GUI Summer 2020

06/02/2020

What is Usability (Definition)

- Usability of what? - computer program in performing a task
- Usability - “the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.” (IEEE 1990)

Usability metrics

- Set of features
  - Operability – Ease of task completion
  - Learnability – Learning curve
  - Understandability
  - Attractiveness (subjective)
  - Effectiveness – in enabling specific Task
  - Efficiency- resources required (e.g., Time on Task ToT) Number of clicks
  - Satisfaction (subjective)
  - Accessibility
  - ~~Compliance Response time SE principles (functional), portability~~

How do you measure Usability?

- Effort Based
  - TOT
  - Number of clicks
  - Mouth path traversed
  - Eye path traversed
- Error rate (operator)
- Questionnaires, polls, observations, focus groups

How do you set Usability requirements?

- Contract base requirements Non-functional

User interface development tools

- QT (was used by Nokia)
- Web tools

Components of an XR / Gaming system

- Graphics Engine
- AI Engine (enhancing player/user experience)
- Physics engine

Modes of interaction

- M2M Interaction
  - The input output is more precise and predictable
    - Sequence of event
    - Throughput ; Latency
  - One to one and one to many
  - Synchronous & and asynchronous (fixed freq. vs Variable frequency)
- M2H Interaction (UI)
  - One to one and one to many
  - Error pruned
  - Asynchronous (variable frequency)

Modes of interaction

- M2M Interaction (serve an I/O device)
  - Manage I/O (synchronous)
  - Polling – requires full attention
  - Interrupt – serve as needed (overhead)
  - Priorities – order of polling; priority encoder
  - Managing tasks (Unix) (serve a task) Signal (like an interrupt)
- M2M Interaction (UI) (asynchronous)
  - Most likely Interrupt
  - Event driven

Modes of interaction (M2H)

- M2H Interaction (UI)
  - Asynchronous (variable frequency)
  - Event driven
  - OS managed event queue (potentially a second Q in the UI app)
  - What happens on an event?
    - Signal/interrupt is generated
    - OS places in the Q
    - UI I/O app fetches from the Q and serves

Event-driven Programming

- Event-driven programming is the standard approach to creating graphical user interfaces (GUIs)
  - An event-driven program is object-oriented
    - Object-oriented programming was originally development to implement graphical objects within GUI operating systems
    - Although object-oriented, flow-of-control is sufficiently different that we are going to classify it as a different paradigm that extends OOP.
  - However, top-level control is expressed differently
    - The user is the top-level loop
      - Think of Word, or a game program
    - Every action in your program is a reaction to the user
      - Decompose program in terms of "what will I do if the user does…"
      - User inaction may trigger background actions (e.g. games)

Handling Events

- Call back function
  - Function that is invoked on an event
  - The programmer provides the function code for anticipated events
- QT Signals and Slots
  - In QT devices and widgets (widgets) can generate signals
  - Signals can be associated with slots (associated with a methods)
- Assume 2 sliders
  - Slider1 can send a signal to slider 2 (same with 2)
  - Each slider might have a slot (method) slot1 (slider 1) slot 2(slider 2)

Handling Events

- Assume 2 sliders
  - Slider1 can send a signal (1) to slider 2 (same with 2)
  - Each slider might have a slot (method) slot1 (slider 1) slot 2 (slider 2)
  - Next, we connect the signal of slider 1 to the slot of slidre2

QT Demos

- /usr/lib/qt4/demos
- Qt4.7 Demos https://doc.qt.io/archives/qt-4.7/demos.html
- QT 4.5 Tutorial https://docs.huihoo.com/qt/4.5/tutorials.html
- The QT Free book (TRACS)

Event-driven Programming

- Event-driven programming is the standard approach to creating graphical user interfaces (GUIs)
  - An event-driven program is object-oriented
    - Object-oriented programming was originally development to implement graphical objects within GUI operating systems
    - Although object-oriented, flow-of-control is sufficiently different that we are going to classify it as a different paradigm that extends OOP.
  - However, top-level control is expressed differently
    - The user is the top-level loop
      - Think of Word, or a game program
    - Every action in your program is a reaction to the user
      - Decompose program in terms of "what will I do if the user does…"
      - User inaction may trigger background actions (e.g. games)

Detecting Asynchronous Events

- Polling
  - Repeatedly read input devices in an infinite loop
- Interrupt-driven
  - Hardware-triggered context-switching in response to user-events
- Event-driven
  - Explicit event waiting
  - Invokes functions in response to user events
  - Note that function invocations are *asynchronous*

Qt uses event-driven processing.

