Eventi in Qt

Code less.
Create more.
Deploy everywhere.

Fondamenti
di informatica

Michele Tomaiuolo
tomamic@ce.unipr.it
http://www.ce.unipr.it/people/tomamic

Eventi

- In Qt, gli eventi sono oggetti, derivati dalla classe astratta `QEvent`
- Rappresentano...
  - Cambiamenti avvenuti all'interno dell'applicazione
  - O attività esterne che interessano l'applicazione
- Gli eventi sono importanti soprattutto per i widget
  - Ma possono essere emessi e/o ricevuti da qualsiasi oggetto che derivi da `QObject`

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Dispatching degli eventi

- Quando si verifica un evento, Qt crea un oggetto che rappresenta l'evento
  - Appropriata sottoclasse di `QEvent`
- Lo consegna a una certa istanza di `QObject` (o sottoclasse) chiamandone il metodo `event`
  - Il metodo `event` non gestisce in proprio l'evento
  - Sulla base del tipo di evento consegnato, chiama un metodo gestore d'evento specifico
  - L'evento può essere accettato oppure ignorato

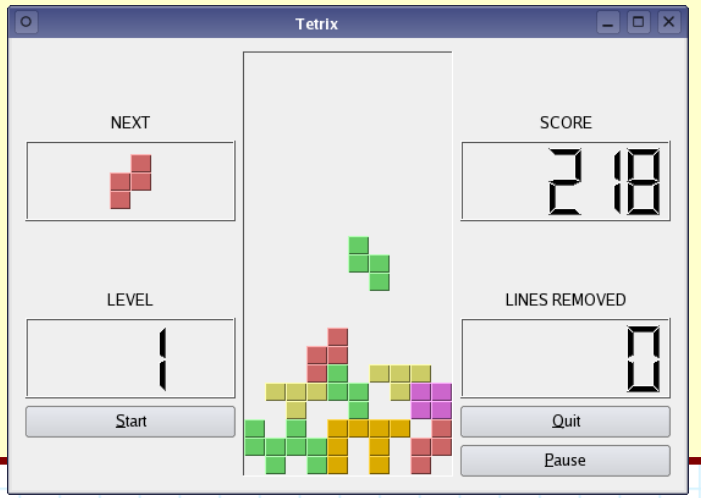
Tipi di evento

- Sottoclassi di `QEvent` per diversi eventi
  - `QResizeEvent`, `QPaintEvent`, `QMouseEvent`, `QKeyEvent`, `QCloseEvent`
- Sottoclassi aggiungono caratteristiche specifiche
  - Es. `QResizeEvent` aggiunge metodi `size` e `oldSize` per specificare dimensioni
- Alcune classi servono per più di un tipo di evento
  - `QMouseEvent` serve per pressione bottoni, doppio click, movimento ecc.
  - Tipo restituito da metodo `type`

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Eventi della tastiera

```
void TetrixBoard::keyPressEvent(QKeyEvent * e) {
  switch (e->key()) {
  case Qt::Key_Left:
    tryMove(curPiece, curX - 1, curY);
    break;
    /* ... */
  case Qt::Key_Down:
    tryMove(curPiece.rotatedRight(), curX, curY);
    break;
  default:
    QFrame::keyPressEvent(e);
  }
}
```



M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Eventi periodici

- Supporto per eventi periodici fornito direttamente da `QObject`, classe base di tutti gli oggetti Qt
  - Avvio con `startTimer`
  - Richiede intervallo (millis), restituisce *timer-id* (int)
  - `QTimerEvent` lanciato ad intervalli regolari
  - Fino a chiamata `killTimer` (richiede *timer-id*)
  - Metodo `timerEvent` per gestire evento
- Ma la API migliore per i timer è `QTimer`
  - Oggetti che emettono un segnale periodico
  - Può essere associato a uno (o più) slot

AnalogClock

```
AnalogClock::AnalogClock(QWidget *parent)
    : QWidget(parent) {
  setWindowTitle(tr("Analog Clock"));
  QTimer* timer = new QTimer(this);
  connect(timer, SIGNAL(timeout()),
    this, SLOT(update()));
  timer->start(1000);
  resize(200, 200);
}

void AnalogClock::paintEvent(
    QPaintEvent *event) {

  QPainter painter(this);
  /* ... */
}
```

