, sender, signal, receiver, slot);
```

exemple :

```
addTrans (state1, state2, button, SIGNAL(clicked( )), this, SLOT(dolt( )));
```

Transitions sur les **événements**

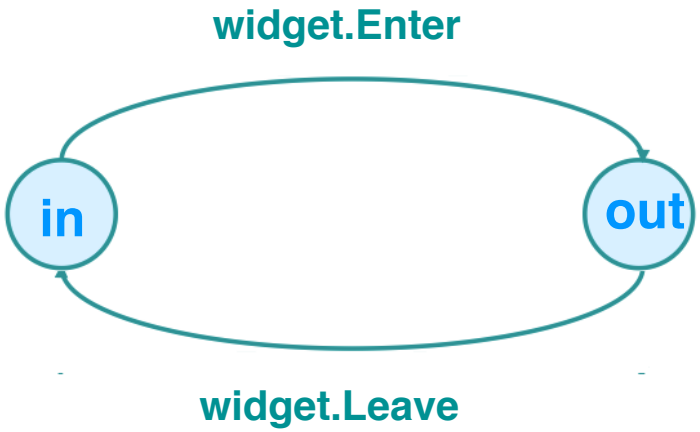
Evénements quelconques

```
addEventTrans (state1, state2, sender, eventType);
addEventTrans (state1, state2, sender, eventType, receiver, slot);
```

eventType est un **type d'événement Qt** (type : QEvent::Type)

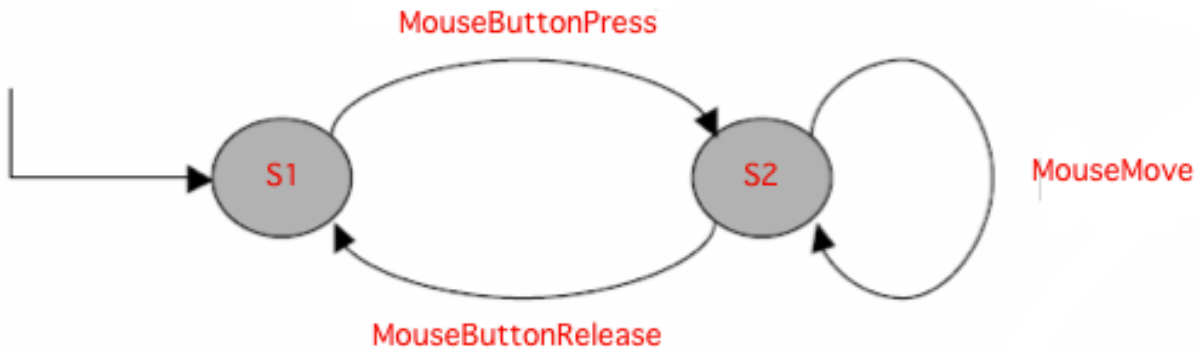
exemple:

```
addEventTrans (out, in, widget, QEvent::Enter);
addEventTrans (in, out, widget, QEvent::Leave);
```



Evénements **souris**

Argument supplémentaire pour spécifier le **bouton de la souris**

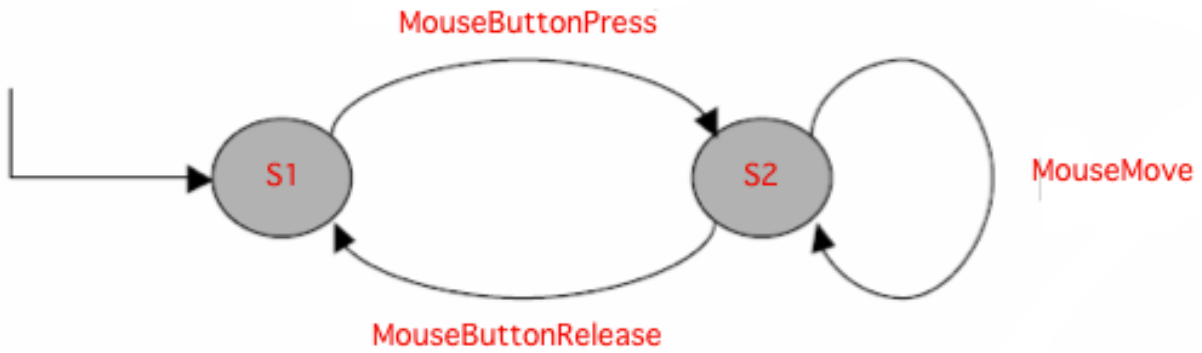


dessin à main levée

```
// transition de s1 vers s2 quand le bouton gauche de la souris est pressé sur «canvas»
addMouseTrans (s1, s2,
               canvas, QEvent::MouseButtonPress, Qt::LeftButton,
               this, SLOT(newLine( )) );

addMouseTrans (s2, s2,
               canvas, QEvent::MouseMove, Qt::NoButton,      // attention : NoButton !
               this, SLOT(adjustLine( )) );

addMouseTrans (s2, s1,
               canvas, QEvent::MouseButtonRelease, Qt::LeftButton );
```

Evénements souris

On peut aussi spécifier
plusieurs boutons
et les **modificateurs clavier**




