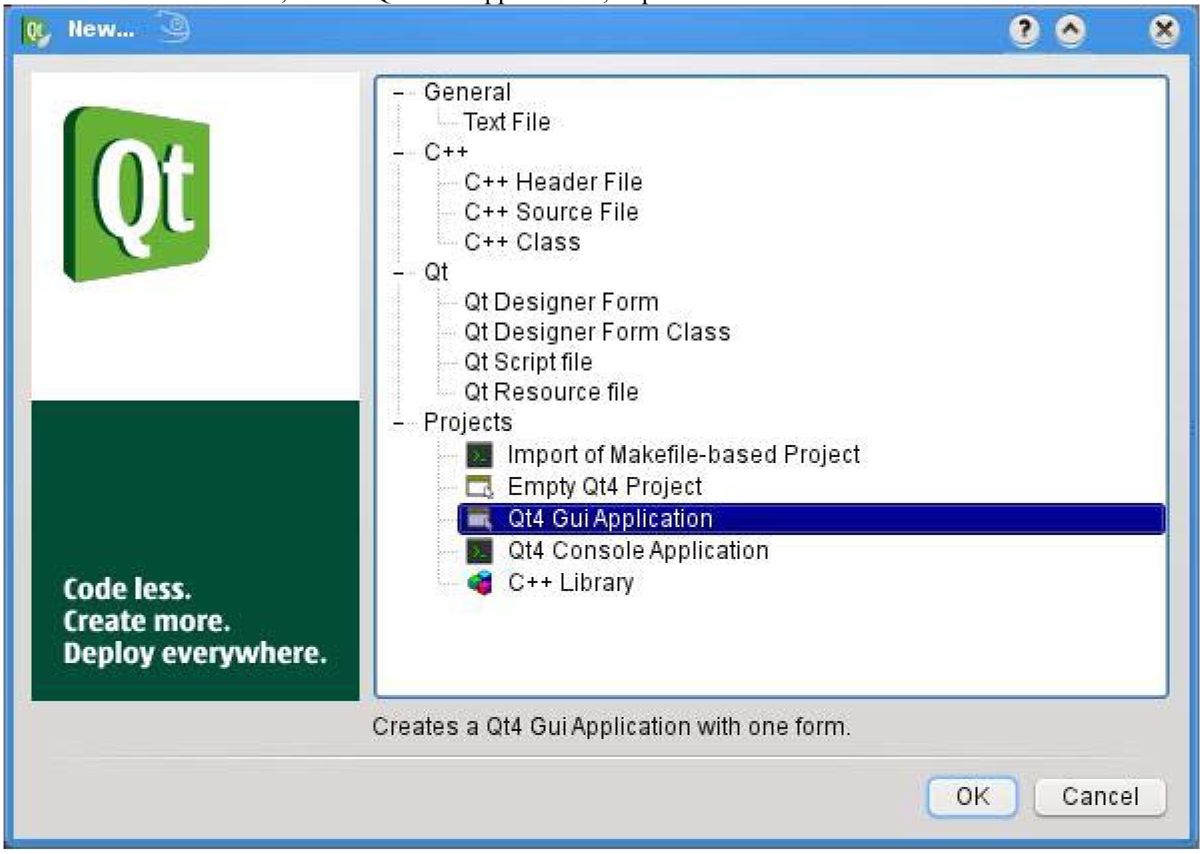
Nous allons nous intéresser aux aspects méconnus de Qt pour faire de la simulation

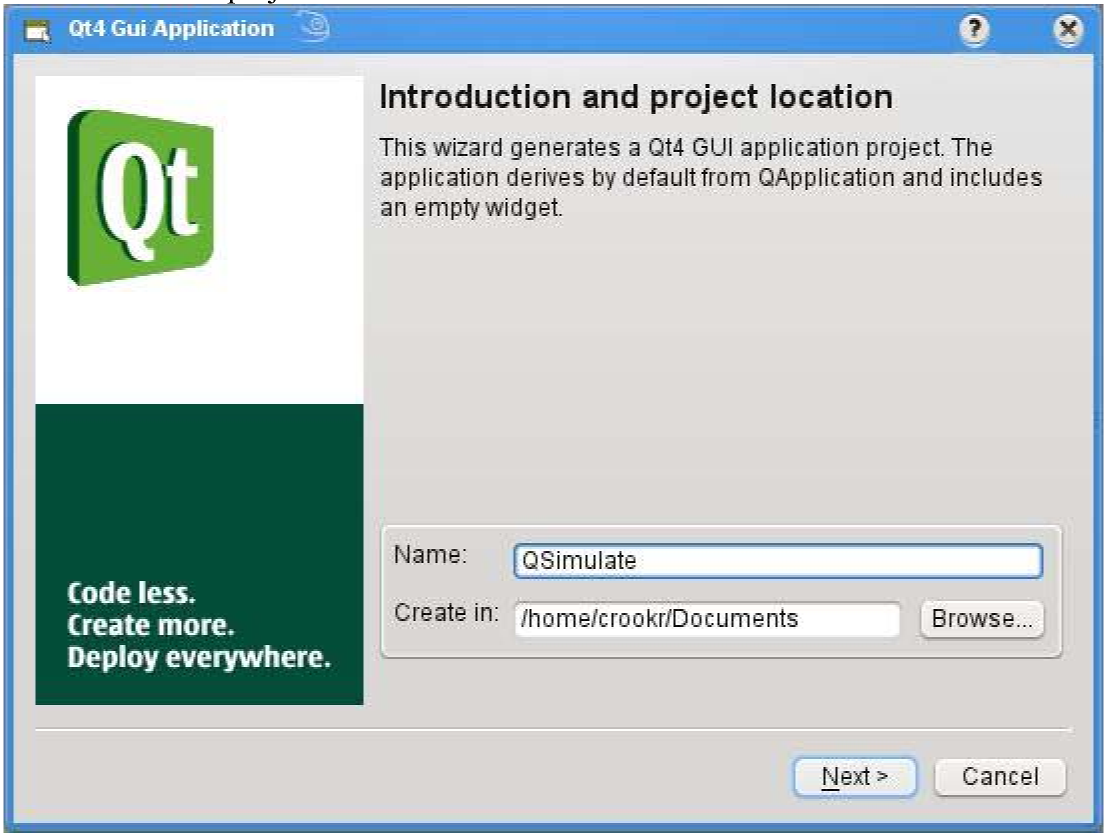
1- Création du projet

Menu File select "New...", select "Qt4 Gui Application", et presser "OK".



Creates a Qt4 Gui Application with one form.

Entrer le nom du projet



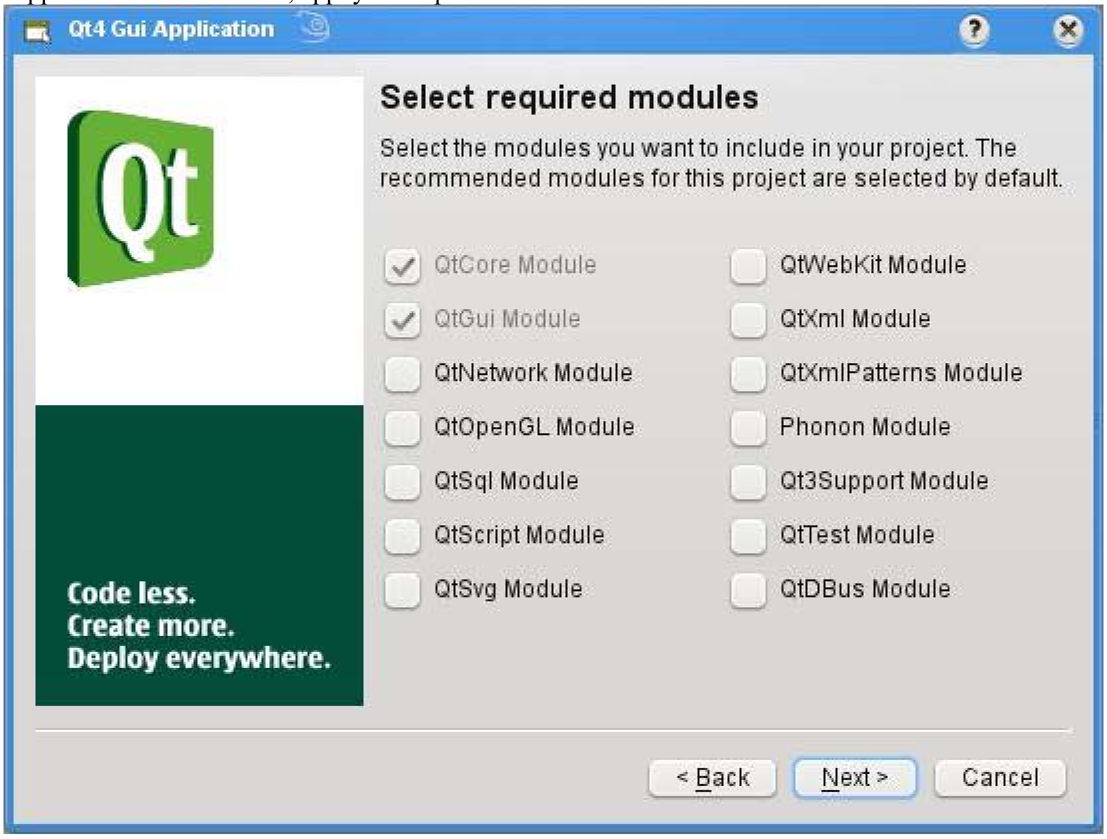
Introduction and project location

This wizard generates a Qt4 GUI application project. The application derives by default from QApplication and includes an empty widget.

Name: QSimulate

Create in: /home/crookr/Documents

QtCreator a automatiquement sélectionné les modules et les QtCore QtGui. Nous n'avons pas besoin de modules supplémentaires à ce stade, appuyez simplement sur "Suivant" à nouveau.



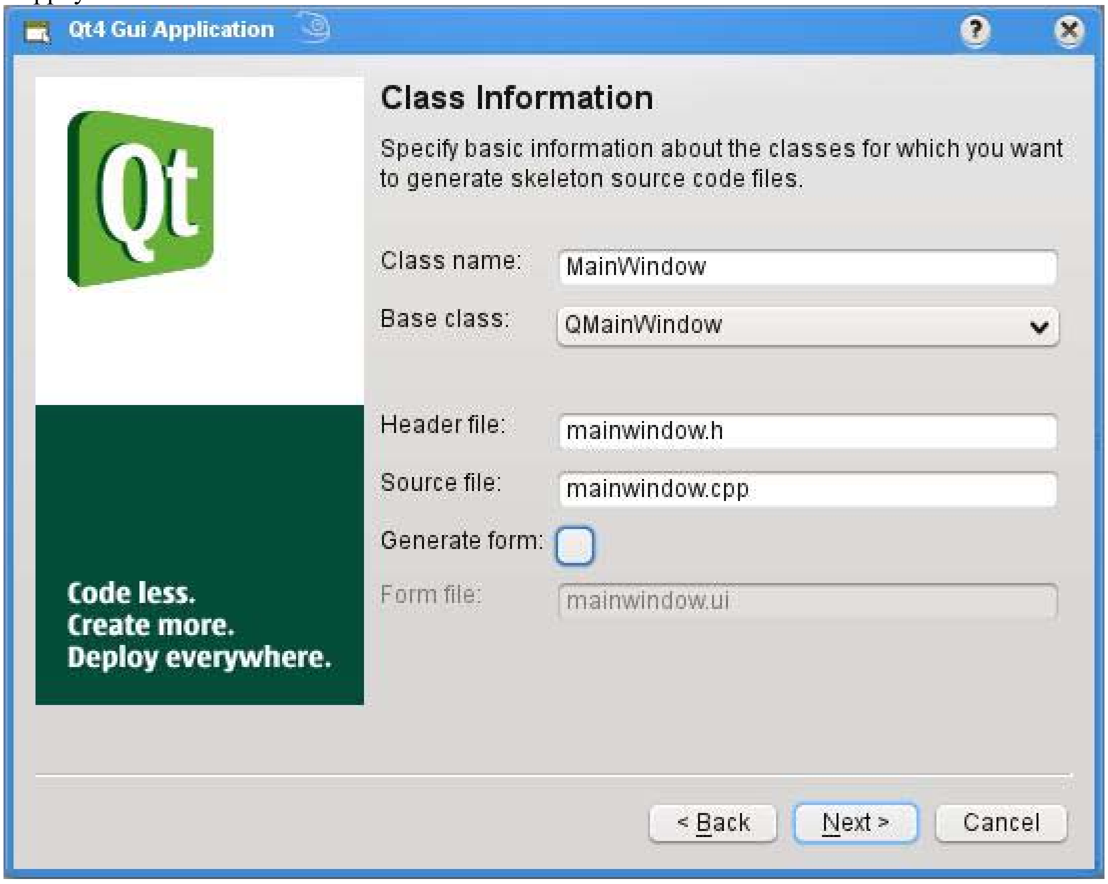
Qt4 Gui Application

Select required modules

Select the modules you want to include in your project. The recommended modules for this project are selected by default.

- ✓ QtCore Module
- ✓ QtGui Module
- QtNetwork Module
- QtOpenGL Module
- QtSql Module
- QtScript Module
- QtSvg Module
- QtWebKit Module
- QtXml Module
- QtXmlPatterns Module
- Phonon Module
- Qt3Support Module
- QtTest Module
- QtDBus Module

Ici QtCreator nous donne la possibilité de modifier les détails de la classe, il va créer pour nous le code du modèle. Toutes les valeurs par défaut sont bonnes, s'assurer que la case à cocher "Generate form" n'est pas cochée avant d'appuyer sur "Suivant".