Method #1 Polling

Interaction is governed by a simple loop:

```
Loop forever:
{
    read input
    respond to input
}
```

What limitations does this have?

Does it have any advantages?

Method #2 Interrupt-driven Processing

1. **Enable device, then**
2. **proceed with “background” processing until an interrupt is received, at which point**
3. **Save state (context-switch)**
4. **Respond to input**
5. **Restore state and go to step #2.**

What advantages/disadvantages does this have?

Method #3: Event-driven Processing

Interaction is once again governed by a loop:

```
Loop forever:
{
    if (event) then
        respond
    else
        do (one unit of) background
            processing or
            go to sleep (for one unit)
}
```

Events / Messages / Signals

- **Any event-driven graphics package has devices that can signal events**
  - **In old standards, this was limited to hardware devices**
  - **In newer packages (e.g. Qt), any widget can signal events; the (hardware) mouse is the same as a (software) slider or button.**
- **Generally, the event tells you**
  - **Which device/widget signaled the event**
  - **Some "measure" giving the new state**
    - **E.g., whether a mouse button was depressed or released**

Warning: old systems tend to use the term "events"

while newer systems may call them messages or signals

Event handlers

- Event handlers are functions invoked in response to an event.
- Predominant approach is using “call-back” functions.
  - Functions are registered with the event
  - Signature of function most conform with language standard for the event

Call-back function: Java

```
btnJava.addActionListener(this);
btnWishJava.addActionListener(this);
btnNoJava.addActionListener(this);

public void actionPerformed(ActionEvent e) {
Object source = e.getSource();
lblInstructionLabel.setText("Got it!");
if (btnJava.equals(source))
 lblFeedback.setText("Yes");
else if (btnWishJava.equals(source))
  lblFeedback.setText("No");
else if (btnNoJava.equals(source))
  lblFeedback.setText("Maybe so");
}//end action performed
```

simple.c (OpenGL Gl/Glu/Glut)

```
#include <GL/glut.h>
void mydisplay(){
     glClear(GL_COLOR_BUFFER_BIT);
      glBegin(GL_POLYGON);
            glVertex2f(-0.5, -0.5);
            glVertex2f(-0.5, 0.5);
            glVertex2f(0.5, 0.5);
            glVertex2f(0.5, -0.5);
      glEnd();
      glFlush();
      glu(…)
}
int main(int argc, char** argv){
      glutCreateWindow("simple");
      glutDisplayFunc(mydisplay);
      glutMainLoop();
}
```

main.c

```
#include <GL/glut.h>

int main(int argc, char** argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("simple");
  glutDisplayFunc(mydisplay);

  init();

  glutMainLoop();
}
```

includes **gl.h**

define window properties

display callback

set OpenGL state

enter event loop

Widgets

- Base class for all UI widgets
- Properties
  - width, height, backgroundColor, font, mouseTracking, backgroundPixmap, etc.
- Slots
  - repaint, show, hide, move, setGeometry, setMainWidget, etc.
- Signals:
  - mouseMoveEvent, keyPressEvent, resizeEvent, paintEvent, enterEvent, leaveEvent, etc.

Events

- Signals: emit events
  - declare as signals, otherwise normal member functions
  - You don't implement them. Rather, you send them with the (new) keyword emit
  - E.g. emit(sliderChanged(5))
- Slots: receive and handle events
  - Normal member functions declared as slots
- Connect: must connect signals to slots
  - QObject::connect(mymenu, SIGNAL(activated(int)), myobject, SLOT(slotDoMenuFunction(int)));
- moc: meta object compiler (preprocessor) converts these new keywords to real C++

Qt, a GUI toolkit

- Events processed with *signals* and *slots*
  - signal generates an event, e.g., button push
  - slot processes the event, e.g., pop up a file dialog box

```
QPushButton * quitB = new QPushButton("Quit",...,...);
```

- connect (quitB, SIGNAL(clicked()), qApp, SLOT(quit());
  - qApp is a global variable, of type *QApplication*
    - one QApplication per program defined first in main()
  - main returns qApp.exec()
  - SIGNAL and SLOT are macros, expanded by a meta-object
- compiler (moc)
  - moc generates .cpp files from user-defined Qt subclasses