```
// il faut appuyer LeftButton ou RightButton
addMouseTrans (s1, s2,
                    canvas, QEvent::MouseButtonPress, Qt::LeftButton | Qt::RightButton,
                    this, SLOT(newLine( )) );
// double clic
addMouseTrans (s1, s2,  canvas, QEvent::MouseButtonDblClick, Qt::LeftButton);

// il faut appuyer SHIFT et LeftButton
addMouseTrans (s1, s2, canvas, QEvent::MouseButtonPress, Qt::LeftButton)
->setModifierMask(Qt::ShiftModifier);
```

Position du curseur

```
addMouseTrans (s1, s2,
               canvas, QEvent::MouseButtonPress, Qt::LeftButton,
               this, SLOT(newLine( )) );

addMouseTrans (s2, s2,
               canvas, QEvent::MouseMove, Qt::NoButton,
               this, SLOT(adjustLine( )) );
```

Problème

Les slots newLine() et adjustLine() n'ont pas de paramètre **QEvent** :

=> comment peuvent-ils accéder à la **position du curseur** ?

Position du curseur

Solution 1

```
QPoint cursorPos(QWidget* w)  { return w->mapFromGlobal(QCursor::pos( )); }
```

- renvoie la **position du curseur** relativement à un widget
- petit décalage éventuel avec les **systèmes asynchrones** (X11 en réseau)

Solution 2

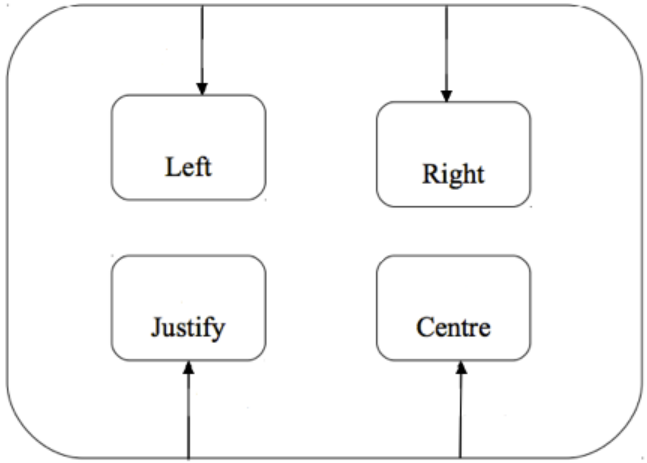
```
QPoint cursorPos;
addMouseTrans (s2, s2,
                    canvas, QEvent::MouseMove, Qt::NoButton, cursorPos,
                    this, SLOT(adjustLine( )) );
```

- cette variante met à jour **cursorPos** (la position du curseur) à chaque transition

Evénements **utilisateur**

Problème

- un objet de MaClasse a **4 modes** exclusifs
  - ex : aligner à gauche, a droite, etc.
- cette méthode change son **mode**:
  - void MaClasse::setMode(int mode)



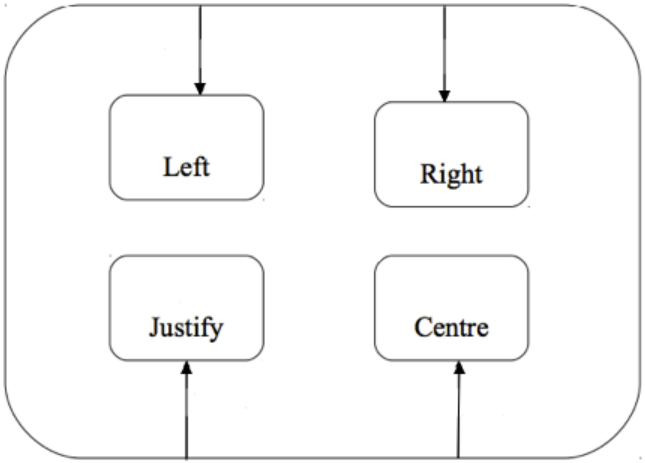
- comment **déclencher** une **transition** quand setMode() est appelée ?

Evénements utilisateur

Problème