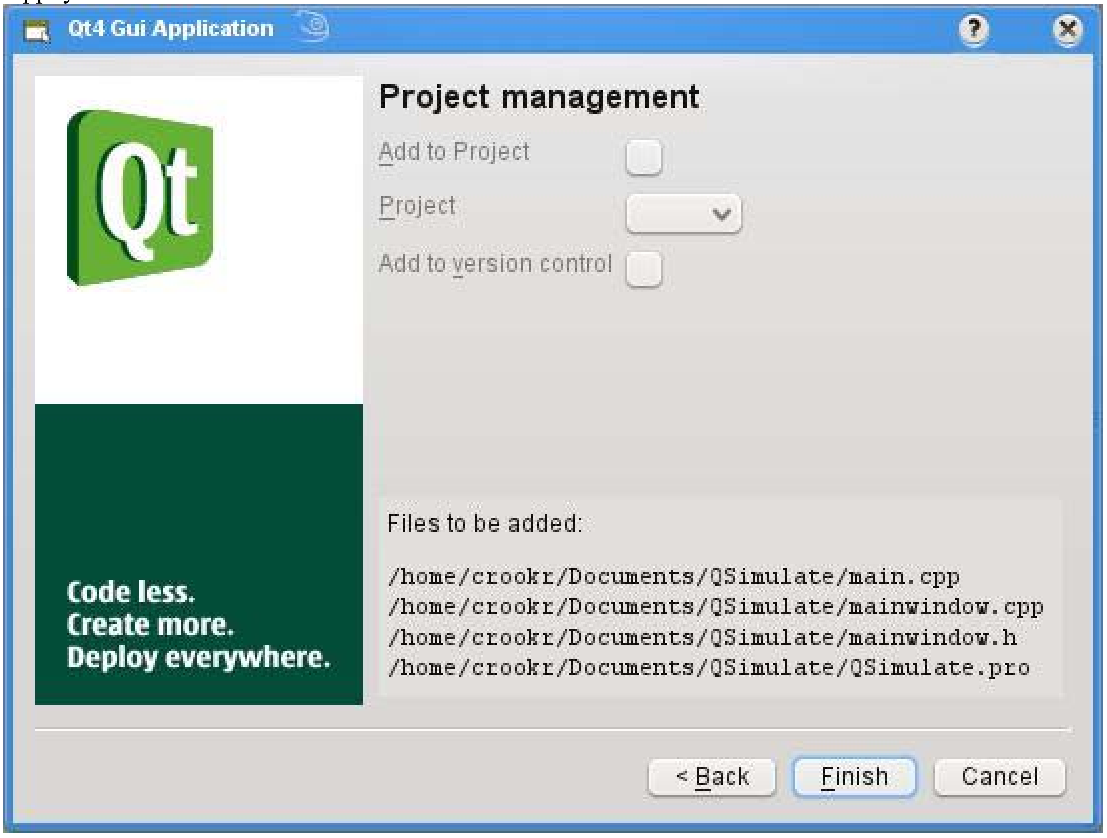
Qt4 Gui Application

Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name:	MainWindow
Base class:	QMainWindow
Header file:	mainwindow.h
Source file:	mainwindow.cpp
Generate form:	☐
Form file:	mainwindow.ui

Appuyer sur Finish

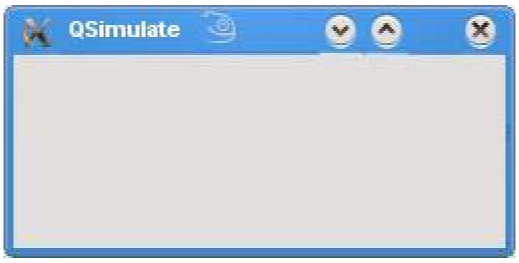


Files to be added:

```
/home/crookr/Documents/QSimulate/main.cpp
/home/crookr/Documents/QSimulate/mainwindow.cpp
/home/crookr/Documents/QSimulate/mainwindow.h
/home/crookr/Documents/QSimulate/QSimulate.pro
```

QtCreator a maintenant tout mis en place pour nous permettre de développer notre application Qt et a créé automatiquement un code modèle pour nous aider à démarrer. En plus d'un projet Qt qmake (. Pro) fichier dont QtCreator s'occupera pour nous, QtCreator a créé 3 fichiers C++ appelé "main.cpp", "mainwindow.cpp", et "mainwindow.h".

QtCreator simplifie la compilation et l'exécution du code en appuyant sur la commande "Exécuter" triangle vert, ou par le raccourci clavier Ctrl + R, ou des options de menu. Si QtCreator demande de sauver tout changement de l'un des fichiers, s'il vous plaît dites oui.



2- Ajout d'une menu barre et d'une barre de status

Maintenant que nous avons commencé notre projet `QtCreator`, nous pouvons commencer à ajouter de nouvelles fonctionnalités à notre demande. Le premier ajout de nouvelles fonctionnalités que nous mettrons en œuvre est d'ajouter une barre de menu et une barre d'état. La classe `QMainWindow` offre de nombreuses installations utiles qui rendent cette tâche très facile.

Le code du modèle fourni par `QtCreator` est un bon début, mais nous allons le remplacer par notre code montré ci-dessous pour ajouter quelques commentaires et d'y ajouter nos fonctionnalités.

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

int main(int argc, char *argv[])
```

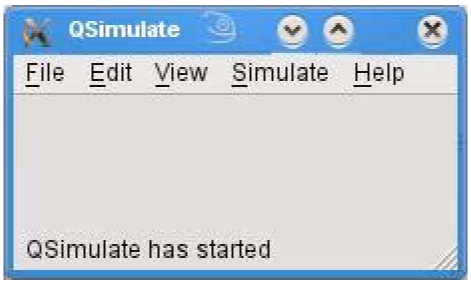
```
{
  // create main event loop handler and parse command line arguments
  QApplication app(argc, argv);

  // create application main window & enter main event loop
  MainWindow window;
  window.show();
  return app.exec();
}
```

Le fichier `main.cpp` contient le "main" qui est le point d'entrée lorsque l'application est lancée, et utilise le `QApplication` classe Qt et une autre classe appelée `MainWindow` pour créer une fenêtre très simple interface graphique qui peut être redimensionnée, déplacée, et fermée. Lorsque la fenêtre est fermée, l'application se termine.

```
#include "mainwindow.h"

#include <QMenuBar>
#include <QStatusBar>

/*************************************************************/
/***** Main application window for QSimulate *****************/
/*************************************************************/

MainWindow::MainWindow() : QMainWindow()
{
  // add drop down menus (currently empty)
  menuBar()->addMenu("&File");
  menuBar()->addMenu("&Edit");
  menuBar()->addMenu("&View");
  menuBar()->addMenu("&Simulate");
  menuBar()->addMenu("&Help");

  // add status bar message
  statusBar()->showMessage("QSimulate has started");
}
```

Dans le constructeur, nous utilisons la `menuBar` () et `statusBar` () fonctions héritées de `QMainWindow` pour ajouter nos menus à la barre de menu et pour afficher un message sur la barre d'état. Pour utiliser avec succès nous avons besoin d'inclure des déclarations pour le `QMenuBar` Qt classes et `QStatusBar`.

Cinq menus déroulants sont créés (`File`, `Edit`, `View`, `Simulate` et `Help`), actuellement sans inscriptions. Nous allons ajouter des entrées plus tard. Notez que le placement d'une esperluette dans le titre du menu indique à Qt pour souligner le caractère suivant et de créer un raccourci clavier automatique. La barre d'état, comme la barre de menu principale, est créé automatiquement par `QMainWindow` quand nous commençons à l'utiliser.



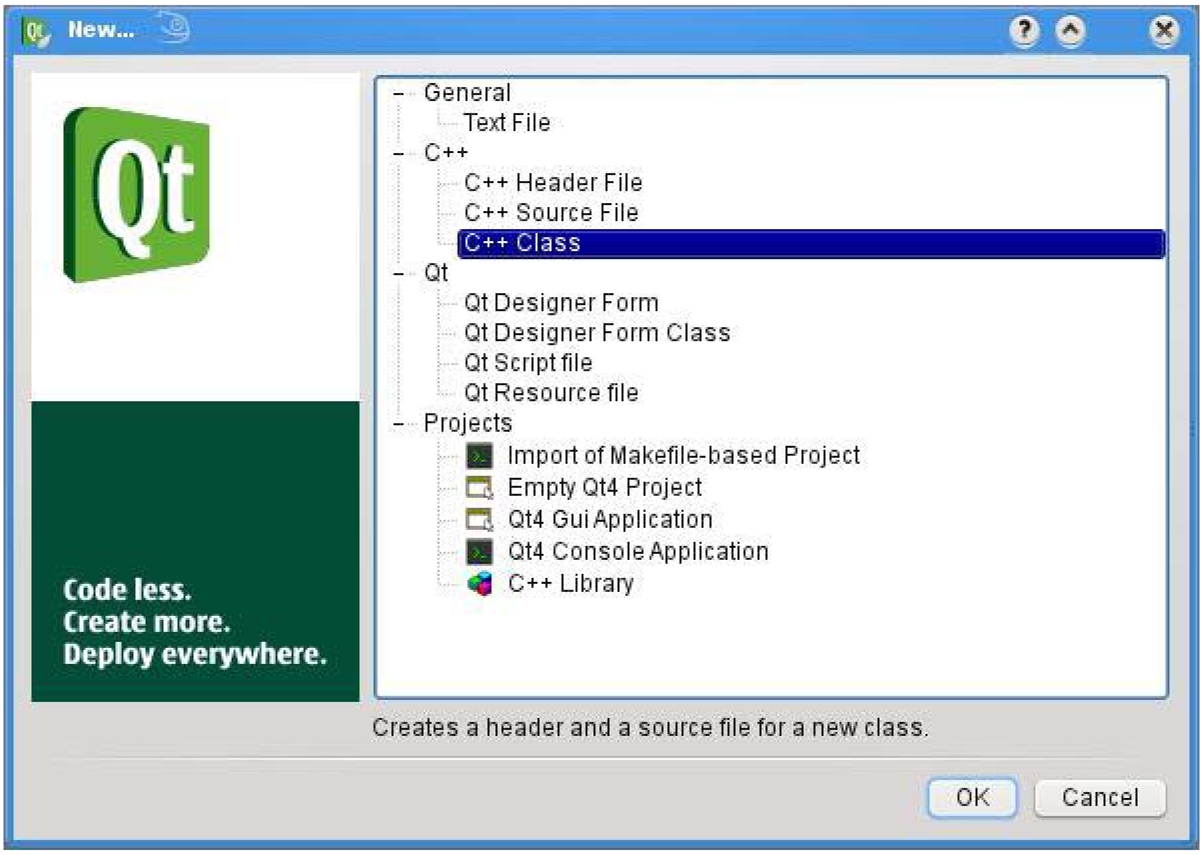
QSimulate has started

3- Ajout d’une zone graphique

Pour permettre à l'utilisateur de gérer et d'interagir avec nos données métier de manière graphique, nous allons utiliser un widget "Graphics View" centrée sur notre fenêtre principale. Le Qt Graphics View Framework fournit de nombreux équipements utiles pour afficher et interagir avec ceux faits en 2D éléments graphiques.

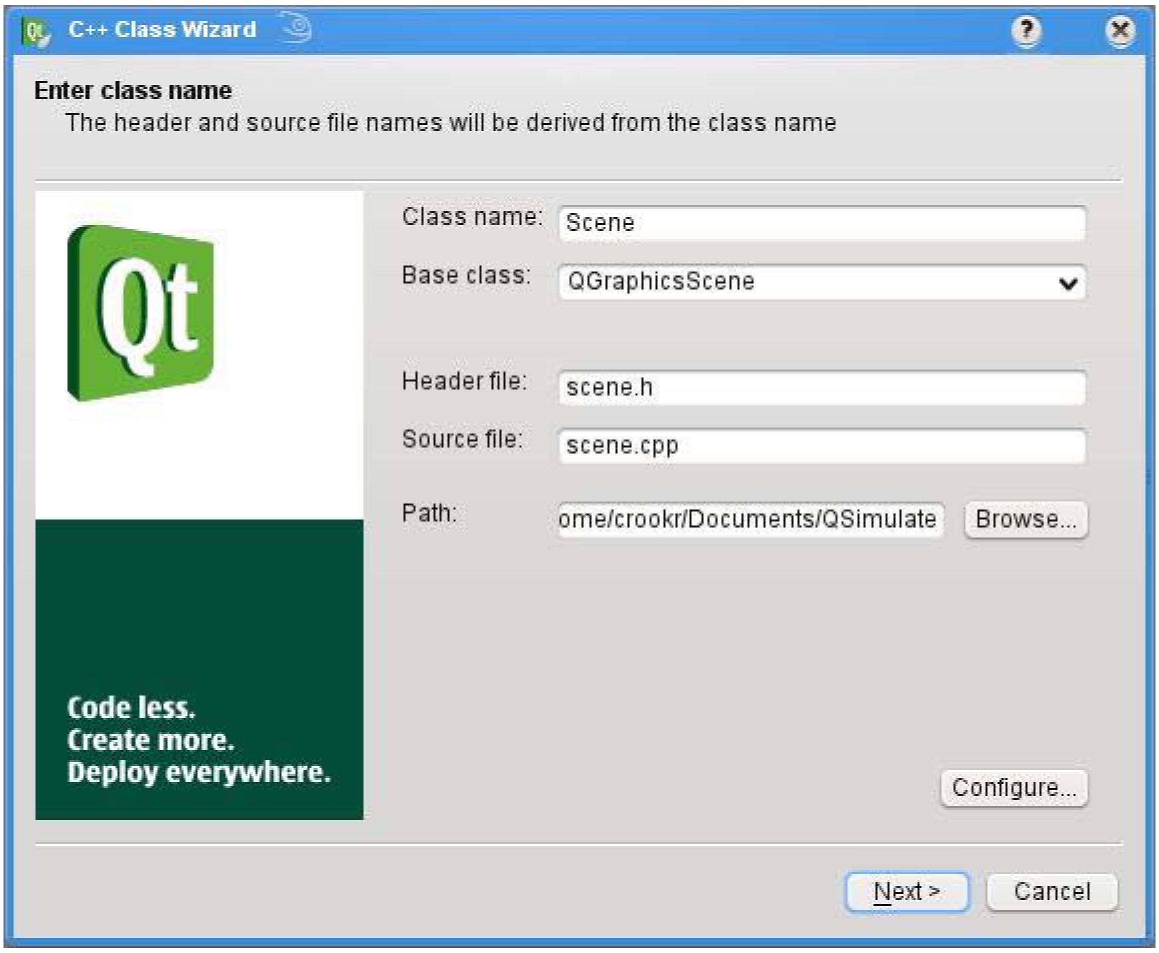
Pour mettre en œuvre les zones graphiques, nous allons créer une scène qui va contenir nos données métier et un widget pour la vue à afficher dans la région centrale de notre fenêtre principale. Pour nous donner la flexibilité pour l'avenir et pour encapsuler les données au sein de la scène, nous allons créer une nouvelle classe à mettre en œuvre la scène.

La nouvelle classe sera appelée «`Scene`» et est ajoutée à notre projet en utilisant le menu Fichier `QtCreator` "Nouveau ..." Assistant. Sélectionnez "Classe C + +" et appuyez sur "OK".



Creates a header and a source file for a new class.

Entrez le nom de la classe «`Scene`» et la classe de base "`QGraphicsScene`" avant d'appuyer sur "Suivant". Les noms de fichier en-tête et la source doivent être générés automatiquement, et le chemin par défaut devrait être bon.


C++ Class Wizard

Enter class name

The header and source file names will be derived from the class name

Class name: Scene

Base class: QGraphicsScene

Header file: scene.h

Source file: scene.cpp

Path: ome/crookr/Documents/QSimulate

Enfin, appuyez sur "Terminer" pour ajouter ces fichiers à notre projet.

Modifier `Scene.h`