Event Handling

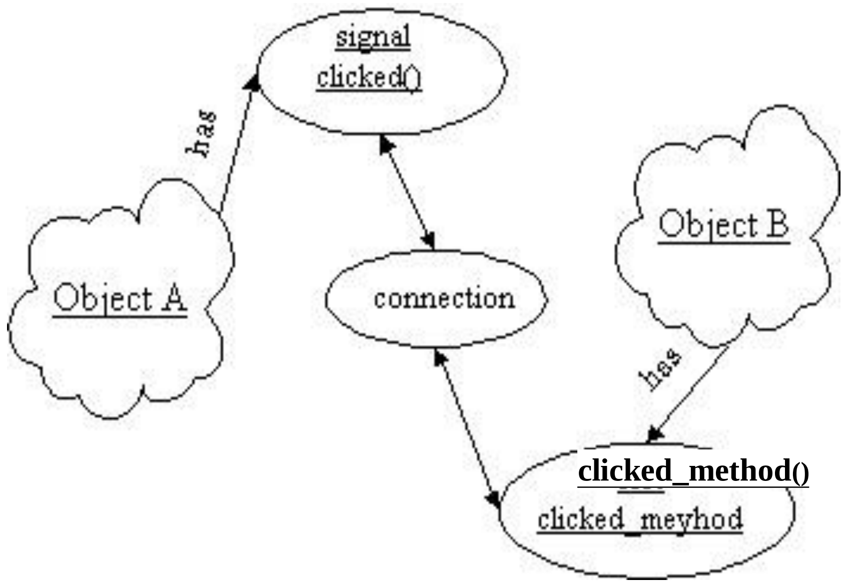
- QT's new approach: signals and slots
  - A widget sends out various *signals*
  - Object methods can be declared as *slots*
  - Compatible signals and slots can be *connected* or plugged together like a telephone switchboard (parameter types must match)
- Strict separation
  - This strict separation between UI components and program elements lends itself to component-based programming
  - Goal: separate UI from program logic

Signals and Slots

- A widget emits a signal
  - widget = windows gadget
- The widget has no knowledge of who will respond to the signal
  - or even if anyone will
- A signal can be connected to a slot
  - slot code is executed after a signal is emitted

Signals and Slots

- All that is required to connect signals and slots is that the signatures match.
- Developers may use predefined or user-defined member functions as signals and slots.



Signals and Slots (cont)

- advantage:
  - independent interface
  - type-safe
  - process transparence
- disadvantage:
  - not as fast as a direct function pointer call.

    (A signal triggering a slot has been measured to approximately 50 microseconds on a SPARC2.)

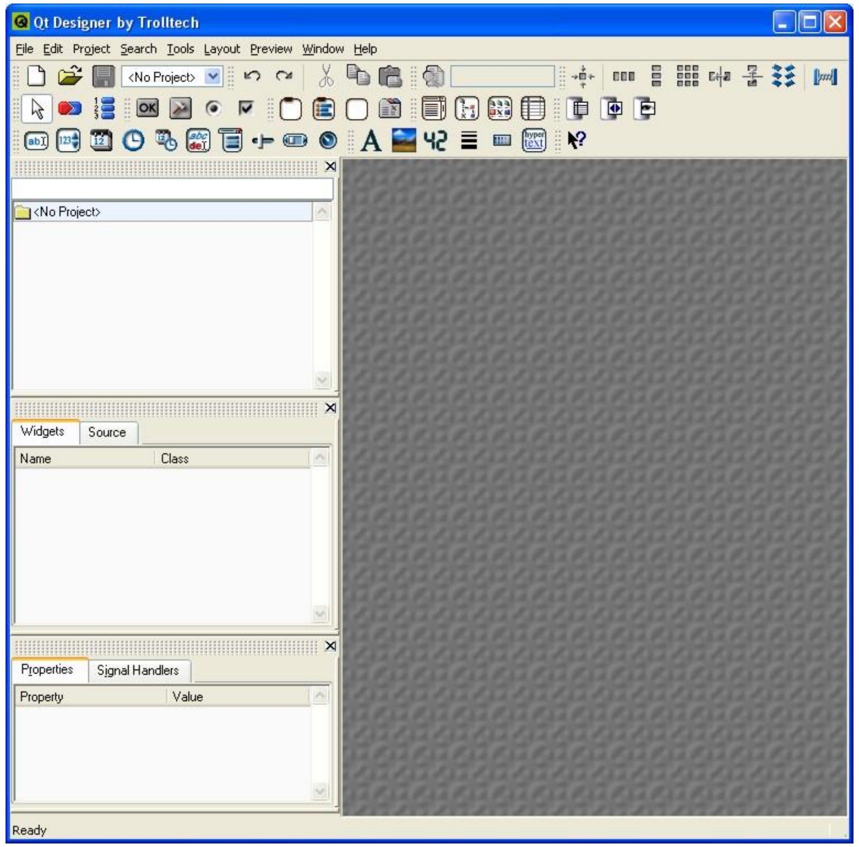
Defining Signals and Slots

- New C++ syntax for defining signals and slots, added to public, private, etc.