```
void MaClasse::setMode(int mode)
```

- comment **déclencher** une **transition** ?

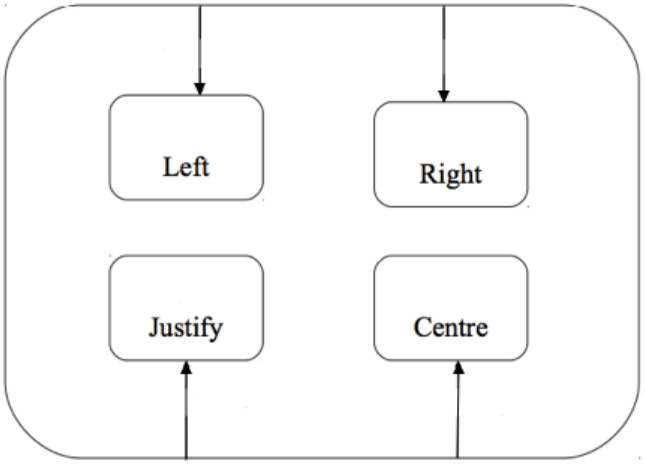


Solution 1

- setMode() émet des **signaux**
  - attention : il faut un signal **pour chaque état**
  - sinon la transition sera franchie quel que soit le mode

```
// la transition sera franchie quelle que soit la valeur de "mode"
s1->addTransition (myobject, SIGNAL(modeChanged( )), s2);
```

Evénements utilisateur



Solution 2

- setMode() envoie des **user events**
  - identifiés par des **entiers** choisis par le programmeur

```
void MaClasse::setMode( int mode ) {
    this->mode = mode;
    mac.postEvent( new QEvent(mode) );      // mac est la machine a états
}

// LEFT_MODE etc. sont des entiers entre QEvent::User et QEvent::MaxUser
addUserEventTrans (parent_state, left_state, LEFT_MODE);
addUserEventTrans (parent_state, right_state, RIGHT_MODE);
addUserEventTrans (parent_state, justify_state, JUSTIFY_MODE);
addUserEventTrans (parent_state, centre_state, CENTRE_MODE);
```

Déclencher une transition après un **délai**

Avec des événements utilisateur

- même principe, avec la méthode **postDelayedEvent**() de la **machine à états**

```
void MaClasse::setMode( int mode ) {
    this->mode = mode;
    mac.postDelayedEvent( new QEvent(mode),  delay);
}
```

Avec des Timers

- voir slide dédié

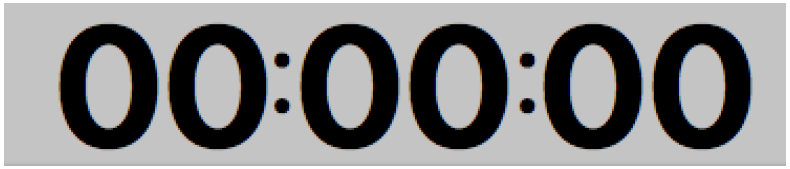
```
QTimer * timer = new QTimer( );
addTrans (state1, state2, timer, SIGNAL(timeout( )),
// puis :
timer->start(delay);
....
timer->stop();
```

Savoir si un état est **actif**

Solution

- deux méthodes (la 2e **n'existe pas** dans les vielles versions de Qt)

- **attention :** lorsque l'on **franchit** une **transition** les **états** qui y sont liés sont **inactifs**



réglage de l'heure d'une horloge

```
QSet<QAbstractState*>
config  = machine->configuration();

if (config.contains(hours_state)) {
    ...
 }
 else if (config.contains(minutes_state)) {
    ...
 }
```

```
void MainWindow::setClock(int value) {
  if (hours_state->active()) {
      setHours(value);
  }
  else if (minutes_state->active()) {
      setMinutes(value);
  }
  else if (seconds_state->active()) {
      setSecond(value);
  }
}
```

Gardes

Rappel

- **conditions** de déclenchement
- notation : **prédicat : événement / action**

Transitions.h

addTrans (guard, state1, state2, sender, signal);

addTrans (guard, state2, sender, signal, receiver, slot);

guard est un **bool** (ou un objet comparable a bool)

Gardes (filtres de transitions)

Principe

- 1) quand une transition **peut être activée** sa méthode **eventTest( )** est appelée
- 2) la transition n'est **franchie que si eventTest( )** renvoie **true**

```
class MouseTransition : public QMouseEventTransition {
    QPoint& pos;
    bool eventTest(QEvent * e) {
        if ( ! QMouseEventTransition::eventTest(e))  return false;                  // ne pas oublier cette ligne !
        QEvent* realEvent = static_cast<QStateMachine::WrappedEvent*>(e)->event( );  // vrai événement
        switch (realEvent->type( )) {
        case QEvent::MouseMove:
        case QEvent::MouseButtonPress:
        case QEvent::MouseButtonRelease:
            _pos = static_cast<QMouseEvent*>(realEvent)->pos( );        // sauver la position de la souris
        }
        return true;                  // true signifie que l'événement déclenche la transition (sinon il est ignoré)
    }
};
```

TP

Implémenter la logique de l'interface d'un micro-ondes

Clock

Mode

Stop

Power

Defrost

Auto Menu

Dial

Start

Touch

Initialiser `setAttribute(Qt::WA_AcceptTouchEvents);`

Redéfinir la méthode event() de QWidget