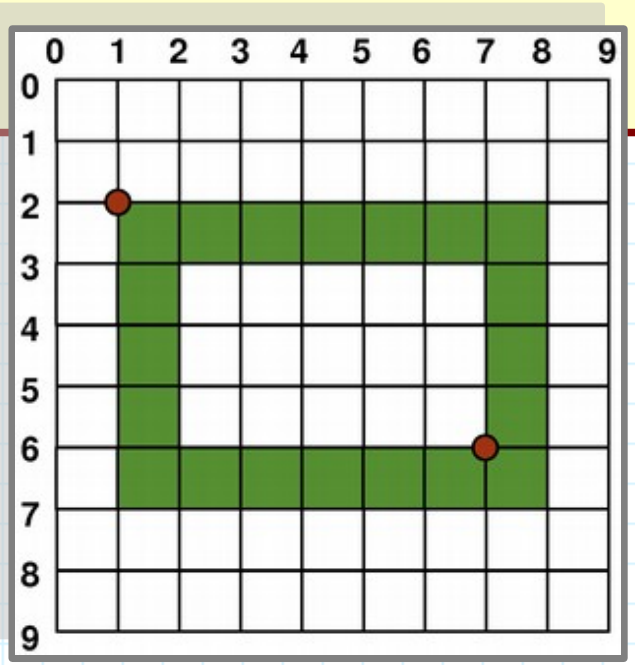
M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

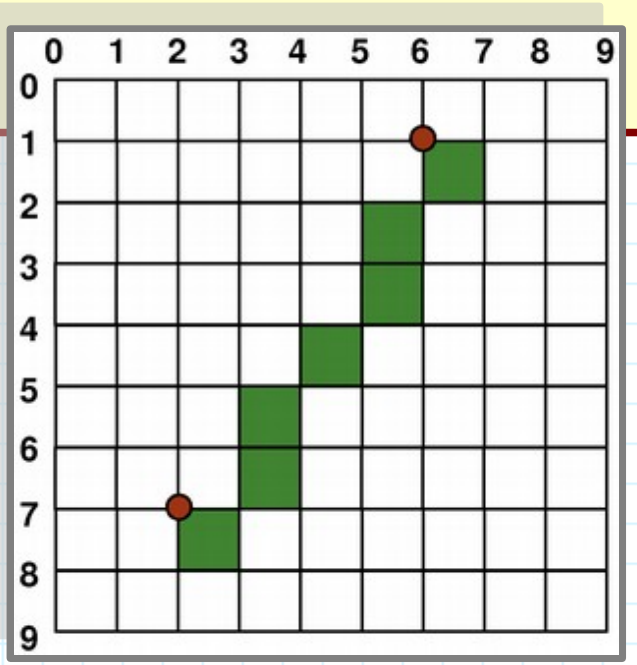
Disegni e animazioni

- Metodo `update` accoda una richiesta di ridisegno asincrono del widget (evento)
  - L'operazione non avviene immediatamente
  - L'*event dispatcher* esegue il metodo `paintEvent` appena può
- Widget Qt operano di default in double buffering
  - No *flicker* dovuto a lettura e scrittura effettuate direttamente su aree di memoria → schermo
  - Es. si traccia sfondo prima di img in primo piano
  - Se parte visualizzazione a schermo, img in primo piano scompare per un frame

Sistema di coordinate

```
/* ... */
QPainter painter(this);
painter.setPen(Qt::darkGreen);
painter.drawRect(1, 2, 6, 4);
painter.drawLine(2, 7, 6, 1);
```