```
#ifndef SCENE_H
#define SCENE_H

#include <QGraphicsScene>

/*****************************************************/
/***** Scene representing the simulated landscape *****/
/*****************************************************/
class Scene : public QGraphicsScene
{
public:
  Scene();              // constructor
};
#endif  // SCENE_H
```

`Scene.cpp`

```
#include "scene.h"

/******************************************************/
/***** Scene representing the simulated landscape ******/
/******************************************************/
Scene::Scene() : QGraphicsScene()
{
}
```

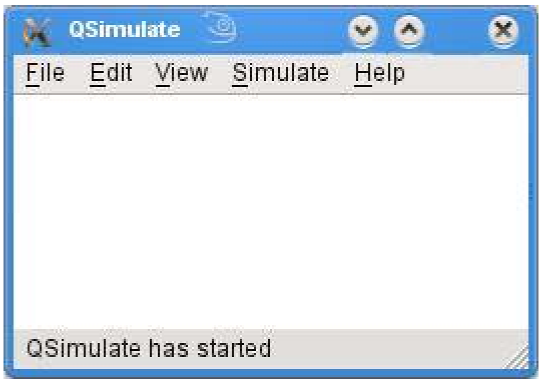
La scène et la vue seront créés dans le constructeur de la fenêtre principale.

`MainWindow.h`

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
class Scene;
#include <QMainWindow>

/*******************************************************/
/********* Main application window for QSimulate *******/
/*******************************************************/
class MainWindow : public QMainWindow
{
public:
  MainWindow();     // constructor
private:
  Scene* m_scene;   // scene representing the simulated landscape
};
#endif  // MAINWINDOW_H
```

Parce que nous n'avons pas besoin de la définition complète de la classe Scene dans cette entête, au lieu juste ajouter une déclaration avant nous plaçons une déclaration incomplète. Cela rend la compilation des projets beaucoup plus rapide, car le compilateur passe habituellement la plupart de ses temps d'analyse des fichiers d'en-tête, pas le code source.
`MainWindow.cpp`

```
#include "mainwindow.h"
#include "scene.h"
#include <QMenuBar>
#include <QStatusBar>
#include <QGraphicsView>

/****************************************************/
/******* Main application window for QSimulate ******/
/****************************************************/
MainWindow::MainWindow() : QMainWindow()
{
  // add drop down menus (currently empty)
  menuBar()->addMenu("&File");
  menuBar()->addMenu("&Edit");
  menuBar()->addMenu("&View");
  menuBar()->addMenu("&Simulate");
  menuBar()->addMenu("&Help");

  // create scene and central widget view of scene
  m_scene               = new Scene();
  QGraphicsView*    view = new QGraphicsView( m_scene );
  view->setAlignment(Qt::AlignLeft | Qt::AlignTop );
  view->setFrameStyle(0);
  setCentralWidget(view);

  // add status bar message
  statusBar()->showMessage("QSimulate has started");
}
```



QSimulate has started

4- Amélioration de la barre de status

Nous allons utiliser le mécanisme de Qt signal et slot pour améliorer la barre d'état et dire où l'utilisateur a cliqué sur la zone d'affichage central. Avec le signal Qt et le mécanisme de slot, il est facile pour les objets d'application de communiquer sans interdépendances lors de la mise en œuvre. Le concept est que les objets peuvent envoyer des signaux contenant les informations d'événement qui peuvent être reçu par d'autres objets en utilisant des fonctions spéciales appelées slots. Le système signal / slot correspond bien à la façon dont les interfaces utilisateur graphiques sont conçues.

Le cadre Qt Graphics View utilisé pour créer le widget écran central fournit de nombreuses installations utiles, y compris ce qui nous permet de détecter lorsque l'utilisateur a cliqué. Nous améliorerons notre classe Scene afin de détecter lorsque l'utilisateur a cliqué qui sera alors envoyer un signal, et nous améliorerons notre classe MainWindow pour détecter ce signal avec un slot pour afficher les informations sur la barre d'état.

Pour détecter l'endroit où l'utilisateur a cliqué, nous mettrons en œuvre la méthode qui reçoit les événements `mousePressEvent` avec un `QGraphicsSceneMouseEvent`. Pour envoyer le signal nous devons ajouter la macro `Q_OBJECT` et ajouter une méthode de signal. Notre méthode de signal sera appelé "`message`" et contiendra une chaîne de texte. La macro `Q_OBJECT` à l'intérieur de la section privée de la déclaration de classe est utilisé pour activer `Qt méta-objet` fonctionnalités telles que les propriétés dynamiques, les signaux et slots.
Amélioration de la `Scene.h`

```
#ifndef SCENE_H
#define SCENE_H
class QGraphicsSceneMouseEvent;
#include <QGraphicsScene>

/************************************************************/
/******** Scene representing the simulated landscape *******/
/************************************************************/
class Scene : public QGraphicsScene
{
  Q_OBJECT
public:
  Scene();                  // constructor

signals:
  void  message(QString); // info text message signal

protected: // receive mouse press events
  void  mousePressEvent(QGraphicsSceneMouseEvent* );
};
#endif  // SCENE_H
```

Amélioration de la `Scene.cpp`

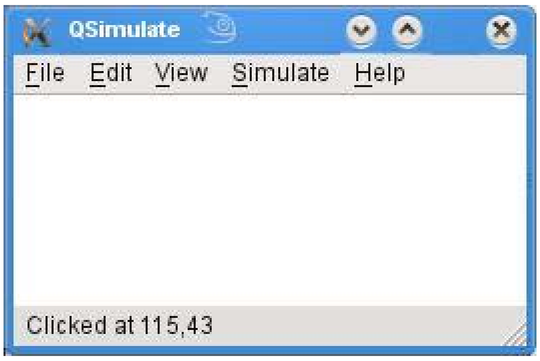
```
#include "scene.h"
#include <QGraphicsSceneMouseEvent>

/*******************************************************/
/******* Scene representing the simulated landscape ***/
/*******************************************************/
Scene::Scene() : QGraphicsScene()
{
}

/*************** mousePressEvent ********************/
void Scene::mousePressEvent(QGraphicsSceneMouseEvent* event)
{
  // only interested if left mouse button pressed
  if ( event->button() != Qt::LeftButton ) return;

  // emit informative message
  qreal  x = event->scenePos().x();
  qreal  y = event->scenePos().y();
  emit message( QString("Clicked at %1,%2").arg(x).arg(y) );
}
```

Amélioration de la `MainWindow.h`

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
class Scene;

#include <QMainWindow>

/*************************************************************/
/******** Main application window for QSimulate ************/
/*************************************************************/
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  MainWindow();  // constructor

public slots:
  void showMessage(QString); // show message on status bar

private:
  Scene* m_scene; // scene representing the simulated landscape
};
#endif  // MAINWINDOW_H
```

Amélioration de la `MainWindow.cpp`

```
#include "mainwindow.h"
#include "scene.h"
#include <QMenuBar>
#include <QStatusBar>
#include <QGraphicsView>

/*************************************************************/
/****** Main application window for QSimulate **************/
/*************************************************************/
MainWindow::MainWindow() : QMainWindow()
{
  // add drop down menus (currently empty)
  menuBar()->addMenu("&File");
  menuBar()->addMenu("&Edit");
  menuBar()->addMenu("&View");
  menuBar()->addMenu("&Simulate");
  menuBar()->addMenu("&Help");

  // create scene and central widget view of scene
  m_scene               = new Scene();
  QGraphicsView*   view = new QGraphicsView( m_scene );
  view->setAlignment( Qt::AlignLeft | Qt::AlignTop );
  view->setFrameStyle( 0 );
  setCentralWidget( view );

  // connect message signal from scene to showMessage slot
  connect(m_scene, SIGNAL(message(QString)), this,
SLOT(showMessage(QString)) );

  // add status bar message
  statusBar()->showMessage("QSimulate has started");
}

/***************** showMessage **************/
void  MainWindow::showMessage( QString msg )
{
  // display message on main window status bar
  statusBar()->showMessage( msg );
}
```



Clicked at 115,43

5- Dessin dans la zone d'affichage

Maintenant, nous allons améliorer notre application afin de permettre à l'utilisateur de placer des icônes graphiques représentant les stations de radio à l'affichage de la simulation en cliquant sur la zone d'affichage central.

Les icônes seront dessinées en ajoutant des éléments graphiques à la scène de graphiques que nous avons créé précédemment. Chaque élément sera une instance d'une nouvelle classe appelée station qui sera dérivé de la `QGraphicsItem` Qt classe.

Notre nouvelle classe appelée "`Station`" doit être ajoutée à notre projet de la même manière comme précédemment. Utilisez le menu Fichier QtCreator "Nouveau ..." Assistant. Sélectionnez "Classe C + +", entrez le nom de classe et appuyez sur "Suivant" puis sur "Terminer". Pour mettre en œuvre le code nécessaire, remplacez le code du modèle fourni par QtCreator avec notre propre code ci-dessous.

`Station.h`

```
#ifndef STATION_H
#define STATION_H
#include <QGraphicsItem>

/*************************************************************/
/****** Represents a radio station in the simulation **********/
/*************************************************************/
class Station : public QGraphicsItem
{
public:
  Station( qreal, qreal );    // constructor
  void paint(QPainter*, const QStyleOptionGraphicsItem*, QWidget*);
  // implement virtual paint function
  QRectF boundingRect() const
  { // implement virtual boundingRect
    return QRectF(-6.5, -13, 13, 18);
  }
};
#endif  // STATION_H
```

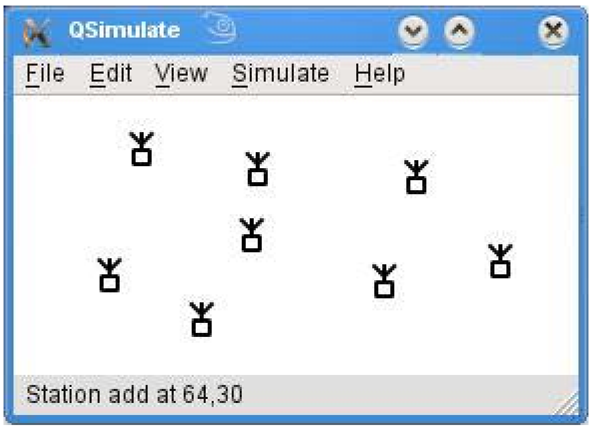
`Station.cpp`

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

/*************************************************************/
```

```
/** Represents a radio station in the simulation ************/
/************************************************************/
Station::Station( qreal x, qreal y ) : QGraphicsItem()
{ // set Station pixmap and position
  setPos( x, y );
}

/*********** paint ***********************/
void  Station::paint(QPainter *painter,
                     const QStyleOptionGraphicsItem *option,
                     QWidget *widget)
{
  // paint station symbol, must be smaller than bounding rectangle
  painter->setRenderHint( QPainter::Antialiasing );
  painter->setPen( QPen( Qt::black, 2 ) );
  painter->drawRect( -4,  -3,  8,   7 );
  painter->drawLine(  0,  -4,  0, -11 );
  painter->drawLine( -5, -11,  0,  -6 );
  painter->drawLine( +5, -11,  0,  -6 );
}
```

Amélioration de la `Scene.cpp`

```
#include "scene.h"
#include "station.h"
#include <QGraphicsSceneMouseEvent>

/**********************************************************/
/****** Scene representing the simulated landscape ********/
/**********************************************************/
Scene::Scene() : QGraphicsScene()
{
  // create invisible item to provide default top-left anchor to scene
  addLine( 0, 0, 0, 1, QPen(Qt::transparent, 1) );
}

/********************** mousePressEvent ****************/
void  Scene::mousePressEvent( QGraphicsSceneMouseEvent* event )
{
  // only interested if left mouse button pressed
  if ( event->button() != Qt::LeftButton ) return;
  // create new Station at point where user clicked scene
  qreal  x = event->scenePos().x();
  qreal  y = event->scenePos().y();
  addItem( new Station( x, y ) );
  // emit informative message
  emit message( QString("Station add at %1,%2").arg(x).arg(y) );
}
```



Station add at 64,30

6- Déplacement et suppression d’éléments

Ici, nous allons améliorer notre application afin de permettre à l'utilisateur de déplacer et supprimer des stations de radio préalablement placé. L'utilisateur sera en mesure de sélectionner une station en le pointant avec le curseur de la souris et en appuyant sur le bouton de gauche, tout en maintenant le bouton enfoncé déplacer la station en déplaçant la souris, et en libérant de la station en relâchant le bouton de la souris. Pour permettre à l'utilisateur de supprimer un poste, nous fournirons un menu contextuel lorsque l'utilisateur appuie sur le bouton droit tout en pointant à une station.

Pour implémenter cette fonctionnalité, nous ferons une simple modification de notre classe Station, et d'améliorer notre classe Scene en mettant en œuvre l'héritées QGraphicsScene ContextMenuEvent méthode virtuelle.