```
bool MyWidget::event (QEvent * e) {
    switch (e->type( )) {
    case QEvent::TouchBegin:
    case QEvent::TouchUpdate:
    case QEvent::TouchEnd:
        return myFunc( static_cast<QTouchEvent *>(e) );     // ma méthode pour le touch
     default:
        return QWidget::event(e);                          // sinon appeler la méthode standard
    }
}
```

myFunc() doit renvoyer **true** si l'événement est accepté
elle à accès aux points, leur état, leur pression… via QTouchEvent

Gestes

Pour détecter les gestes prédéfinis faire :

```
grabGesture(Qt::PanGesture);
grabGesture(Qt::PinchGesture);
grabGesture(Qt::SwipeGesture);
```

Puis redéfinir la méthode event( ) de QWidget

```
bool MyWidget::event (QEvent *e) {
    switch (e->type( )) {
    case QEvent::Gesture:
        return myFunc(static_cast<QGestureEvent *>(e));    // ma méthode pour les gestes
     default:
         return QWidget::event(e);                         // sinon appeler la méthode standard
    }
}
```

e->gesture() permet de filtrer les **gestes** en fonction de leur **type**.

MVC

Objectifs

- mieux **structurer** les applications
  - en particulier, séparer le **noyau fonctionnel** de l'**interface graphique** (GUI)
  - ex : tableur : **données et logique** vs. **représentation graphique**

- faciliter la **synchronisation** des représentations **multi-vues**
  - ex : représentation **textuelle** et **graphique** des mêmes données

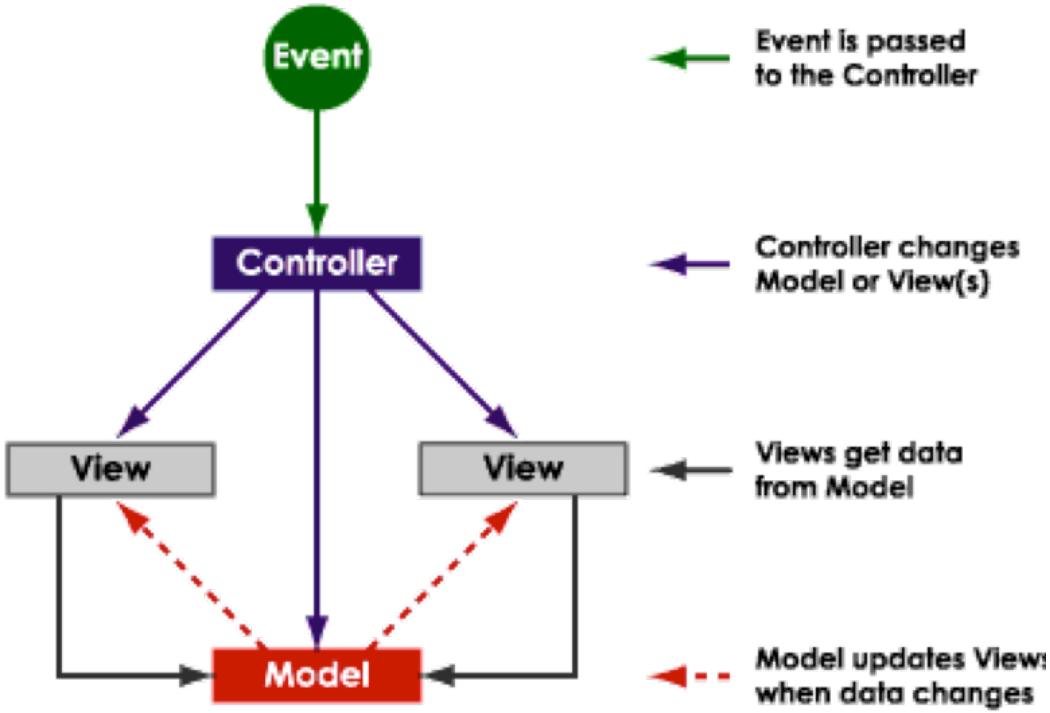