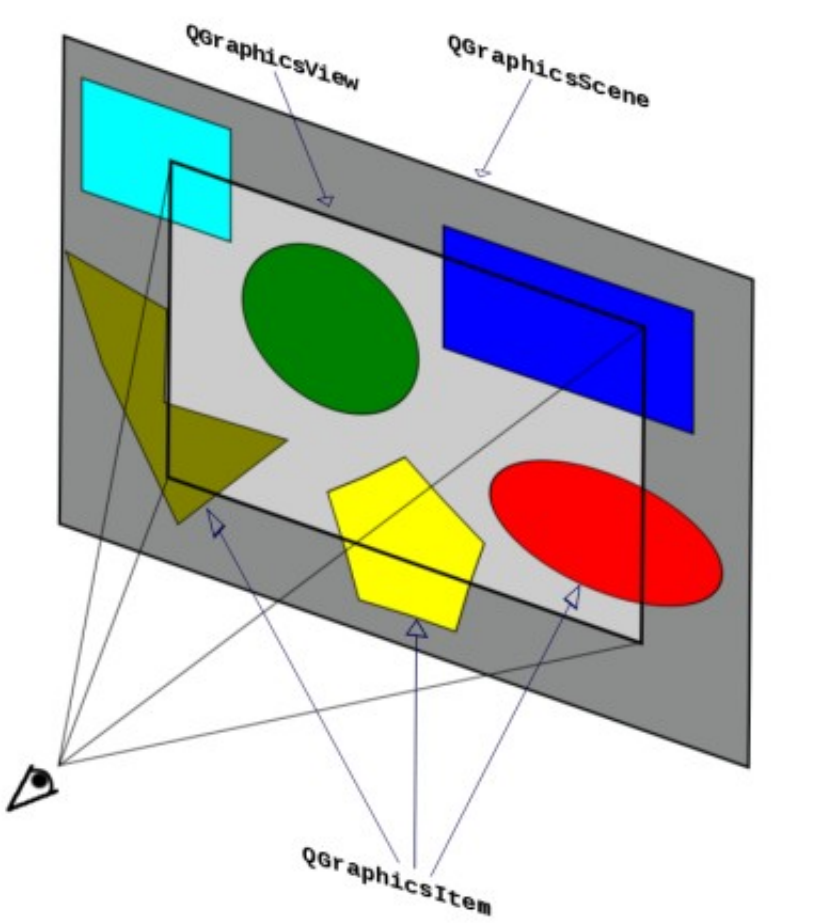
M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Disegno di immagini

- `QIcon` e `QPixmap`: classi predefinite per aggiungere immagini a etichette e bottoni, rispettivamente
  - Immagine caricata facilmente da file... passandone il nome al costruttore
- I metodi `setIcon` e `setPixmap` di etichette e bottoni impostano una immagine come loro contenuto
- Le immagini `QPixmap` possono anche essere usate come superfici di disegno custom
  - `QPainter::QPainter(QPaintDevice * d);`
  - `QPainter::drawPixmap(int x, int y, const QPixmap &pixmap);`

Superfici ed elementi grafici

- **QGraphicsScene**
  Superficie che contiene diversi elementi grafici bidimensionali: fornisce supporto per animazioni e rilevamento collisioni
- **QGraphicsItem**
  Immagini, linee, poligoni, testo e altri elementi
- **QGraphicsView**
  Widget per visualizzare l'intera scena o per zoomare su una parte



M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

QGraphicsScene

```
/* A surface to manage various 2D graphical items; it has
algorithms for animations and collision detection */
QGraphicsScene* scene = new QGraphicsScene();

/* A scene contains instances of QGraphicsItem, such as
lines, rectangles, text, or custom items */
QGraphicsItem* hello = scene->addText("Hello, world!");

/* QGraphicsView can either visualize the whole scene, or
zoom in and view only parts of the scene */
QGraphicsView* view = new QGraphicsView(scene);
view->setSceneRect(0, 0, 640, 480);

/* Use OpenGL acceleration, if available */
/* In .pro file: Qt += opengl */
view->setViewport(new QGLWidget());
view->show();
```

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

QGraphicsItemAnimation

```
QTimeLine* timer = new QTimeLine(5000);
timer->setLoopCount(0);
timer->setCurveShape(QTimeLine::LinearCurve);

QGraphicsItemAnimation* animation = new QGraphicsItemAnimation();
animation->setItem(hello);
animation->setTimeLine(timer);

animation->setPosAt(0.0, QPointF(0, 0));
animation->setRotationAt(0.0, 0);
animation->setScaleAt(0.0, 0, 0);
animation->setPosAt(0.5, QPointF(360, 360));
animation->setRotationAt(0.5, 360);
animation->setScaleAt(0.5, 4, 4);
animation->setPosAt(1.0, QPointF(0, 0));
animation->setRotationAt(1.0, 0);
animation->setScaleAt(1.0, 0, 0);

timer->start();
```

Cicli con durata 5 s.
Ripetizione all'infinito
Velocità costante

Posizione, angolo, scala
All'inizio, metà, fine animaz.
(Andata e ritorno)

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Dichiarazione di segnali

- Qt: distribuzione dei segnali gestita automaticamente
  - I segnali si dichiarano in maniera simile a metodi
  - Non devono essere implementati da nessuna parte
  - Devono restituire `void`
- Nota sugli argomenti
  - L'esperienza mostra che segnali e slot sono più riusabili se non usano tipi speciali
  - Raccogliere segnali da diversi widget sarebbe molto più difficile