Amélioration de la `Station.cpp`

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

/*************************************************************/
/******** Represents a radio station in the simulation *******/
/*************************************************************/
Station::Station( qreal x, qreal y ) : QGraphicsItem()
{
  // set Station pixmap and position
  setPos( x, y );
  setFlags( QGraphicsItem::ItemIsMovable |
            QGraphicsItem::ItemIsSelectable |
            QGraphicsItem::ItemIgnoresTransformations );
}

/*********************** paint ************************/
void  Station::paint(QPainter *painter,
                     const QStyleOptionGraphicsItem *option,
                     QWidget *widget)
{
  // paint station symbol, must be smaller than bounding rectangle
  painter->setRenderHint( QPainter::Antialiasing );
  painter->setPen( QPen( Qt::black, 2 ) );
  painter->drawRect( -4,  -3,  8,   7 );
  painter->drawLine(  0,  -4,  0, -11 );
  painter->drawLine( -5, -11,  0,  -6 );
  painter->drawLine( +5, -11,  0,  -6 );
}
```

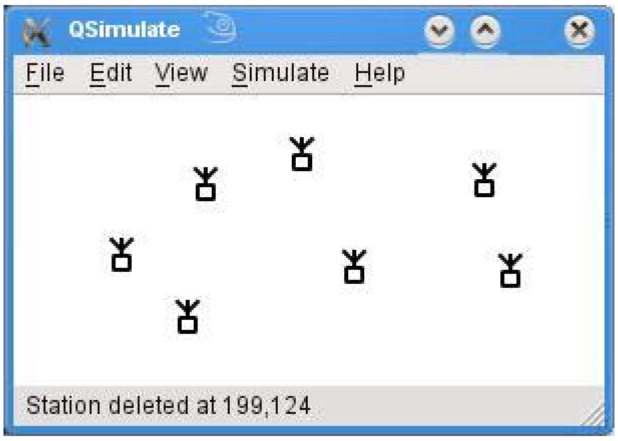
Amélioration de la `Scene.h`

```
#ifndef SCENE_H
#define SCENE_H
class QGraphicsSceneMouseEvent;

#include <QGraphicsScene>
/*************************************************************/
/******* Scene representing the simulated landscape **********/
/*************************************************************/

class Scene : public QGraphicsScene
{
  Q_OBJECT
public:
  Scene();   // constructor

signals:
  void  message(QString ); // info text message signal
protected:
  void mousePressEvent(QGraphicsSceneMouseEvent* );
  // receive mouse press events
  void contextMenuEvent(QGraphicsSceneContextMenuEvent*);
  // receive context menu events
};
#endif  // SCENE_H
```

Amélioration de la `Scene.cpp`

```
#include "scene.h"
#include "station.h"
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsSceneContextMenuEvent>
#include <QMenu>
#include <QAction>

/*******************************************************/
/***** Scene representing the simulated landscape ******/
/*******************************************************/
Scene::Scene() : QGraphicsScene()
{
  // create invisible item to provide default top-left anchor to scene
  addLine( 0, 0, 0, 1, QPen(Qt::transparent, 1) );
}

/***************** mousePressEvent ******************/
void  Scene::mousePressEvent( QGraphicsSceneMouseEvent* event )
{
  // set local variables and check if existing station clicked
  qreal           x = event->scenePos().x();
  qreal           y = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );
  // if station not clicked and right mouse button pressed, create new Station
  if ( station == 0 && event->button() == Qt::LeftButton )
  {
    addItem( new Station( x, y ) );
    emit message( QString("Station add at %1,%2").arg(x).arg(y) );
  }
  // call base mousePressEvent to handle other mouse press events
  QGraphicsScene::mousePressEvent( event );
}

/************ contextMenuEvent *********************/
void  Scene::contextMenuEvent( QGraphicsSceneContextMenuEvent* event )
{
  // we only want to display a menu if user clicked a station
  qreal     x       = event->scenePos().x();
  qreal     y       = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );
  if ( station == 0 ) return;

  // display context menu and action accordingly
  QMenu     menu;
  QAction*  deleteAction = menu.addAction("Delete Station");
  if ( menu.exec( event->screenPos() ) == deleteAction )
  {
    removeItem( station );
    delete station;
    emit message( QString("Station deleted at %1,%2").arg(x).arg(y) );
  }
}
```

En utilisant la méthode héritée `itemAt` nous vérifions si l'utilisateur est pointé sur une station, et seulement si l'utilisateur n'est pas orientée dans une station-nous ajoutons une nouvelle station. Nous appelons également la base QGraphicsScene mousePressEvent méthode pour utiliser la fonctionnalité héritée QGraphicsScene pour déplacer nos stations.



Station deleted at 199,124

7- Ajout d'une boite de dialogue

Toutes les applications ne doivent avoir la possibilité de permettre aux utilisateurs d'annuler et refaire commandes ou d'actions, mais si elle est à mettre en œuvre, il est plus facile si vous commencez tôt dans la conception de l'application et le cycle de mise en œuvre. Pour implémenter cette fonctionnalité dans notre application, nous utiliserons le Qt Undo framework.

Nous allons faire le travail de préparation en ajoutant une pile d'annulation et de défaire vue pile pour notre application. Cela se fera en améliorant notre classe `MainWindow` pour créer la pile d'annulation, ajouter certaines options de menu, et passer un pointeur vers la pile d'annulation de notre classe `Scene`.

Amélioration de la `MainWindow.h`

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
class Scene;
class QUndoStack;
class QUndoView;

#include <QMainWindow>

/*************************************************/
/******* Main application window for QSimulate ****/
/*************************************************/
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  MainWindow();                  // constructor

public slots:
  void showMessage( QString ); // show message on status bar
  void showUndoStack();        // open up undo stack window

private:
  Scene*       m_scene;      // scene representing the simulated landscape
  QUndoStack*  m_undoStack;  // undo stack for undo & redo of commands
  QUndoView*   m_undoView;   // undo stack window to view undo & redo commands
};
#endif  // MAINWINDOW_H
```

Amélioration de la `MainWindow.cpp`

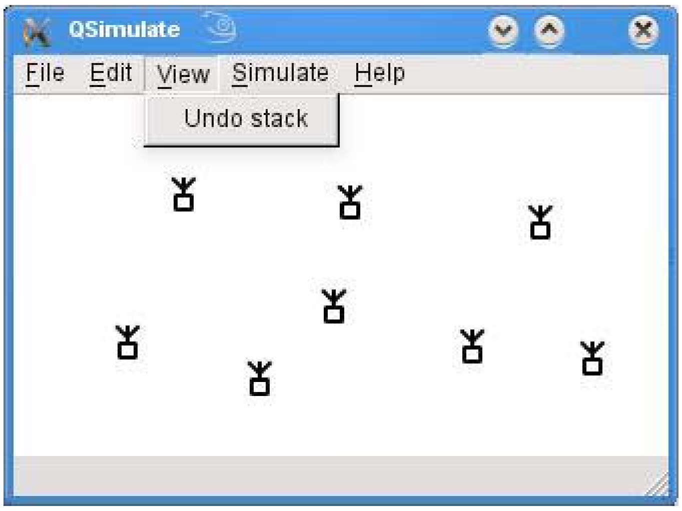
```
#include "mainwindow.h"
#include "scene.h"
#include <QMenuBar>
#include <QStatusBar>
#include <QGraphicsView>
#include <QUndoStack>
#include <QUndoView>

/*************************************************************/
/**** Main application window for QSimulate ****************/
/*************************************************************/
MainWindow::MainWindow() : QMainWindow()
{
  // add drop down menus
  menuBar()->addMenu( "&File" );
  QMenu*  editMenu = menuBar()->addMenu( "&Edit" );
  QMenu*  viewMenu = menuBar()->addMenu( "&View" );
  menuBar()->addMenu( "&Simulate" );
  menuBar()->addMenu( "&Help" );

  // create undo stack and associated menu actions
  m_undoStack = new QUndoStack( this );
  m_undoView  = 0;
  viewMenu->addAction( "Undo stack", this, SLOT(showUndoStack()) );
  QAction* undoAction = m_undoStack->createUndoAction( this );
  QAction* redoAction = m_undoStack->createRedoAction( this );
  undoAction->setShortcut( QKeySequence::Undo );
  redoAction->setShortcut( QKeySequence::Redo );
  editMenu->addAction( undoAction );
  editMenu->addAction( redoAction );

  // create scene and central widget view of scene
  m_scene               = new Scene( m_undoStack );
  QGraphicsView*   view = new QGraphicsView( m_scene );
  view->setAlignment( Qt::AlignLeft | Qt::AlignTop );
  view->setFrameStyle( 0 );
  setCentralWidget( view );

  // connect message signal from scene to showMessage slot
  connect(m_scene, SIGNAL(message(QString)), this, SLOT(showMessage(QString)));
  // add status bar message
  statusBar()->showMessage("QSimulate has started");
}

/************ showMessage ******************/
void  MainWindow::showMessage( QString msg )
{ // display message on main window status bar
  statusBar()->showMessage( msg );
}

/************ showUndoStack *****************/
void  MainWindow::showUndoStack()
{
  // open up undo stack window
  if ( m_undoView == 0 )
  {
    m_undoView = new QUndoView( m_undoStack );
    m_undoView->setWindowTitle( "QSimulate - Undo stack" );
    m_undoView->setAttribute( Qt::WA_QuitOnClose, false );
  }
  m_undoView->show();
}
```

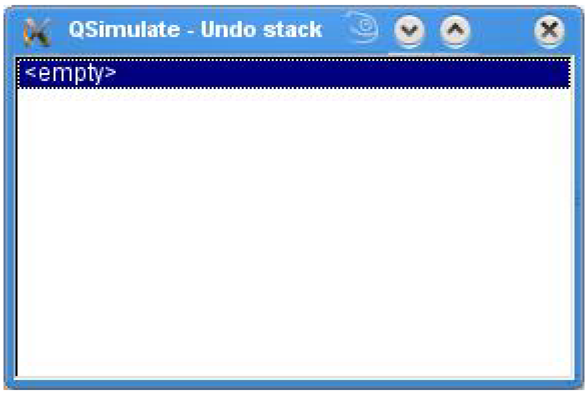
Amélioration de la `Scene.h`

```
#ifndef SCENE_H
#define SCENE_H
class QGraphicsSceneMouseEvent;
class QGraphicsSceneContextMenuEvent;
class QUndoStack;
#include <QGraphicsScene>

/**********************************************************/
/***** Scene representing the simulated landscape **********/
/**********************************************************/
class Scene : public QGraphicsScene
{
  Q_OBJECT
public:
  Scene( QUndoStack* );     // constructor

signals:
  void  message( QString ); // info text message signal

protected:
  void  mousePressEvent( QGraphicsSceneMouseEvent* );
  // receive mouse press events
  void  contextMenuEvent( QGraphicsSceneContextMenuEvent* );
  // receive context menu events

private:
  QUndoStack*  m_undoStack; // undo stack for undo & redo of commands
};
#endif  // SCENE_H
```

Amélioration de la `Scene.cpp`

```
#include "scene.h"
#include "station.h"
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsSceneContextMenuEvent>
#include <QMenu>
#include <QAction>
#include <QUndoStack>

/*********************************************************/
/****** Scene representing the simulated landscape **********/
/*********************************************************/
Scene::Scene(QUndoStack* undoStack ) : QGraphicsScene()
{
  // initialise variables
  m_undoStack     = undoStack;

  // create invisible item to provide default top-left anchor to scene
  addLine( 0, 0, 0, 1, QPen(Qt::transparent, 1) );
}

/************ mousePressEvent ******************/
void  Scene::mousePressEvent( QGraphicsSceneMouseEvent* event )
{
  // set local variables and check if existing station clicked
  qreal          x = event->scenePos().x();
  qreal          y = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );

  // if station not clicked and right mouse button pressed, create new Station
  if ( station == 0 && event->button() == Qt::LeftButton )
  {
    addItem( new Station( x, y ) );
    emit message( QString("Station add at %1,%2").arg(x).arg(y) );
  }

  // call base mousePressEvent to handle other mouse press events
  QGraphicsScene::mousePressEvent( event );
}

/************ contextMenuEvent ******************/
void  Scene::contextMenuEvent( QGraphicsSceneContextMenuEvent* event )
{
  // we only want to display a menu if user clicked a station
  qreal     x        = event->scenePos().x();
  qreal     y        = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );
  if ( station == 0 ) return;

  // display context menu and action accordingly
  QMenu     menu;
  QAction*  deleteAction = menu.addAction("Delete Station");
  if ( menu.exec( event->screenPos() ) == deleteAction )
  {
    removeItem( station );
    delete station;
    emit message( QString("Station deleted at %1,%2").arg(x).arg(y) );
  }
}
```





8- Implémentation de la commande Undo

Qt undo framework utilise le design pattern commande pour la mise en œuvre de la fonctionnalité undo / redo dans les applications. Le design pattern de commande est basé sur l'idée que toute l'édition dans une application se fait en créant des instances d'objets de commande. Chaque commande sait aussi comment annuler ses changements. Ici, nous allons mettre en œuvre les commandes undo / redo pour ajouter, déplacer et supprimer des postes en mettant en œuvre trois nouvelles classes dérivées de `QUndoCommand` appelé "`CommandStationAdd`", "`CommandStationMove`" et "`CommandStationDelete`".

La part importante à noter à propos des classes de commande est qu'elles fournissent des implémentations des fonctions undo et redo. Ce sont ces fonctions qui font le repaint, pas le constructeur de la classe ou le destructeur. Quand une instance d'un objet de commande est poussée sur la pile d'annulation, la fonction redo est appelée par le Qt undo framework pour faire la modification désirée. Aussi dans le constructeur une courte chaîne décrivant ce que la commande fait qui est affiché dans la pile d'annulation et sur l'undo / redo actions.
Nouvelle classe `CommandStationAdd.h`

```
#ifndef COMMANDSTATIONADD_H
#define COMMANDSTATIONADD_H
#include <QUndoCommand>
#include <QGraphicsScene>

/************************************************************/
/******* Undostack command for adding a station to a scene *****/
/************************************************************/
class CommandStationAdd : public QUndoCommand
{
public:
  CommandStationAdd( QGraphicsScene* scene, qreal x, qreal y )
    {
      m_station = new Station( x, y );
      m_scene   = scene;
      setText( QString("Station add %1,%2").arg(x).arg(y) );
    }

  ~CommandStationAdd()
    {
      // if station not on scene then delete station
      if ( !m_scene->items().contains( m_station ) )
        delete m_station;
    }

  virtual void undo()    { m_scene->removeItem( m_station ); }
  virtual void redo()    { m_scene->addItem( m_station ); }

private:
  Station*          m_station;     // station being added
  QGraphicsScene*   m_scene;       // scene where station being added
};
#endif  // COMMANDSTATIONADD_H
```

Nouvelle classe `CommandStationAdd.cpp`

```
#ifndef COMMANDSTATIONMOVE_H
#define COMMANDSTATIONMOVE_H
#include "station.h"
#include <QUndoCommand>

/************************************************************/
/***** Undostack command for moving a station on a scene ******/
/************************************************************/
class CommandStationMove : public QUndoCommand
{
public:
  CommandStationMove(Station* station, qreal fromX, qreal fromY, qreal toX,
qreal toY )
  {
    m_station = station;
    m_from    = QPointF( fromX, fromY );
    m_to      = QPointF( toX, toY );
    setText(QString("Station move %1,%2 -> %3,%4").arg(fromX).arg(fromY)
                                                  .arg(toX).arg(toY) );
  }

  virtual void undo()    { m_station->setPos( m_from ); }
  virtual void redo()    { m_station->setPos( m_to ); }

private:
  Station*   m_station;       // station being moved
  QPointF    m_from;          // original coords
  QPointF    m_to;            // new coords
};
#endif  // COMMANDSTATIONMOVE_H
```