- permettre le **développement simultané** par **plusieurs** personnes ou équipes

- faciliter la **réutilisation du code**, en particulier :
  - si on change de **noyau fonctionnel**
  - si on change d'**interface** (par ex. de **toolkit graphique** et/ou de **plateforme**)

MVC

Trois composantes

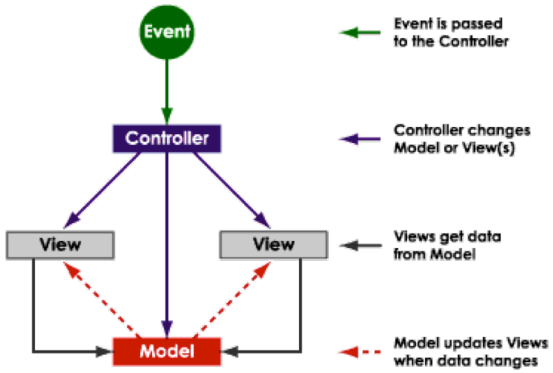
- **Model** : **données** et **logique** du programme
- **View** : représentation **visuelle**
- **Controller** : gestion des **entrées** et du **comportement** du système

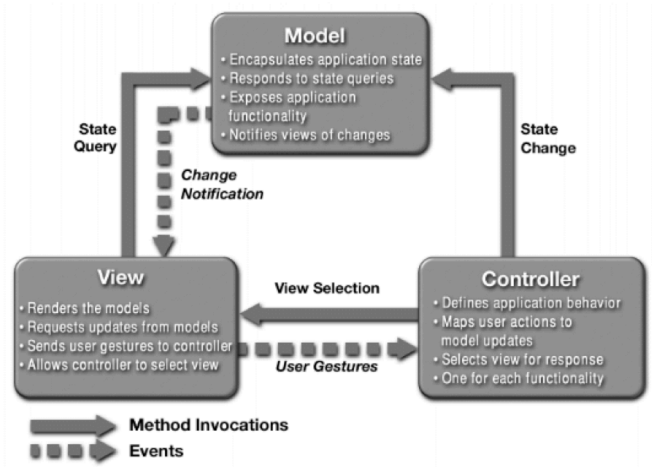


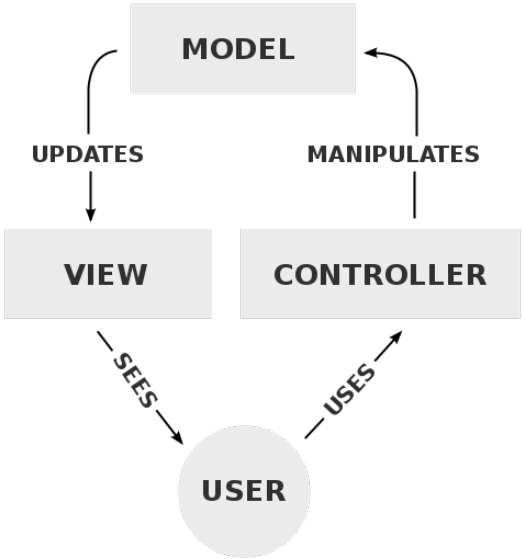
MVC

Variantes

- à l'origine Xerox PARC pour **Smalltalk** (1978)
- populaire pour **GUIs** et développement **Web**
- diverses **variantes** dont :
  - hierarchical model–view–controller (HMVC)
  - model–view–adapter (MVA)
  - model–view–presenter (MVP)
  - model–view–view model (MVVM)



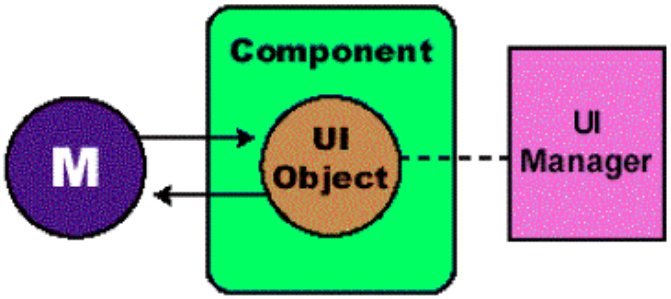




MVC et Java Swing

Architecture interne M+VC

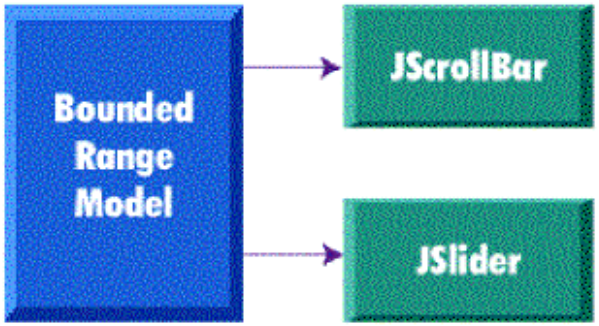
- **Controller** et **View** regroupés dans un **UIComponent**
  - ce pour chaque **JComponent**
- certains **JComponent** comportent un **Model**
- ce **Model** peut être "extrait" et **partagé** par :
  - plusieurs **JComponents**
  - et le **noyau fonctionnel**



Exemple

- **JScrollBar** et **JSlider**

  contiennent un **BoundedRangeModel**



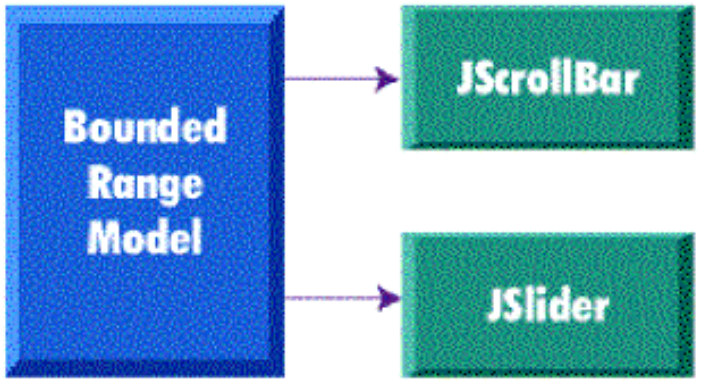
MVC et Java Swing : exemple

Dans l'API de JSlider