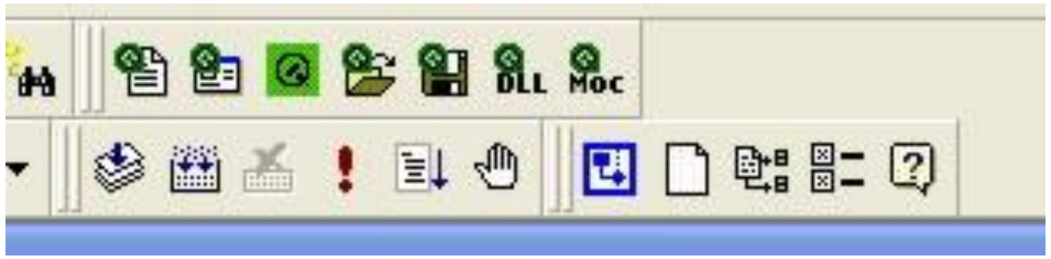
```
class myClass : public Qobject {
Q_OBJECT         //required macro, no semicolon
…
signals:
  void somethingHappened();
…
public slots:
  void slotDoSomething();
…
private slots:
  void slotDoSomethingInternal();
…
};
```

Qt Designer

- Qt provides a GUI builder called Designer
- Designer simplifies widget placement and can even provide simple behavior
- Qt Designer will be covered later in the semester



Qt Plug-in for Visual Studio



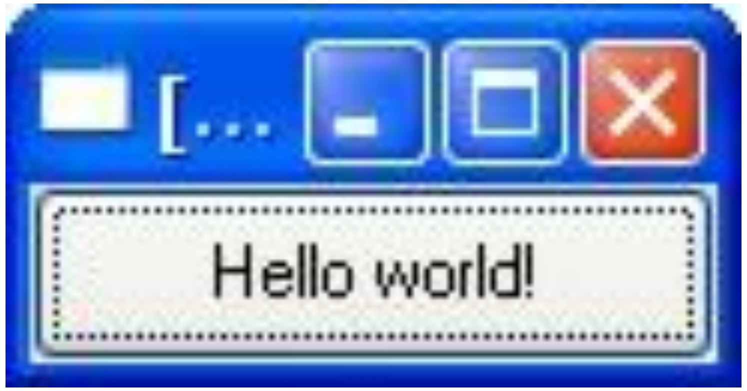
- Qt provides a plug-in for use with Microsoft Visual Studio 6
    - Does not work with Visual Studio .NET
- The plug-in can create a default Qt project and provides shortcuts to Qt tools

Achieving Portability with qmake

- Different platforms use different compilers
- Different compilers require different compilations flags/syntax
- To make it easier to port Qt applications, the qmake utility was created
    - tmake used to be a separate tool that served this purpose in Qt versions prior to 3.0

A Sample Project File

```
TEMPLATE        = app
CONFIG          += qt warn_on release
HEADERS         =
SOURCES         = main.cpp
TARGET          = t1
REQUIRES=small-config
```



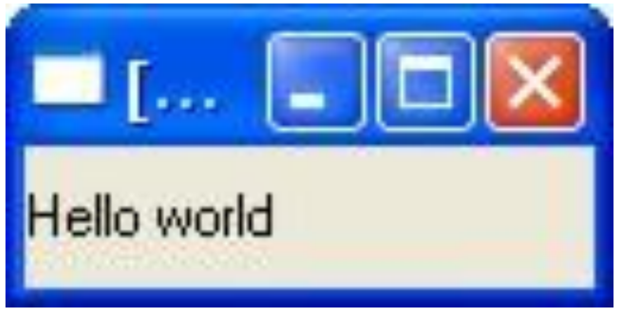
qmake

- The qmake utility is typically invoked with the following three commands]

```
qmake –project
qmake
make (or nmake under Windows)
```

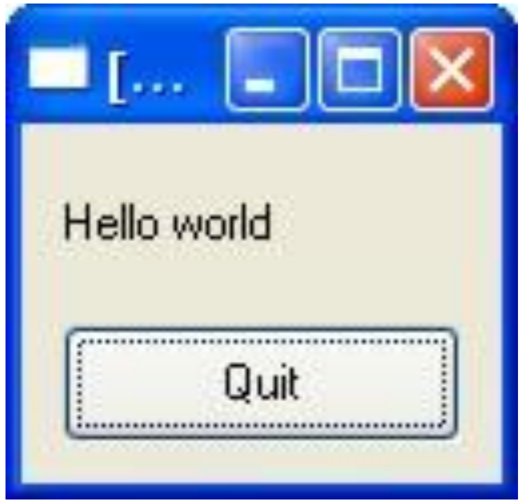
- Rules:
  - Be sure to place code in its own directory.
  - qmake scans all subdirectories for dependencies. Do not place archive version under a “save” subdirectory.
  - If you reorganize your files, like adding a new .h, delete all the .pro and other working files, then start over.