Nouvelle classe `CommandStationAdd.cpp`

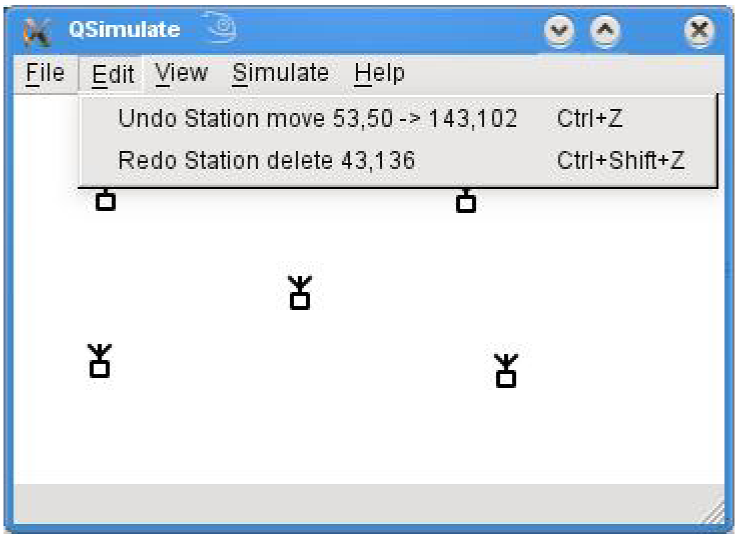
```
#ifndef COMMANDSTATIONDELETE_H
#define COMMANDSTATIONDELETE_H
#include <QUndoCommand>
#include <QGraphicsScene>

/*********************************************************************/
/****** Undostack command for deleting a station from a scene *********/
/*********************************************************************/
class CommandStationDelete : public QUndoCommand
{
public:
  CommandStationDelete( QGraphicsScene* scene, Station* station )
  {
    m_scene   = scene;
    m_station = station;
    setText(QString("Station delete %1,%2").arg(station->x()).arg(station->y())
);
  }

  virtual void undo()    { m_scene->addItem( m_station ); }
  virtual void redo()    { m_scene->removeItem( m_station ); }

private:
  QGraphicsScene*  m_scene;      // scene where station being deleted
  Station*         m_station;    // station being deleted
};
#endif  // COMMANDSTATIONDELETE_H
```

Amélioration de la `Scene.h`

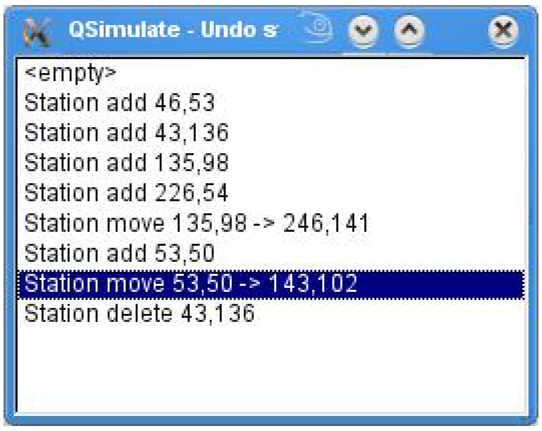
```
#ifndef SCENE_H
#define SCENE_H
class QGraphicsSceneMouseEvent;
class QGraphicsSceneContextMenuEvent;
class QUndoStack;
class Station;
#include <QGraphicsScene>

/******************************************************************/
/*********** Scene representing the simulated landscape ************/
/******************************************************************/
class Scene : public QGraphicsScene
{
  Q_OBJECT
public:
  Scene( QUndoStack* );       // constructor

signals:
  void  message( QString );   // info text message signal

public slots:
  void  selectStations();     // records selected stations & positions

protected:
  void mousePressEvent(QGraphicsSceneMouseEvent* );// receive mouse press events
  void mouseReleaseEvent(QGraphicsSceneMouseEvent*);
  // receive mouse release events
  void contextMenuEvent(QGraphicsSceneContextMenuEvent*);
  // receive context menu events

private:
  typedef QPair<Station*,QPointF> StationPos;
  QList<StationPos> m_stations; // currently selected stations & start positions
  QUndoStack* m_undoStack;      // undo stack for undo & redo of commands
};
#endif  // SCENE_H
```

Amélioration de la `Scene.cpp`

```
#include "scene.h"
#include "station.h"
#include "commandstationadd.h"
#include "commandstationdelete.h"
#include "commandstationmove.h"
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsSceneContextMenuEvent>
#include <QMenu>
#include <QAction>
#include <QUndoStack>

/*****************************************************/
/***** Scene representing the simulated landscape *******/
/*****************************************************/
Scene::Scene( QUndoStack* undoStack ) : QGraphicsScene()
{
  // initialise variables
  m_undoStack     = undoStack;

  // create invisible item to provide default top-left anchor to scene
  addLine( 0, 0, 0, 1, QPen(Qt::transparent, 1) );

  // connect selectionChanged signal to selectStations slot
  connect( this, SIGNAL(selectionChanged()), this, SLOT(selectStations()) );
}

/**************** mousePressEvent ****************/
void  Scene::mousePressEvent( QGraphicsSceneMouseEvent* event )
{
  // set local variables and check if existing station clicked
  qreal           x = event->scenePos().x();
  qreal           y = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );

  // if station not clicked and right mouse button pressed, create new Station
  if ( station == 0 && event->button() == Qt::LeftButton )
  {
    m_undoStack->push( new CommandStationAdd( this, x, y ) );
    emit message( QString("Station add at %1,%2").arg(x).arg(y) );
  }

  // call base mousePressEvent to handle other mousePressEvent such as selecting
  QGraphicsScene::mousePressEvent( event );
}

/**************** contextMenuEvent **************/
void  Scene::contextMenuEvent( QGraphicsSceneContextMenuEvent* event )
{
  // we only want to display a menu if user clicked a station
  qreal     x       = event->scenePos().x();
  qreal     y       = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );
  if ( station == 0 ) return;

  // display context menu and action accordingly
  QMenu     menu;
  QAction*  deleteAction = menu.addAction("Delete Station");
  if ( menu.exec( event->screenPos() ) == deleteAction )
  {
    m_undoStack->push( new CommandStationDelete( this, station ) );
    emit message( QString("Station deleted at %1,%2").arg(x).arg(y) );
  }
}

/*************** selectStations ********************/
void  Scene::selectStations()
{
  // refresh record of selected stations and their starting positions
  m_stations.clear();
  foreach( QGraphicsItem* item, selectedItems() )
    if ( dynamic_cast<Station*>( item ) )
    m_stations.append(qMakePair( dynamic_cast<Station*>( item ), item->pos()));
}

/****************** mouseReleaseEvent ****************/
void  Scene::mouseReleaseEvent( QGraphicsSceneMouseEvent* event )
{
  // if any stations moved, then create undo commands
  foreach( StationPos station , m_stations )
    if ( station.first->pos() != station.second )
      m_undoStack->push( new CommandStationMove( station.first,
                             station.second.x(), station.second.y(),
                             station.first->x(), station.first->y() ) );

  // refresh record of selected stations and call base mouseReleaseEvent
  selectStations();
  QGraphicsScene::mouseReleaseEvent( event );
}
```



Undo Station move 53,50 -> 143,102   Ctrl+Z
Redo Station delete 43,136   Ctrl+Shift+Z



```
<empty>
Station add 46,53
Station add 43,136
Station add 135,98
Station add 226,54
Station move 135,98 -> 246,141
Station add 53,50
Station move 53,50 -> 143,102
Station delete 43,136
```

9- Sauvegarde dans un fichier

Ici, nous allons permettre à l'utilisateur de sauvegarder leurs données de simulation dans un fichier au format XML. Nous ajoutons une action new File menu "Enregistrer sous ...", utilisez `QFileDialog` fonctionnalité de laisser l'utilisateur choisir le nom et l'emplacement, et d'utiliser la fonctionnalité `QXmlStreamWriter` pour écrire le fichier XML.

Amélioration de la `MainWindow.h`

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
class Scene;
class QUndoStack;
class QUndoView;
#include <QMainWindow>

/*************************************************************/
/***** Main application window for QSimulate ****************/
/*************************************************************/
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  MainWindow();                // constructor

public slots:
  void showMessage( QString ); // show message on status bar
  void showUndoStack();        // open up undo stack window
  bool fileSaveAs();     // save simulation to file returning true if successful

private:
  Scene*       m_scene;  // scene representing the simulated landscape
  QUndoStack*  m_undoStack;   // undo stack for undo & redo of commands
  QUndoView*   m_undoView;    // undo stack window to view undo & redo commands
};
#endif  // MAINWINDOW_H
```

Amélioration de la `MainWindow.cpp`

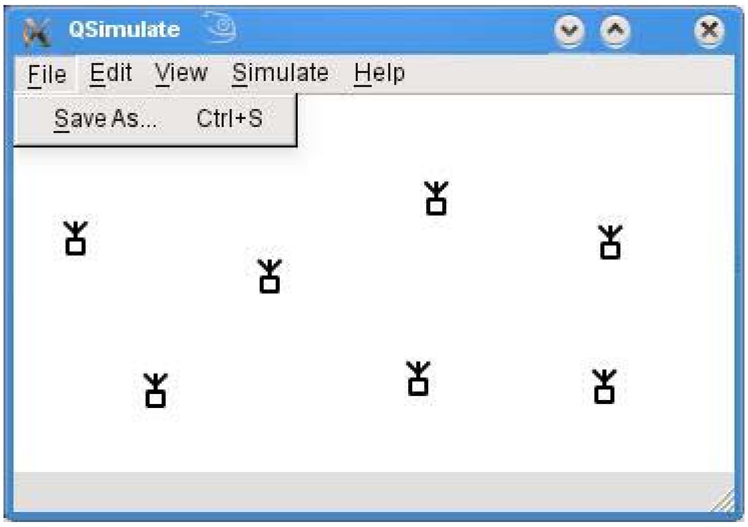
```
#include "mainwindow.h"
#include "scene.h"
#include <QMenuBar>
#include <QStatusBar>
#include <QGraphicsView>
#include <QUndoStack>
#include <QUndoView>
#include <QFileDialog>
#include <QXmlStreamWriter>
#include <QDateTime>

/*************************************************************/
/*************** Main application window for QSimulate *******/
/*************************************************************/
MainWindow::MainWindow() : QMainWindow()
{
  // add drop down menus
  QMenu*  fileMenu = menuBar()->addMenu( "&File" );
  QMenu*  editMenu = menuBar()->addMenu( "&Edit" );
  QMenu*  viewMenu = menuBar()->addMenu( "&View" );
  menuBar()->addMenu( "&Simulate" );
  menuBar()->addMenu( "&Help" );
  // create file menu options
  QAction* saveAction = fileMenu->addAction("&Save As...", this,
                                            SLOT(fileSaveAs()) );
  saveAction->setShortcut( QKeySequence::Save );

  // create undo stack and associated menu actions
  m_undoStack = new QUndoStack( this );
  m_undoView  = 0;
  viewMenu->addAction( "Undo stack", this, SLOT(showUndoStack()) );
  QAction* undoAction = m_undoStack->createUndoAction( this );
  QAction* redoAction = m_undoStack->createRedoAction( this );
  undoAction->setShortcut( QKeySequence::Undo );
  redoAction->setShortcut( QKeySequence::Redo );
  editMenu->addAction( undoAction );
  editMenu->addAction( redoAction );

  // create scene and central widget view of scene
  m_scene               = new Scene( m_undoStack );
  QGraphicsView*   view = new QGraphicsView( m_scene );
  view->setAlignment( Qt::AlignLeft | Qt::AlignTop );
  view->setFrameStyle( 0 );
  setCentralWidget( view );

  // connect message signal from scene to showMessage slot
  connect(m_scene, SIGNAL(message(QString)), this, SLOT(showMessage(QString)));

  // add status bar message
  statusBar()->showMessage("QSimulate has started");
}

/**************** showMessage ********************/
void  MainWindow::showMessage( QString msg )
{
  // display message on main window status bar
  statusBar()->showMessage( msg );
}

/***************** showUndoStack ******************/
void  MainWindow::showUndoStack()
{
  // open up undo stack window
  if ( m_undoView == 0 )
  {
    m_undoView = new QUndoView( m_undoStack );
    m_undoView->setWindowTitle( "QSimulate - Undo stack" );
    m_undoView->setAttribute( Qt::WA_QuitOnClose, false );
  }
  m_undoView->show();
}

/****************** fileSaveAs ********************/
bool  MainWindow::fileSaveAs()
{
  // get user to select filename and location
  QString filename = QFileDialog::getSaveFileName();
  if ( filename.isEmpty() ) return false;

  // open the file and check we can write to it
  QFile file( filename );
  if ( !file.open( QIODevice::WriteOnly ) )
  {
    showMessage( QString("Failed to write to '%1'").arg(filename) );
    return false;
  }

  // open an xml stream writer and write simulation data
  QXmlStreamWriter  stream( &file );
  stream.setAutoFormatting( true );
  stream.writeStartDocument();
  stream.writeStartElement( "qsimulate" );
  stream.writeAttribute("version", "2009-05" );
  stream.writeAttribute("user", QString(getenv("USERNAME")) );
  stream.writeAttribute("when",
                        QDateTime::currentDateTime().toString(Qt::ISODate) );
  m_scene->writeStream( &stream );
  stream.writeEndDocument();

  // close the file and display useful message
  file.close();
  showMessage( QString("Saved to '%1'").arg(filename) );
  return true;
}
```

Amélioration de la `Scene.h`

```
#ifndef SCENE_H
#define SCENE_H
class QGraphicsSceneMouseEvent;
class QGraphicsSceneContextMenuEvent;
class QUndoStack;
class QXmlStreamWriter;
class Station;
#include <QGraphicsScene>

/************************************************************/
/********* Scene representing the simulated landscape ********/
/************************************************************/
class Scene : public QGraphicsScene
{
  Q_OBJECT
public:
  Scene( QUndoStack* );                    // constructor
  void  writeStream( QXmlStreamWriter* ); // write scene data to xml stream

signals:
  void  message( QString );               // info text message signal

public slots:
  void  selectStations();                 // records selected stations &
positions

protected:
  void mousePressEvent(QGraphicsSceneMouseEvent* );// receive mouse press events
  void mouseReleaseEvent(QGraphicsSceneMouseEvent* );
  // receive mouse release events
  void contextMenuEvent(QGraphicsSceneContextMenuEvent* );
  // receive context menu events

private:
  typedef QPair<Station*,QPointF> StationPos;
  QList<StationPos> m_stations; // currently selected stations & start positions
  QUndoStack* m_undoStack;      // undo stack for undo & redo of commands
};
#endif  // SCENE_H
```

Amélioration de la `Scene.cpp`

```
#include "scene.h"
#include "station.h"
#include "commandstationadd.h"
#include "commandstationdelete.h"
#include "commandstationmove.h"
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsSceneContextMenuEvent>
#include <QMenu>
#include <QAction>
#include <QUndoStack>
#include <QXmlStreamWriter>

/*************************************************************/
/***** Scene representing the simulated landscape ***********/
/*************************************************************/
Scene::Scene( QUndoStack* undoStack ) : QGraphicsScene()
{
  // initialise variables
  m_undoStack     = undoStack;

  // create invisible item to provide default top-left anchor to scene
  addLine( 0, 0, 0, 1, QPen(Qt::transparent, 1) );

  // connect selectionChanged signal to selectStations slot
  connect( this, SIGNAL(selectionChanged()), this, SLOT(selectStations()) );
}

/***************** mousePressEvent *************/
void  Scene::mousePressEvent( QGraphicsSceneMouseEvent* event )
{
  // set local variables and check if existing station clicked
  qreal           x = event->scenePos().x();
  qreal           y = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );

  // if station not clicked and right mouse button pressed, create new Station
  if ( station == 0 && event->button() == Qt::LeftButton )
  {
    m_undoStack->push( new CommandStationAdd( this, x, y ) );
    emit message( QString("Station add at %1,%2").arg(x).arg(y) );
  }

  // call base mousePressEvent to handle other mousePressEvent such as selecting
  QGraphicsScene::mousePressEvent( event );
}

/****************** contextMenuEvent ***************/
void  Scene::contextMenuEvent( QGraphicsSceneContextMenuEvent* event )
{
  // we only want to display a menu if user clicked a station
  qreal     x       = event->scenePos().x();
  qreal     y       = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );
  if ( station == 0 ) return;

  // display context menu and action accordingly
  QMenu     menu;
  QAction*  deleteAction = menu.addAction("Delete Station");
  if ( menu.exec( event->screenPos() ) == deleteAction )
  {
    m_undoStack->push( new CommandStationDelete( this, station ) );
    emit message( QString("Station deleted at %1,%2").arg(x).arg(y) );
  }
}

/******************** selectStations ***************/
void  Scene::selectStations()
{
  // refresh record of selected stations and their starting positions
  m_stations.clear();
  foreach( QGraphicsItem* item, selectedItems() )
    if ( dynamic_cast<Station*>( item ) )
    m_stations.append(qMakePair(dynamic_cast<Station*>(item ), item->pos() ) );
}

/****************** mouseReleaseEvent ****************/
void  Scene::mouseReleaseEvent( QGraphicsSceneMouseEvent* event )
{
  // if any stations moved, then create undo commands
  foreach( StationPos station , m_stations )
    if ( station.first->pos() != station.second )
      m_undoStack->push( new CommandStationMove( station.first,
                            station.second.x(), station.second.y(),
                            station.first->x(), station.first->y() ) );

  // refresh record of selected stations and call base mouseReleaseEvent
  selectStations();
  QGraphicsScene::mouseReleaseEvent( event );
}

/****************** writeStream *********************/
void  Scene::writeStream( QXmlStreamWriter* stream )
{
  // write station data to xml stream
  foreach( QGraphicsItem*  item, items() )
  {
    Station*  station = dynamic_cast<Station*>( item );
    if ( station )
    {
      stream->writeEmptyElement( "station" );
      stream->writeAttribute( "x", QString("%1").arg(station->x()) );
      stream->writeAttribute( "y", QString("%1").arg(station->y()) );
    }
  }
}
```



```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<qsimulate version="2009-05" user="crookr" when="2009-11-06T23:38:47">
   <station x="71" y="151"/>
   <station x="128" y="94"/>
   <station x="202" y="145"/>
   <station x="211" y="55"/>
   <station x="298" y="77"/>
   <station x="295" y="149"/>
   <station x="31" y="75"/>
</qsimulate>
```

10- Chargement depuis un fichier

Ici, nous allons permettre à l'utilisateur de charger des données de simulation à partir de fichiers au format XML enregistré précédemment. Nous ajoutons une action new File menu "Ouvrir ...", utiliser la fonctionnalité `QFileDialog` de laisser l'utilisateur sélectionner le fichier à lire, et d'utiliser la fonctionnalité `QXmlStreamReader` de lire le contenu du fichier XML.

Amélioration de la `MainWindow.h`

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
class Scene;
class QUndoStack;
class QUndoView;
#include <QMainWindow>

/*************************************************************/
/******** Main application window for QSimulate *************/
/*************************************************************/
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  MainWindow();                  // constructor

public slots:
  void showMessage( QString );   // show message on status bar
  void showUndoStack();          // open up undo stack window
  bool fileSaveAs();     // save simulation to file returning true if successful
  bool fileOpen();       // load simulation file returning true if successful

private:
  Scene*       m_scene;     // scene representing the simulated landscape
  QUndoStack*  m_undoStack; // undo stack for undo & redo of commands
  QUndoView*   m_undoView;  // undo stack window to view undo & redo commands
};
#endif  // MAINWINDOW_H
```

Amélioration de la `MainWindow.cpp`

```
#include "mainwindow.h"
#include "scene.h"
#include <QMenuBar>
#include <QStatusBar>
#include <QGraphicsView>
#include <QUndoStack>
#include <QUndoView>
#include <QFileDialog>
#include <QXmlStreamWriter>
#include <QXmlStreamReader>
#include <QDateTime>
```

```
/*********************************************************************/
/******** Main application window for QSimulate ********************/
/*********************************************************************/
MainWindow::MainWindow() : QMainWindow()
{
  // add drop down menus
  QMenu*  fileMenu = menuBar()->addMenu( "&File" );
  QMenu*  editMenu = menuBar()->addMenu( "&Edit" );
  QMenu*  viewMenu = menuBar()->addMenu( "&View" );
  menuBar()->addMenu( "&Simulate" );
  menuBar()->addMenu( "&Help" );

  // create file menu options
  QAction* saveAction = fileMenu->addAction("&Save As...", this,
SLOT(fileSaveAs()) );
  QAction* openAction = fileMenu->addAction("&Open ...", this,SLOT(fileOpen()));
  saveAction->setShortcut( QKeySequence::Save );
  openAction->setShortcut( QKeySequence::Open );

  // create undo stack and associated menu actions
  m_undoStack = new QUndoStack( this );
  m_undoView  = 0;
  viewMenu->addAction( "Undo stack", this, SLOT(showUndoStack()) );
  QAction* undoAction = m_undoStack->createUndoAction( this );
  QAction* redoAction = m_undoStack->createRedoAction( this );
  undoAction->setShortcut( QKeySequence::Undo );
  redoAction->setShortcut( QKeySequence::Redo );
  editMenu->addAction( undoAction );
  editMenu->addAction( redoAction );

  // create scene and central widget view of scene
  m_scene              = new Scene( m_undoStack );
  QGraphicsView*   view = new QGraphicsView( m_scene );
  view->setAlignment( Qt::AlignLeft | Qt::AlignTop );
  view->setFrameStyle( 0 );
  setCentralWidget( view );

  // connect message signal from scene to showMessage slot
  connect(m_scene, SIGNAL(message(QString)), this, SLOT(showMessage(QString)));

  // add status bar message
  statusBar()->showMessage("QSimulate has started");
}

/***************** showMessage *****************/
void  MainWindow::showMessage( QString msg )
{
  // display message on main window status bar
  statusBar()->showMessage( msg );
}

/****************** showUndoStack **************/
void  MainWindow::showUndoStack()
{
  // open up undo stack window
  if ( m_undoView == 0 )
  {
    m_undoView = new QUndoView( m_undoStack );
    m_undoView->setWindowTitle( "QSimulate - Undo stack" );
    m_undoView->setAttribute( Qt::WA_QuitOnClose, false );
  }
  m_undoView->show();
}
```

```
/********************* fileSaveAs ***************/
bool  MainWindow::fileSaveAs()
{
  // get user to select filename and location
  QString filename = QFileDialog::getSaveFileName();
  if ( filename.isEmpty() ) return false;

  // open the file and check we can write to it
  QFile file( filename );
  if ( !file.open( QIODevice::WriteOnly ) )
  {
    showMessage( QString("Failed to write to '%1'").arg(filename) );
    return false;
  }

  // open an xml stream writer and write simulation data
  QXmlStreamWriter  stream( &file );
  stream.setAutoFormatting( true );
  stream.writeStartDocument();
  stream.writeStartElement( "qsimulate" );
  stream.writeAttribute( "version", "2009-05" );
  stream.writeAttribute( "user", QString(getenv("USERNAME")) );
  stream.writeAttribute( "when",
                         QDateTime::currentDateTime().toString(Qt::ISODate) );
  m_scene->writeStream( &stream );
  stream.writeEndDocument();

  // close the file and display useful message
  file.close();
  showMessage( QString("Saved to '%1'").arg(filename) );
  return true;
}

/*********************** fileOpen *******************/
bool  MainWindow::fileOpen()
{
  // get user to select filename and location
  QString filename = QFileDialog::getOpenFileName();
  if ( filename.isEmpty() ) return false;

  // open the file and check we can read from it
  QFile file( filename );
  if ( !file.open( QIODevice::ReadOnly ) )
  {
    showMessage( QString("Failed to open '%1'").arg(filename) );
    return false;
  }

  // open an xml stream reader and load simulation data
  QXmlStreamReader  stream( &file );
  Scene*            newScene = new Scene( m_undoStack );
  while ( !stream.atEnd() )
  {
    stream.readNext();
    if ( stream.isStartElement() )
    {
      if ( stream.name() == "qsimulate" )
        newScene->readStream( &stream );
      else
        stream.raiseError(QString("Unrecognised element '%1'").arg(stream.name()
                                                                   .toString()) );
    }
  }
  // check if error occured
  if ( stream.hasError() )
  {
    file.close();
    showMessage(QString("Failed to load '%1' (%2)").arg(filename)
                                                 .arg(stream.errorString()) );
    delete newScene;
    return false;
  }

  // close file, display new scene, delete old scene, and display useful message
  file.close();
  m_undoStack->clear();
  QGraphicsView*   view = dynamic_cast<QGraphicsView*>( centralWidget() );
  view->setScene( newScene );
  delete m_scene;
  m_scene = newScene;
  showMessage( QString("Loaded '%1'").arg(filename) );
  return true;
}
```

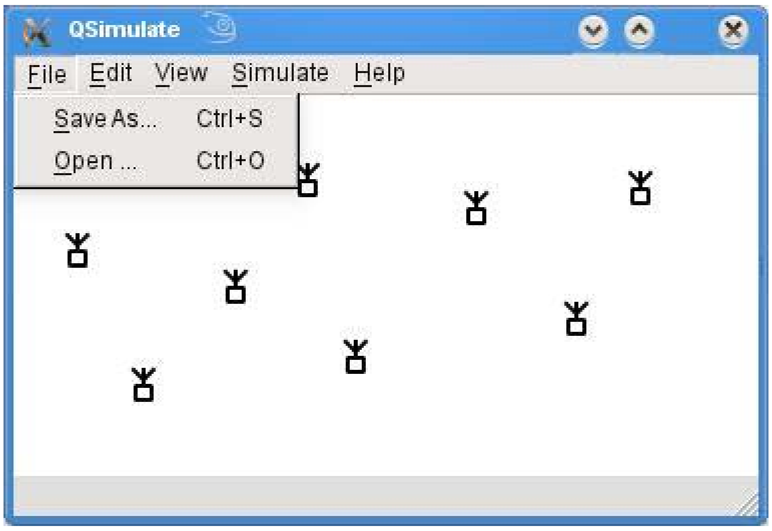
Amélioration de la `Scene.h`

```
#ifndef SCENE_H
#define SCENE_H
class QGraphicsSceneMouseEvent;
class QGraphicsSceneContextMenuEvent;
class QUndoStack;
class QXmlStreamWriter;
class QXmlStreamReader;
class Station;
#include <QGraphicsScene>

/*********************************************************/
/***** Scene representing the simulated landscape *********/
/*********************************************************/
class Scene : public QGraphicsScene
{
  Q_OBJECT
public:
  Scene( QUndoStack* );                   // constructor
  void  writeStream( QXmlStreamWriter* ); // write scene data to xml stream
  void  readStream( QXmlStreamReader* );  // read scene data from xml stream

signals:
  void  message( QString );               // info text message signal

public slots:
  void  selectStations();              // records selected stations & positions

protected:
  void mousePressEvent(QGraphicsSceneMouseEvent*); // receive mouse press events
  void mouseReleaseEvent(QGraphicsSceneMouseEvent*);
      // receive mouse release events
  void contextMenuEvent(QGraphicsSceneContextMenuEvent* );
      // receive context menu events

private:
  typedef QPair<Station*,QPointF> StationPos;
  QList<StationPos> m_stations; // currently selected stations & start positions
  QUndoStack*  m_undoStack;     // undo stack for undo & redo of commands
};
#endif  // SCENE_H
```

Amélioration de la `Scene.cpp`

```
#include "scene.h"
#include "station.h"
#include "commandstationadd.h"
#include "commandstationdelete.h"
#include "commandstationmove.h"
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsSceneContextMenuEvent>
#include <QMenu>
#include <QAction>
#include <QUndoStack>
#include <QXmlStreamWriter>
#include <QXmlStreamReader>

/************************************************************/
/***** Scene representing the simulated landscape ************/
/************************************************************/
Scene::Scene( QUndoStack* undoStack ) : QGraphicsScene()
{
  // initialise variables
  m_undoStack     = undoStack;

  // create invisible item to provide default top-left anchor to scene
  addLine( 0, 0, 0, 1, QPen(Qt::transparent, 1) );

  // connect selectionChanged signal to selectStations slot
  connect( this, SIGNAL(selectionChanged()), this, SLOT(selectStations()) );
}

/********************* mousePressEvent ******************/
void  Scene::mousePressEvent( QGraphicsSceneMouseEvent* event )
{
  // set local variables and check if existing station clicked
  qreal           x = event->scenePos().x();
  qreal           y = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );

  // if station not clicked and right mouse button pressed, create new Station
  if ( station == 0 && event->button() == Qt::LeftButton )
  {
    m_undoStack->push( new CommandStationAdd( this, x, y ) );
    emit message( QString("Station add at %1,%2").arg(x).arg(y) );
  }

  // call base mousePressEvent to handle other mousePressEvent such as selecting
  QGraphicsScene::mousePressEvent( event );
}

/****************** contextMenuEvent *****************/
void  Scene::contextMenuEvent( QGraphicsSceneContextMenuEvent* event )
{
  // we only want to display a menu if user clicked a station
  qreal     x       = event->scenePos().x();
  qreal     y       = event->scenePos().y();
  Station*  station = dynamic_cast<Station*>( itemAt( x, y ) );
  if ( station == 0 ) return;

  // display context menu and action accordingly
  QMenu     menu;
  QAction*  deleteAction = menu.addAction("Delete Station");
  if ( menu.exec( event->screenPos() ) == deleteAction )
  {
    m_undoStack->push( new CommandStationDelete( this, station ) );
    emit message( QString("Station deleted at %1,%2").arg(x).arg(y) );
  }
}

/******************** selectStations ******************/
void  Scene::selectStations()
{
  // refresh record of selected stations and their starting positions
  m_stations.clear();
  foreach( QGraphicsItem* item, selectedItems() )
    if ( dynamic_cast<Station*>( item ) )
    m_stations.append( qMakePair( dynamic_cast<Station*>( item ), item->pos() )
);
}

/***************** mouseReleaseEvent *****************/
void  Scene::mouseReleaseEvent( QGraphicsSceneMouseEvent* event )
{
  // if any stations moved, then create undo commands
  foreach( StationPos station , m_stations )
    if ( station.first->pos() != station.second )
      m_undoStack->push( new CommandStationMove( station.first,
                              station.second.x(), station.second.y(),
                              station.first->x(), station.first->y() ) );

  // refresh record of selected stations and call base mouseReleaseEvent
  selectStations();
  QGraphicsScene::mouseReleaseEvent( event );
}

/********************** writeStream ********************/
void  Scene::writeStream( QXmlStreamWriter* stream )
{
  // write station data to xml stream
  foreach( QGraphicsItem*  item, items() )
  {
    Station*  station = dynamic_cast<Station*>( item );
    if ( station )
    {
      stream->writeEmptyElement( "station" );
      stream->writeAttribute( "x", QString("%1").arg(station->x()) );
      stream->writeAttribute( "y", QString("%1").arg(station->y()) );
    }
  }
}

/********************** readStream ********************/
void  Scene::readStream( QXmlStreamReader* stream )
{
  // read station data from xml stream
  while ( !stream->atEnd() )
  {
    stream->readNext();
    if ( stream->isStartElement() && stream->name() == "station" )
    {
      qreal x = 0.0, y = 0.0;
      foreach( QXmlStreamAttribute attribute, stream->attributes() )
      {
        if (attribute.name() == "x" )
          x = attribute.value().toString().toDouble();
        if (attribute.name() == "y" )
```

```
          y = attribute.value().toString().toDouble();
        }
        addItem( new Station( x, y ) );
      }
    }
}
```



11- Impression et visualisation

Afin de permettre l'impression et la génération d'un aperçu avant impression nous avons seulement besoin d'apporter des améliorations à notre classe `MainWindow`. Notre sortie imprimée sera une simple page montrant le paysage simulé comme le montre notre scène et certaines étiquettes sur les coins de la page. Le difficile travail sera effectué par le `QPrinter` Qt classes, `QPrintPreviewDialog` et `QPrintDialog`. Nous allons aussi ajouter deux nouvelles actions du menu Fichier "Aperçu avant impression ..." et "Imprimer ...".