```
public BoundedRangeModel getModel();
public void setModel(BoundedRangeModel);
```

Changer le modèle du JSlider

```
JSlider slider = new JSlider();
BoundedRangeModel model =
      new DefaultBoundedRangeModel( ) {
            public void setValue(int n) {
                  System.out.println("SetValue: "+ n);
                  super.setValue(n);
            }
      });
 slider.setModel(model);
 scrollbar.setModel(model);
```



On peut aussi ignorer l'existence des modèles

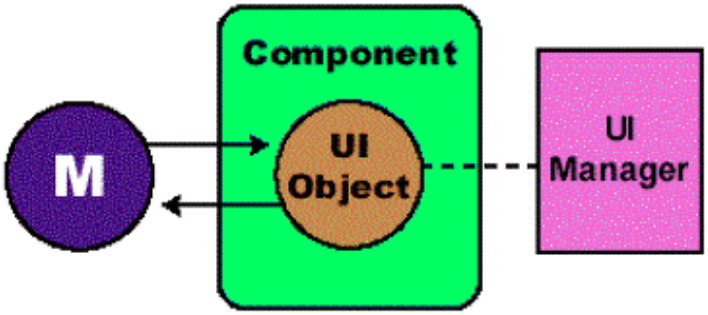
```
JSlider slider = new JSlider();
int value = slider.getValue();

// cohérent car dans l'API de JSlider on a:
public int getValue()  {return getModel().getValue(); }
```

MVC et Java Swing : exemple

"Pluggable Look and Feel"

le **UIManager** peut changer dynamiquement

les **ComponentUI**s



Java Metal:

```
public static void main(String[ ] args) {
   try {
        UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
   } catch (Exception e) { }
}
```

Windows:

```
UIManager.setLookAndFeel(
     "com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
     );
```

MVC et Java Swing

A plus haut niveau, autre "vision" possible

- **Listeners** = **Controllers** (de haut niveau)
- **JComponents** = **Views** (incluent en fait des **Controllers** internes de bas niveau)
- **Models**

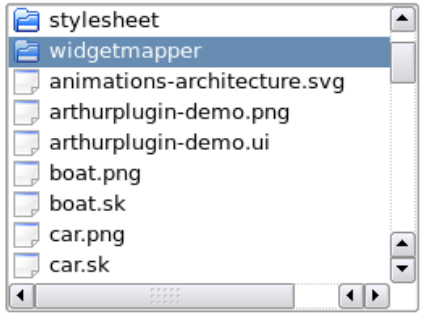
Proche de PAC

- **Presentation = View**
- **Abstraction = Model**
- **Control = Controllers/Listeners**
- noter que **PAC** est plus général (et hiérarchique)

MVC et Qt

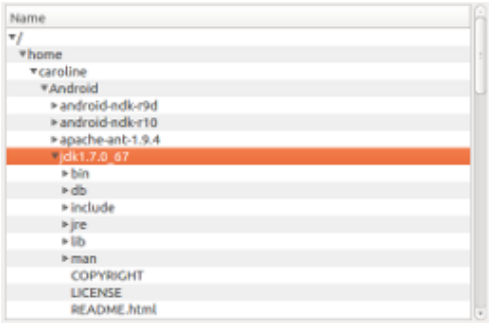
Deux formes pour certains widgets

- pour **Table**, **List** et **Tree** widgets
- soit widgets **standards**
- soit classes **Model/View**
  - évitent d'avoir **2 copies** des données
  - et à gérer leur **synchronisation**

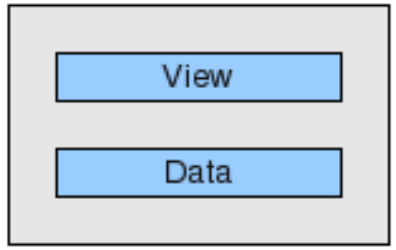


QListWidget
QListView

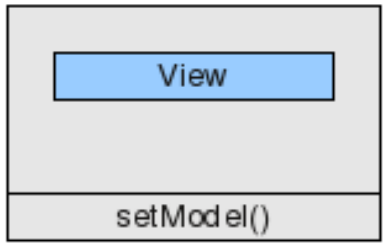
QTableWidget
QTableView

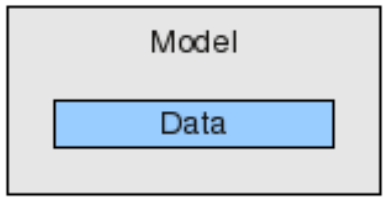


QTreeWidget
QTreeView



standard



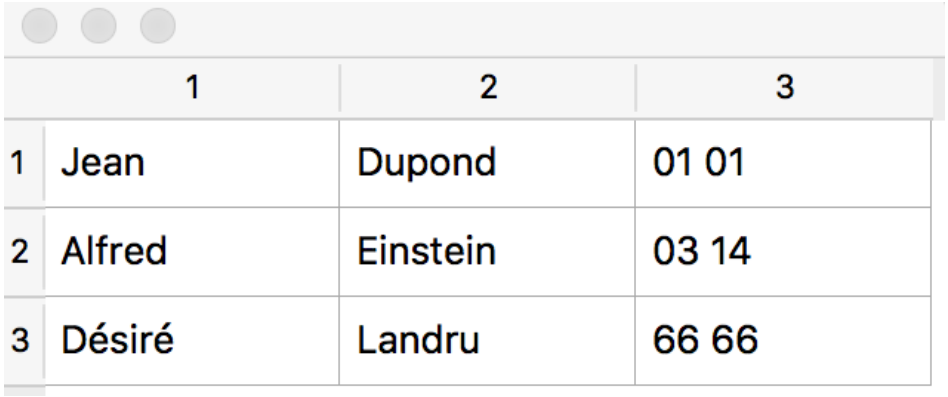


model / view

MVC et Qt : example

Exemple

- afficher / éditer une liste de **personnes** dans une **table**



	1	2	3
1	Jean	Dupond	01 01
2	Alfred	Einstein	03 14
3	Désiré	Landru	66 66

- chaque **personne** est un objet **Person** avec :
  - un prénom
  - un nom
  - un numéro de téléphone

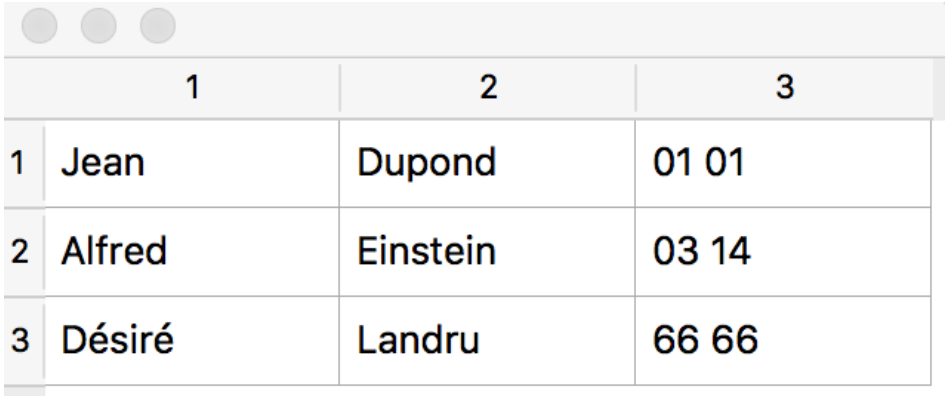
- la liste est stockée dans un **std::vector**

```
struct Person {
    std::string firstname, lastname, phonenumber;
};

std::vector<Person> persons;
```

MVC et Qt : example

fichier main.cpp

```
#include <QApplication>
#include <QtWidgets/QTableView>
#include "mymodel.h"

int main (int argc, char *argv[])
{
   QApplication app (argc, argv);

   MyModel model;
   QTableView view;
   view.setModel (&model);

   view.show ();
   return app.exec ();
 }
```



	1	2	3
1	Jean	Dupond	01 01
2	Alfred	Einstein	03 14
3	Désiré	Landru	66 66

note : en dehors de main(), créer ces objets avec new

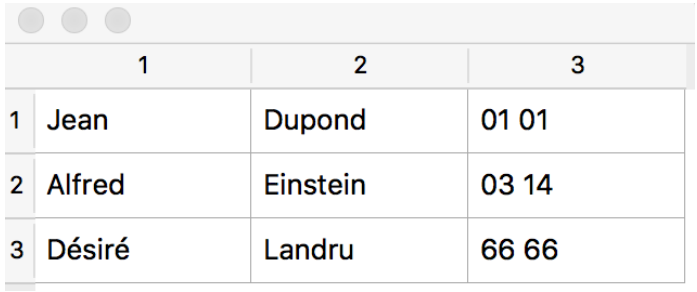
example

	1	2	3
1	Jean	Dupond	01 01
2	Alfred	Einstein	03 14
3	Désiré	Landru	66 66

fichier mymodel.hpp

```
#include <QAbstractTableModel>
#include <string>
#include <vector>

class MyModel : public QAbstractTableModel {
   Q_OBJECT
public:
   MyModel (QObject *parent);
                                   ces méthodes doivent être redéfinies
   int rowCount (const QModelIndex& = QModelIndex()) const override { return persons.size(); }
   int columnCount (const QModelIndex& = QModelIndex()) const override {return 3;}