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Emissione di segnali

- Un oggetto emette un segnale chiamando `emit`
  - Quando si verifica evento o stato interno cambia
  - Se il cambiamento può interessare altri oggetti
- Emesso segnale → slot connessi eseguiti subito
  - Come una normale chiamata a metodo
  - Codice seguente ad `emit` eseguito dopo aver eseguito tutti gli slot connessi al segnale
  - Se più slot, eseguiti in sequenza arbitraria
- Esistono anche connessioni asincrone (*queued*)
  - Codice dopo `emit` eseguito subito, poi gli slot

Slot

- Uno slot è chiamato quando viene emesso un segnale ad esso connesso
- Gli slot sono normali metodi C++ che possono essere invocati normalmente
- La loro unica caratteristica speciale è che possono essere connessi a dei segnali

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Meta-Object Compiler

- Il *moc* è un programma che gestisce le estensioni di Qt al C++
- Se una dichiarazione di classe contiene la macro `Q_OBJECT`, il *moc* produce altro codice C++ per quella classe
- Tra le altre cose, il "*meta-object code*" è necessario per il meccanismo di segnali e slot

Segnali e slot sono una estensione

specifica di Qt alla sintassi C++

Bottone con click destro

```
class RightPushButton : public QPushButton {
  Q_OBJECT
public:
  RightPushButton(QWidget* parent = NULL);
protected:
  void mouseReleaseEvent(QMouseEvent* e);
signals:
  void rightClicked();
};
```

Metodo ridefinito (overridden)

Nuovo segnale, oltre a `clicked` di `QPushButton`

```
RightPushButton::RightPushButton(QWidget* parent)
    : QPushButton(parent) { /* empty */ }

void RightPushButton::mouseReleaseEvent(QMouseEvent* e) {
  if (e->button() == Qt::RightButton)
      emit rightClicked();
  QPushButton::mouseReleaseEvent(e);
}
```

Alla fine, eseguito il codice della superclasse

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione
http://www.ce.unipr.it/people/tomaiuolo

Gruppo di bottoni con click destro

```
class RightButtonGroup : public QButtonGroup {
    Q_OBJECT

public:
  RightButtonGroup(QObject* parent = NULL);
  void addButton(QAbstractButton* button, int id);

signals:
  void buttonRightClicked(int id);
  void buttonRightClicked(QAbstractButton* button);

private slots:
  void emitRightClicked();
};
```

Metodo ridefinito (overridden)

Nuovi segnali, oltre ai due `buttonClicked` di `QButtonGroup`

Gruppo di bottoni con click destro

```
void RightButtonGroup::addButton(
    QAbstractButton* button, int id) {
  RightPushButton* rb =
      dynamic_cast<RightPushButton*>(button);
  if (rb != NULL) {
    connect(rb, SIGNAL(rightClicked()),
        this, SLOT(emitRightClicked()));
  }
  QButtonGroup::addButton(button, id);
}

void RightButtonGroup::emitRightClicked() {
  QAbstractButton* button =
      dynamic_cast<QAbstractButton*>(sender());
  emit buttonRightClicked(button);
  emit buttonRightClicked(id(button));
}
```

Downcasting: conversione da super- a sottoclasse
`NULL` se errore tipo

Alla fine, eseguito il codice della superclasse

Metodo di `QObject`
Restituisce l'oggetto che ha emesso il segnale

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

FifteenModel – Segnale dal modello

```
class FifteenModel
        : public QObject, public FifteenPuzzle
{
    Q_OBJECT
public:
    FifteenModel(int rows,
                 int columns);
    void shuffle() override;
    Coord getBlank();
    Coord getMoved();
signals:
    void blankMoved();
protected:
    void moveBlank(Coord delta) override;
    bool silent;
};
```

Macro `Q_OBJECT` + derivazione da `QObject` → gestione segnali/slot

Nuovo segnale: mossa compiuta (spostata la casella vuota)

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

FifteenModel – Lancio segnale

```
void FifteenModel::shuffle()
{
    silent = true;
    FifteenPuzzle::shuffle();
    silent = false;
}

void FifteenModel::moveBlank(Coord delta)
{
    FifteenPuzzle::moveBlank(delta);
  // while shuffling, no signals are emitted
    if (!silent) emit blankMoved();
}
```

Emesso segnale dopo ogni mossa (spostamento della casella vuota)