Amélioration de la `MainWindow.h`

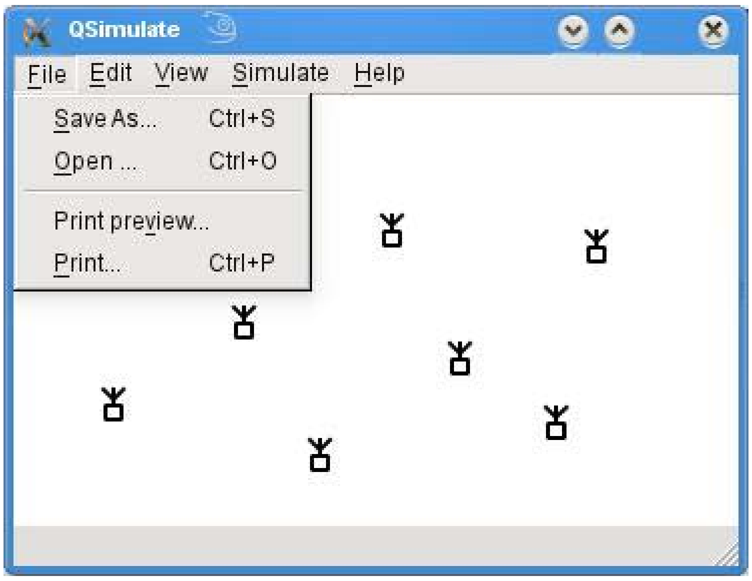
```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
class Scene;
class QUndoStack;
class QUndoView;
class QPrinter;
#include <QMainWindow>

/***********************************************************/
/************ Main application window for QSimulate ******/
/***********************************************************/
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  MainWindow();                  // constructor

public slots:
  void showMessage( QString ); // show message on status bar
  void showUndoStack();          // open up undo stack window
  bool fileSaveAs();      // save simulation to file returning true if successful
  bool fileOpen();        // load simulation file returning true if successful
  void filePrintPreview();       // display print preview dialog
  void filePrint();              // display print dialog
  void print( QPrinter* );       // draw print page

private:
  Scene*      m_scene;        // scene representing the simulated landscape
  QUndoStack* m_undoStack;    // undo stack for undo & redo of commands
  QUndoView*  m_undoView;     // undo stack window to view undo & redo commands
};
#endif  // MAINWINDOW_H
```

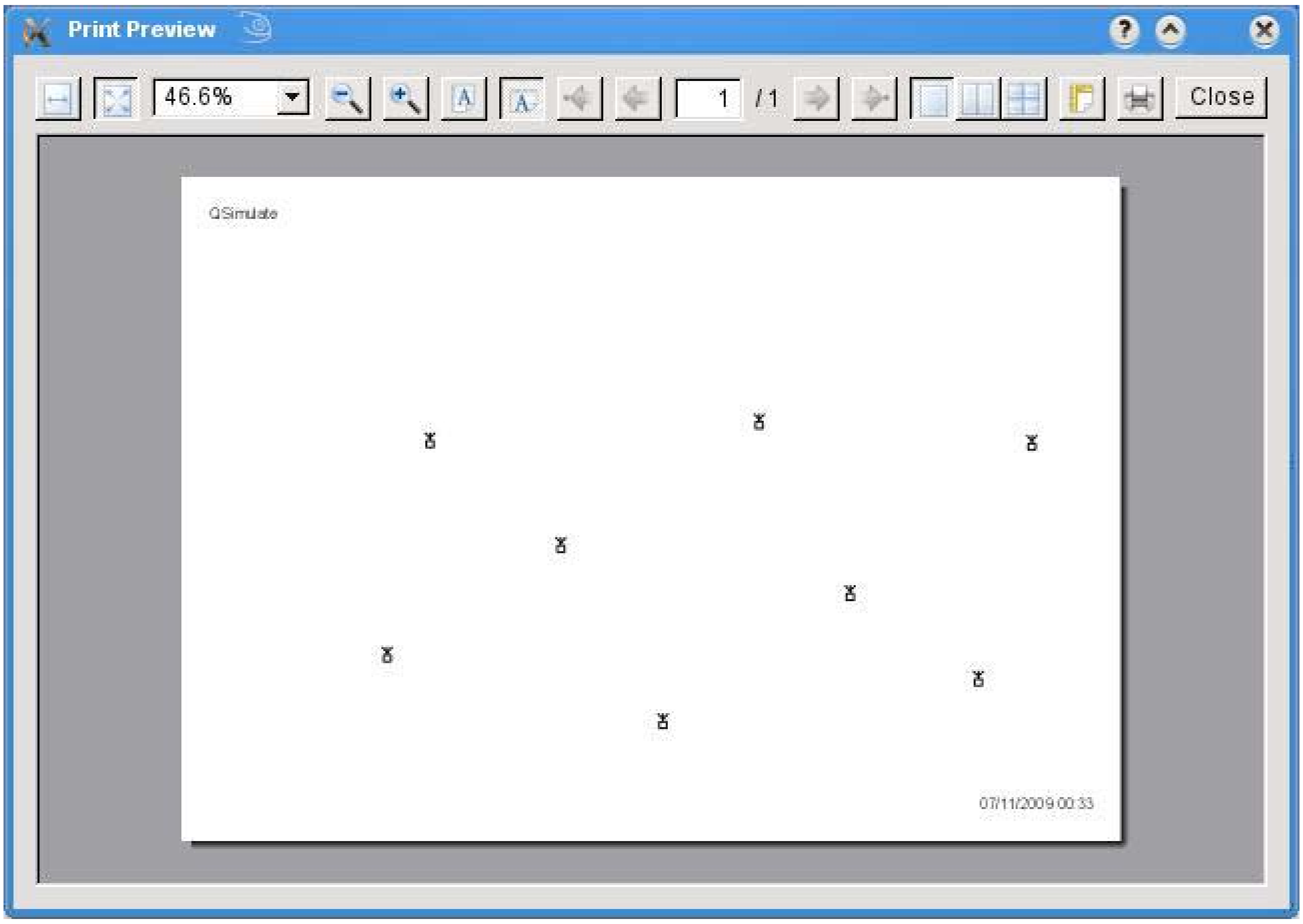
Amélioration de la `MainWindow.cpp`

```
#include "mainwindow.h"
#include "scene.h"
#include <QMenuBar>
#include <QStatusBar>
#include <QGraphicsView>
#include <QUndoStack>
#include <QUndoView>
#include <QFileDialog>
#include <QXmlStreamWriter>
#include <QXmlStreamReader>
#include <QDateTime>
#include <QPrinter>
#include <QPrintPreviewDialog>
#include <QPrintDialog>

/*************************************************************/
/********* Main application window for QSimulate ************/
/*************************************************************/
MainWindow::MainWindow() : QMainWindow()
{
  // add drop down menus
  QMenu*  fileMenu = menuBar()->addMenu( "&File" );
  QMenu*  editMenu = menuBar()->addMenu( "&Edit" );
  QMenu*  viewMenu = menuBar()->addMenu( "&View" );
  menuBar()->addMenu( "&Simulate" );
  menuBar()->addMenu( "&Help" );

  // create file menu options
  QAction* saveAction = fileMenu->addAction("&Save As...", this,
                                            SLOT(fileSaveAs()) );
  QAction* openAction = fileMenu->addAction("&Open ...", this,SLOT(fileOpen()));
  fileMenu->addSeparator();
  QAction* previewAction = fileMenu->addAction("Print pre&view...", this,
                                               SLOT(filePrintPreview()) );
  QAction* printAction = fileMenu->addAction("&Print...", this,
                                             SLOT(filePrint()) );
  saveAction->setShortcut( QKeySequence::Save );
  openAction->setShortcut( QKeySequence::Open );
  printAction->setShortcut( QKeySequence::Print );

  // create undo stack and associated menu actions
  m_undoStack = new QUndoStack( this );
  m_undoView  = 0;
  viewMenu->addAction( "Undo stack", this, SLOT(showUndoStack()) );
  QAction* undoAction = m_undoStack->createUndoAction( this );
  QAction* redoAction = m_undoStack->createRedoAction( this );
  undoAction->setShortcut( QKeySequence::Undo );
  redoAction->setShortcut( QKeySequence::Redo );
  editMenu->addAction( undoAction );
  editMenu->addAction( redoAction );

  // create scene and central widget view of scene
  m_scene              = new Scene( m_undoStack );
  QGraphicsView*   view = new QGraphicsView( m_scene );
  view->setAlignment( Qt::AlignLeft | Qt::AlignTop );
  view->setFrameStyle( 0 );
  setCentralWidget( view );

  // connect message signal from scene to showMessage slot
  connect(m_scene, SIGNAL(message(QString)), this, SLOT(showMessage(QString)));

  // add status bar message
  statusBar()->showMessage("QSimulate has started");
}

/************** showMessage ****************/
void  MainWindow::showMessage( QString msg )
{
  // display message on main window status bar
  statusBar()->showMessage( msg );
}

/*************** showUndoStack **************/
void  MainWindow::showUndoStack()
{
  // open up undo stack window
  if ( m_undoView == 0 )
  {
    m_undoView = new QUndoView( m_undoStack );
    m_undoView->setWindowTitle( "QSimulate - Undo stack" );
    m_undoView->setAttribute( Qt::WA_QuitOnClose, false );
  }
  m_undoView->show();
}

/**************** fileSaveAs *****************/
bool  MainWindow::fileSaveAs()
{
  // get user to select filename and location
  QString filename = QFileDialog::getSaveFileName();
  if ( filename.isEmpty() ) return false;

  // open the file and check we can write to it
  QFile file( filename );
  if ( !file.open( QIODevice::WriteOnly ) )
  {
    showMessage( QString("Failed to write to '%1'").arg(filename) );
    return false;
  }

  // open an xml stream writer and write simulation data
  QXmlStreamWriter  stream( &file );
  stream.setAutoFormatting( true );
  stream.writeStartDocument();
  stream.writeStartElement( "qsimulate" );
  stream.writeAttribute( "version", "2009-05" );
  stream.writeAttribute( "user", QString(getenv("USERNAME")) );
  stream.writeAttribute( "when",
                         QDateTime::currentDateTime().toString(Qt::ISODate) );
  m_scene->writeStream( &stream );
  stream.writeEndDocument();

  // close the file and display useful message
  file.close();
  showMessage( QString("Saved to '%1'").arg(filename) );
  return true;
}
```

```
/****************** fileOpen ********************/
bool  MainWindow::fileOpen()
{
  // get user to select filename and location
  QString filename = QFileDialog::getOpenFileName();
  if ( filename.isEmpty() ) return false;

  // open the file and check we can read from it
  QFile file( filename );
  if ( !file.open( QIODevice::ReadOnly ) )
  {
    showMessage( QString("Failed to open '%1'").arg(filename) );
    return false;
  }

  // open an xml stream reader and load simulation data
  QXmlStreamReader  stream( &file );
  Scene*            newScene = new Scene( m_undoStack );
  while ( !stream.atEnd() )
  {
    stream.readNext();
    if ( stream.isStartElement() )
    {
      if ( stream.name() == "qsimulate" )
        newScene->readStream( &stream );
      else
        stream.raiseError(QString("Unrecognised element '%1'").arg(stream.name()
                                                             .toString()) );
    }
  }

  // check if error occured
  if ( stream.hasError() )
  {
    file.close();
    showMessage(QString("Failed to load '%1' (%2)").arg(filename)
                                                  .arg(stream.errorString()) );
    delete newScene;
    return false;
  }

  // close file, display new scene, delete old scene, and display useful message
  file.close();
  m_undoStack->clear();
  QGraphicsView*   view = dynamic_cast<QGraphicsView*>( centralWidget() );
  view->setScene( newScene );
  delete m_scene;
  m_scene = newScene;
  showMessage( QString("Loaded '%1'").arg(filename) );
  return true;
}

/************* filePrintPreview *****************/
void  MainWindow::filePrintPreview()
{
  // display print preview dialog
  QPrinter printer(QPrinter::ScreenResolution ); // QPrinter::HighResolution );
  QPrintPreviewDialog  preview( &printer, this );
  connect(&preview, SIGNAL(paintRequested(QPrinter*)), SLOT(print(QPrinter*)));
  preview.exec();
}

/************** filePrint ****************/
```

```
void  MainWindow::filePrint()
{
  // display print dialog and if accepted print
  QPrinter       printer( QPrinter::ScreenResolution );
  QPrintDialog   dialog( &printer, this );
  if ( dialog.exec() == QDialog::Accepted ) print( &printer );
}

/**************** print *****************/
void  MainWindow::print( QPrinter* printer )
{
  // create painter for drawing print page
  QPainter painter( printer );
  int      w = printer->pageRect().width();
  int      h = printer->pageRect().height();
  QRect    page( 0, 0, w, h );

  // create a font appropriate to page size
  QFont    font = painter.font();
  font.setPixelSize( (w+h) / 100 );
  painter.setFont( font );

  // draw labels in corners of page
  painter.drawText(page, Qt::AlignTop    | Qt::AlignLeft, "QSimulate" );
  painter.drawText(page, Qt::AlignBottom | Qt::AlignLeft,
          QString(getenv("USERNAME")) );
  painter.drawText(page, Qt::AlignBottom | Qt::AlignRight,
          QDateTime::currentDateTime().toString(Qt::DefaultLocaleShortDate ) );

  // draw simulated landscape
  page.adjust( w/20, h/20, -w/20, -h/20 );
  m_scene->render( &painter, page );
}
```





12- Fonction New et Quit

Dans cette partie, nous allons ajouter deux options de menu Fichier, une pour démarrer une nouvelle simulation et une autre pour quitter l'application. Pour aider l'utilisateur à 'éviter de perdre le travail, les deux options offrent à l'utilisateur la chance de sauvegarder ses données de simulation en cours avant de procéder. Nous allons également ajouter une barre d'outils donnant accès rapide aux actions de l'utilisateur principal.