   QVariant data (const QModelIndex& index, int role = Qt::DisplayRole) const override;
   bool setData (const QModelIndex & index, const QVariant & value, int role);
   Qt::ItemFlags flags (const QModelIndex & index) const override;

private:
   struct Person {    une personne
      std::string firstname, lastname, phonenumber;
   };
   std::vector<Person> persons;    la liste de personnes
};
```

example



	1	2	3
1	Jean	Dupond	01 01
2	Alfred	Einstein	03 14
3	Désiré	Landru	66 66

fichier mymodel.cpp

```
#include "mymodel.h"

MyModel::MyModel (QObject *parent) : QAbstractTableModel(parent) {
    persons.push_back( Person{"Jean", "Dupond", "01 01"} );
    persons.push_back( Person{"Alfred", "Einstein", "03 14"} );
    persons.push_back( Person{"Désiré", "Landru", "66 66"} );
}

QVariant MyModel::data (const QModelIndex &index, int role) const {
    if (role == Qt::DisplayRole) {
      switch (index.column()) {
      case 0:
         return QString( persons[index.row()].firstname.c_str() );
         break;
      case 1:
         return QString( persons[index.row()].lastname.c_str() );
         break;
      case 2:
         return QString( persons[index.row()].phonenumber.c_str() );
         break;
      }
    }
    return QVariant();
}
```

initialise la liste de personnes

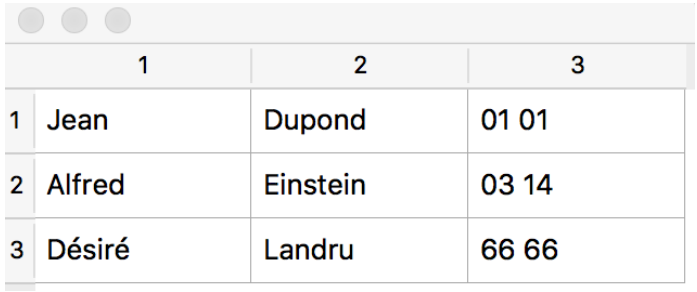
data() lit le modèle

mode "renvoyer les données à afficher"

(à suivre)

example

suite du fichier mymodel.cpp



	1	2	3
1	Jean	Dupond	01 01
2	Alfred	Einstein	03 14
3	Désiré	Landru	66 66

setData() modifie le contenu du modèle

```
bool MyModel::setData (const QModelIndex & index, const QVariant & value, int role) {
    if (role == Qt::EditRole) {    mode "éditer les données"
        switch (index.column()) {
        case 0:
            persons[index.row()].firstname = value.toString().toStdString();
            break;
        case 1:
            persons[index.row()].lastname = value.toString().toStdString();
            break;
        case 2:
            persons[index.row()].phonenumber = value.toString().toStdString();
            break;
        }
        emit dataChanged (index,index);     ne pas oublier ! indique que cette cellule a été modifiée
    }
}
```

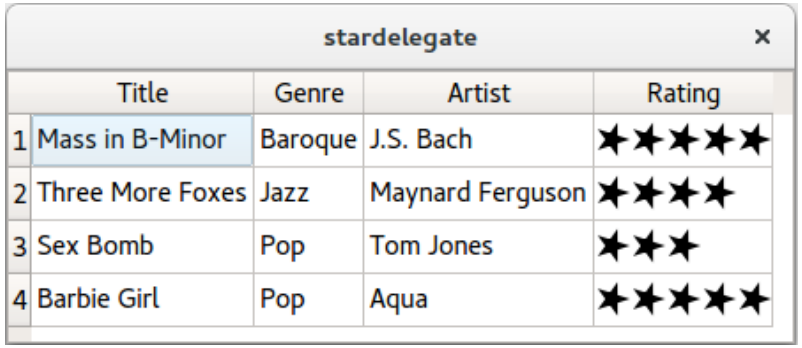
flags() spécifie que l'on peut éditer la table

```
Qt::ItemFlags MyModel::flags (const QModelIndex &index) const {
    return Qt::ItemIsEditable | QAbstractTableModel::flags(index);
}
```

MVC et Qt

Autres possibilités

- changer les **couleurs,** les **polices**, etc.
- afficher / éditer autre chose que du **texte**
- synchroniser **plusieurs vues**
  - et même la **sélection**
  - NB : ne pas oublier : **emit dataChanged** !
- et bien d'autres fonctionnalités...



Remarque

- les **autres widgets** peuvent aussi utiliser les modèles via **QDataWidgetMapper**

```
MyModel * model = new MyModel;
QTableView * view1 = new QTableView;
QTableView * view2 = new QTableView;

view1->setModel (model);
view2->setModel (model);
```

synchroniser deux vues