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

FifteenGui – Nuovo slot

```
class FifteenGui : public QWidget {
  Q_OBJECT
public:
  FifteenGui(FifteenModel* model, QWidget* parent);
private slots:
  void controlButtons(int i);
  void updateAfterMove();
private:
  /* … */
};

FifteenGui::FifteenGui(/* … */) {
  /* … */
  QObject::connect(
    model, SIGNAL(blankMoved()),
    this, SLOT(updateAfterMove()));
}
```

(.h) Nuovo slot per aggiornare i bottoni dopo una mossa sul modello

(.cpp) Intercettato il segnale di nuova mossa emesso dal modello

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

FifteenGui – Gestione segnali

```
void FifteenGui::controlButtons(int i) {
  int y = i / model->getWidth();
  int x = i % model->getWidth();
  model->move({x, y});
  // updateAllButtons(); <- not needed anymore
}

void FifteenGui::updateAfterMove() {
  FifteenModel::Coord moved = model->getMoved();
  buttons->button(index(moved))->setText(
    Qstring(model->get(moved)));

  FifteenModel::Coord = model->getBlank();
  buttons->button(index(blank))->setText(
    Qstring(model->get(blank)));

  checkSolution();
}
```

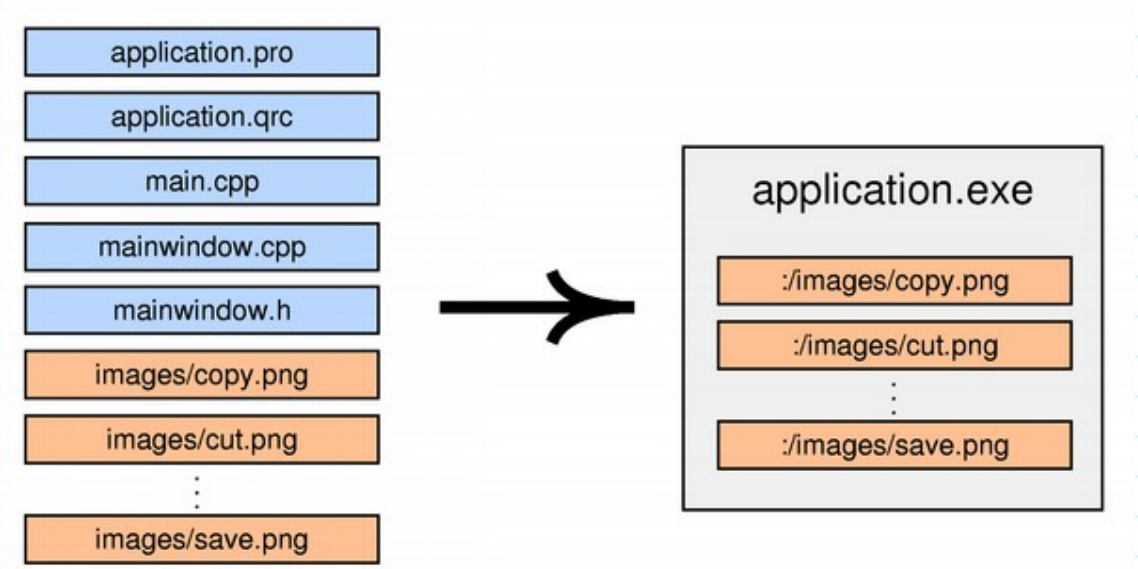
Non è più necessario aggiornare completamente la griglia dei bottoni