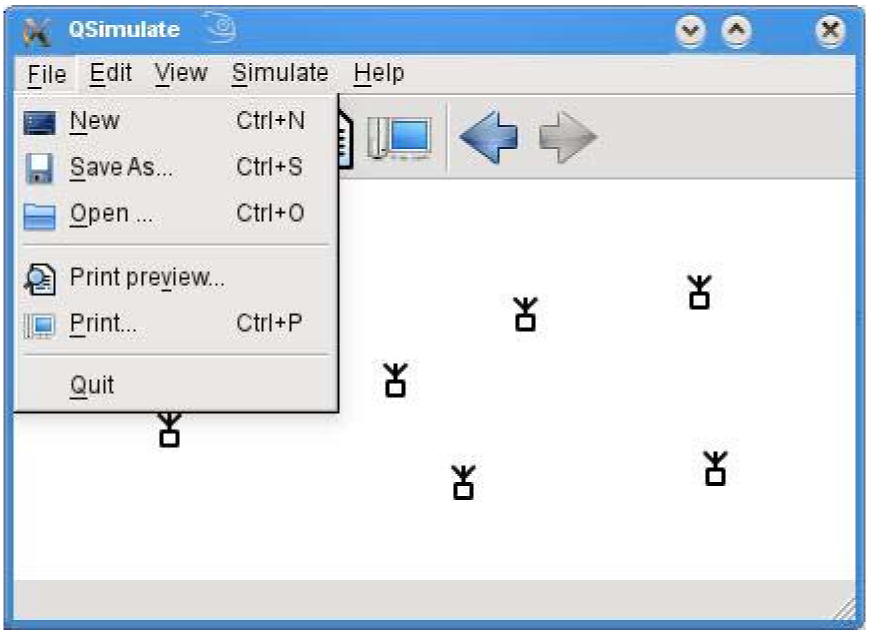
Nous avons seulement besoin de faire des améliorations à notre classe `MainWindow` à mettre en œuvre ces changements. La barre d'outils sera mise en œuvre en utilisant `QToolBar` et d'économiser des efforts au lieu de concevoir et de dessiner nos icônes propres, nous allons utiliser des `pixmaps` standard fourni par `QStyle`.

Amélioration de la `MainWindow.h`

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
class Scene;
class QUndoStack;
class QUndoView;
class QPrinter;
#include <QMainWindow>

/************************************************************/
/***** Main application window for QSimulate **************/
/************************************************************/
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  MainWindow();                    // constructor
public slots:
  void showMessage( QString ); // show message on status bar
  void showUndoStack();        // open up undo stack window
  void fileNew();              // start new simulation
  bool fileSaveAs();     // save simulation to file returning true if successful
  bool fileOpen();       // load simulation file returning true if successful
  void filePrintPreview();     // display print preview dialog
  void filePrint();            // display print dialog
  void print( QPrinter* );     // draw print page

protected:
  void closeEvent( QCloseEvent* );    // check if user really wants to exit

private:
  Scene*       m_scene;        // scene representing the simulated landscape
  QUndoStack*  m_undoStack;    // undo stack for undo & redo of commands
  QUndoView*   m_undoView;     // undo stack window to view undo & redo commands
};
#endif  // MAINWINDOW_H
```

Amélioration de la `MainWindow.cpp`

```
#include "mainwindow.h"
#include "scene.h"
#include <QMenuBar>
#include <QStatusBar>
#include <QGraphicsView>
#include <QUndoStack>
#include <QUndoView>
#include <QFileDialog>
#include <QXmlStreamWriter>
#include <QXmlStreamReader>
#include <QDateTime>
#include <QPrinter>
#include <QPrintPreviewDialog>
#include <QPrintDialog>
#include <QCloseEvent>
#include <QMessageBox>
#include <QToolBar>

/************************************************************/
/****** Main application window for QSimulate *************/
/************************************************************/
MainWindow::MainWindow() : QMainWindow()
{
  // add drop down menus
  QMenu*  fileMenu = menuBar()->addMenu( "&File" );
  QMenu*  editMenu = menuBar()->addMenu( "&Edit" );
  QMenu*  viewMenu = menuBar()->addMenu( "&View" );
  menuBar()->addMenu( "&Simulate" );
  menuBar()->addMenu( "&Help" );

  // create file menu options
  QAction* newAction  = fileMenu->addAction("&New", this, SLOT(fileNew()) );
  QAction* saveAction = fileMenu->addAction("&Save As...", this,
                                            SLOT(fileSaveAs()) );
  QAction* openAction = fileMenu->addAction("&Open ...", this,SLOT(fileOpen()));
  fileMenu->addSeparator();
  QAction* previewAction = fileMenu->addAction("Print pre&view...", this,
                                               SLOT(filePrintPreview()) );
  QAction* printAction   = fileMenu->addAction("&Print...", this,
                                               SLOT(filePrint()) );
  fileMenu->addSeparator();
  fileMenu->addAction( "&Quit", this, SLOT(close()) );
  newAction->setShortcut( QKeySequence::New );
  saveAction->setShortcut( QKeySequence::Save );
  openAction->setShortcut( QKeySequence::Open );
  printAction->setShortcut( QKeySequence::Print );

  // create undo stack and associated menu actions
  m_undoStack = new QUndoStack( this );
  m_undoView  = 0;
  viewMenu->addAction( "Undo stack", this, SLOT(showUndoStack()) );
  QAction* undoAction = m_undoStack->createUndoAction( this );
  QAction* redoAction = m_undoStack->createRedoAction( this );
  undoAction->setShortcut( QKeySequence::Undo );
  redoAction->setShortcut( QKeySequence::Redo );
  editMenu->addAction( undoAction );
  editMenu->addAction( redoAction );

  // create toolbar, set icon size, and add actions
  QToolBar* toolBar = addToolBar( "Standard" );
  QStyle* style     = this->style();
  QSize size        = style->standardIcon(QStyle::SP_DesktopIcon).actualSize(
                                                              QSize(99,99) );
  toolBar->setIconSize(size );
  newAction->setIcon(style->standardIcon(QStyle::SP_DesktopIcon) );
  openAction->setIcon(style->standardIcon(QStyle::SP_DialogOpenButton) );
  saveAction->setIcon(style->standardIcon(QStyle::SP_DialogSaveButton) );
  previewAction->setIcon(style->standardIcon(QStyle::SP_FileDialogContentsView));
  printAction->setIcon( style->standardIcon(QStyle::SP_ComputerIcon) );
  undoAction->setIcon( style->standardIcon(QStyle::SP_ArrowBack) );
  redoAction->setIcon( style->standardIcon(QStyle::SP_ArrowForward) );
  toolBar->addAction( newAction );
  toolBar->addAction( openAction );
  toolBar->addAction( saveAction );
  toolBar->addSeparator();
  toolBar->addAction( previewAction );
  toolBar->addAction( printAction );
  toolBar->addSeparator();
  toolBar->addAction( undoAction );
  toolBar->addAction( redoAction );

  // create scene and central widget view of scene
  m_scene              = new Scene( m_undoStack );
  QGraphicsView*   view = new QGraphicsView( m_scene );
  view->setAlignment( Qt::AlignLeft | Qt::AlignTop );
  view->setFrameStyle( 0 );
  setCentralWidget( view );

  // connect message signal from scene to showMessage slot
  connect( m_scene, SIGNAL(message(QString)), this, SLOT(showMessage(QString)) );

  // add status bar message
  statusBar()->showMessage("QSimulate has started");
}

/****************** showMessage *****************/
void  MainWindow::showMessage( QString msg )
{
  // display message on main window status bar
  statusBar()->showMessage( msg );
}

/**************** showUndoStack *****************/
void  MainWindow::showUndoStack()
{
  // open up undo stack window
  if ( m_undoView == 0 )
  {
    m_undoView = new QUndoView( m_undoStack );
    m_undoView->setWindowTitle( "QSimulate - Undo stack" );
    m_undoView->setAttribute( Qt::WA_QuitOnClose, false );
  }
  m_undoView->show();
}

/****************** fileSaveAs *****************/
bool  MainWindow::fileSaveAs()
{
  // get user to select filename and location
  QString filename = QFileDialog::getSaveFileName();
  if ( filename.isEmpty() ) return false;

  // open the file and check we can write to it
  QFile file( filename );
  if ( !file.open( QIODevice::WriteOnly ) )
  {
    showMessage( QString("Failed to write to '%1'").arg(filename) );
    return false;
  }

  // open an xml stream writer and write simulation data
  QXmlStreamWriter  stream( &file );
  stream.setAutoFormatting( true );
  stream.writeStartDocument();
  stream.writeStartElement( "qsimulate" );
  stream.writeAttribute( "version", "2009-05" );
  stream.writeAttribute( "user", QString(getenv("USERNAME")) );
  stream.writeAttribute( "when",
                       QDateTime::currentDateTime().toString(Qt::ISODate) );
  m_scene->writeStream( &stream );
  stream.writeEndDocument();

  // close the file and display useful message
  file.close();
  showMessage( QString("Saved to '%1'").arg(filename) );
  return true;
}

/****************** fileOpen *********************/
bool  MainWindow::fileOpen()
{
  // get user to select filename and location
  QString filename = QFileDialog::getOpenFileName();
  if ( filename.isEmpty() ) return false;

  // open the file and check we can read from it
  QFile file( filename );
  if ( !file.open( QIODevice::ReadOnly ) )
  {
    showMessage( QString("Failed to open '%1'").arg(filename) );
    return false;
  }

  // open an xml stream reader and load simulation data
  QXmlStreamReader  stream( &file );
  Scene*            newScene = new Scene( m_undoStack );
  while ( !stream.atEnd() )
  {
    stream.readNext();
    if ( stream.isStartElement() )
    {
      if ( stream.name() == "qsimulate" )
        newScene->readStream( &stream );
      else
        stream.raiseError(QString("Unrecognised element '%1'").arg(stream.name()
                                                       .toString()) );
    }
  }

  // check if error occured
  if ( stream.hasError() )
  {
    file.close();
    showMessage( QString("Failed to load '%1' (%2)").arg(filename)
                                              .arg(stream.errorString()) );
    delete newScene;
    return false;
  }

  // close file, display new scene, delete old scene, and display useful message
  file.close();
  m_undoStack->clear();
  QGraphicsView*   view = dynamic_cast<QGraphicsView*>( centralWidget() );
  view->setScene( newScene );
  delete m_scene;
  m_scene = newScene;
  showMessage( QString("Loaded '%1'").arg(filename) );
  return true;
}

/*************** filePrintPreview ************/
void  MainWindow::filePrintPreview()
{
  // display print preview dialog
  QPrinter printer( QPrinter::ScreenResolution ); // QPrinter::HighResolution );
  QPrintPreviewDialog  preview( &printer, this );
  connect( &preview, SIGNAL(paintRequested(QPrinter*)), SLOT(print(QPrinter*)) );
  preview.exec();
}

/**************** filePrint *******************/
void  MainWindow::filePrint()
{
  // display print dialog and if accepted print
  QPrinter       printer( QPrinter::ScreenResolution );
  QPrintDialog   dialog( &printer, this );
  if ( dialog.exec() == QDialog::Accepted ) print( &printer );
}

/********************* print *****************/
void  MainWindow::print( QPrinter* printer )
{
  // create painter for drawing print page
  QPainter painter( printer );
  int      w = printer->pageRect().width();
  int      h = printer->pageRect().height();
  QRect    page( 0, 0, w, h );

  // create a font appropriate to page size
  QFont    font = painter.font();
  font.setPixelSize( (w+h) / 100 );
  painter.setFont( font );
  // draw labels in corners of page
  painter.drawText( page, Qt::AlignTop    | Qt::AlignLeft, "QSimulate" );
  painter.drawText( page, Qt::AlignBottom | Qt::AlignLeft,
                    QString(getenv("USERNAME")) );
  painter.drawText( page, Qt::AlignBottom | Qt::AlignRight,
            QDateTime::currentDateTime().toString(Qt::DefaultLocaleShortDate));

  // draw simulated landscape
  page.adjust( w/20, h/20, -w/20, -h/20 );
  m_scene->render( &painter, page );
}

/******************* fileNew ******************/
void  MainWindow::fileNew()
{
  // if no stations (only default top-left scene anchor) then nothing to do
  if ( m_scene->items().count() <= 1 ) return;

  // check if user wants to save before starting new simulation
  while (true)
    switch ( QMessageBox::warning( this, "QSimulate",
        "Do you want to save before starting new?",
        QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel ) )
    {
      case QMessageBox::Save:
        // if save not successful ask again
        if ( !fileSaveAs() ) break;

      case QMessageBox::Discard:
        // start new simulation
        {
          m_undoStack->clear();
          Scene*         newScene = new Scene( m_undoStack );
          QGraphicsView* view = dynamic_cast<QGraphicsView*>( centralWidget());
          view->setScene( newScene );
          delete m_scene;
          m_scene = newScene;
        }
        return;

      default:    // "Cancel"
        return;
    }
}

/*********** closeEvent *******************/
void  MainWindow::closeEvent( QCloseEvent* event )
{ //if no stations (only default top-left scene anchor) exists
  // then accept close event
  if ( m_scene->items().count() <= 1 )
  {
    event->accept();
    return;
  }

  // check if user wants to save before quitting
  while (true)
    switch ( QMessageBox::warning( this, "QSimulate",
        "Do you want to save before you quit?",
        QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel ) )
    {
      case QMessageBox::Save:
        // if save not successful ask again
        if ( !fileSaveAs() ) break;
      case QMessageBox::Discard:
        event->accept();
        return;

      default:     // "Cancel"
        event->ignore();
        return;
    }
}
```


QSimulate
File
Edit
View
Simulate
Help
New Ctrl+N
Save As... Ctrl+S
Open ... Ctrl+O
Print preview...
Print... Ctrl+P
Quit