Aggiornati solo i due bottoni effettivamente modificati con la mossa

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Altre caratteristiche di Qt

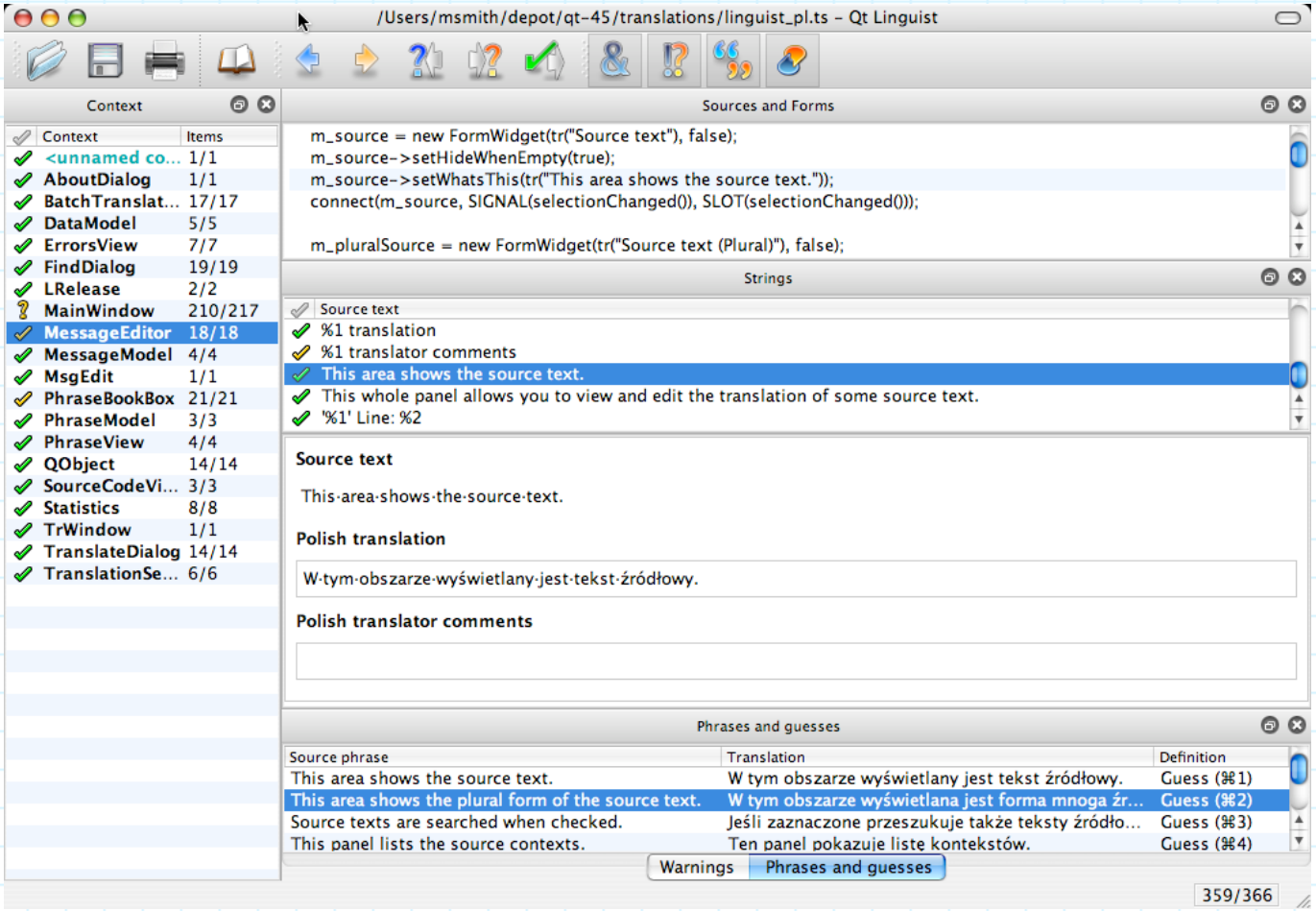
- Possibile inserire dati direttamente nel file eseguibile
  - Es. Immagini, suoni e altri dati
  - Aggiungere un "*Qt Resource File*" al progetto
  - Aggiungere le singole risorse al descrittore `.qrc`
  - Percorso delle risorse richiede ":" come prefisso



M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Qt Linguist

- http://doc.qt.digia.com/qt/linguist-hellotr.html



M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

Utilizzare le traduzioni

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

  // run lupdate(.pro->.ts), linguist and
  // lrelease(.ts->.qm), first!
  QTranslator appTranslator;
  appTranslator.load(":/translations/myapp_"
      + QLocale::system().name());
  a.installTranslator(&appTranslator);

  QTranslator qtTranslator;
  qtTranslator.load("qt_" + QLocale::system().name(),
      QLibraryInfo::location(QLibraryInfo::TranslationsPath));
  a.installTranslator(&qtTranslator);

  return a.exec();
}
```

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/

QString

- Caratteri a 16 bit (UTF-16, rari simboli a due QChar)
  - `QChar.isHighSurrogate / isLowSurrogate`
- Metodi `fromStdString` e `toStdString`
- Metodi `setNum` e `toInt`, `toFloat`
- Metodo `split` (genera una `QStringList`)
- Sostituzione automatica di numeri, char e stringhe
  - ```
    QString status =
    QString("Processing file %1 of %2: %3")
    .arg(i).arg(total).arg(fileName);
    ```
- Operatori `[], >, <, ==, +, +=` ecc.

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – UniPR
http://www.ce.unipr.it/people/tomamic/
    ```