Hello, world!



```
#include <qapplication.h>
#include <qlabel.h>

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

  QLabel* mylabel = new QLabel( “Hello, world”, 0);
  mylabel->resize( 120, 30 );

  myapp.setMainWidget( mylabel );
  mylabel->show();
  return myapp.exec();
}
```

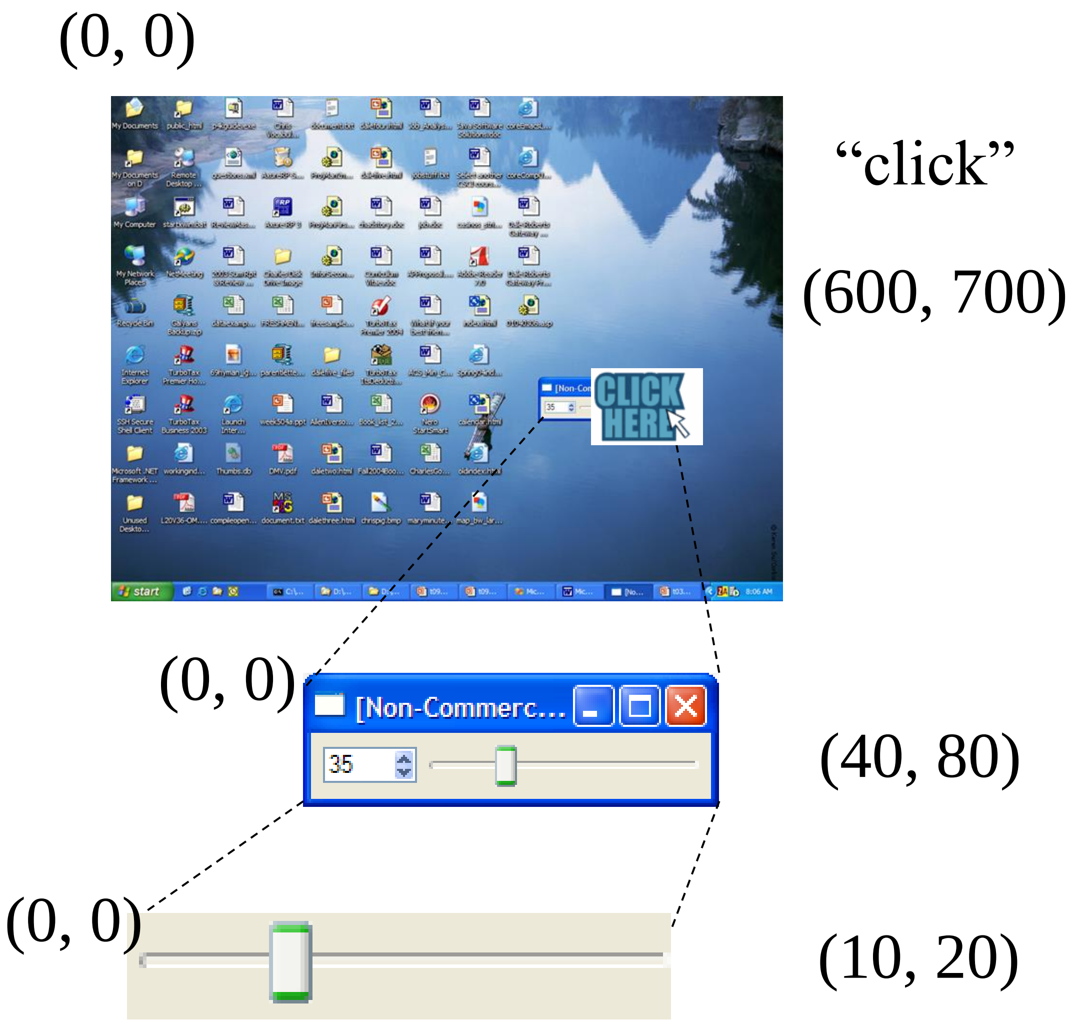
Push Button



```
#include <qapplication.h>
#include <qlabel.h>
#include <qpushbutton.h>

int main( int argc, char* argv[] )
{
  QApplication myapp( argc, argv );
  QWidget* mywidget = new QWidget;
  mywidget->setGeometry( 400, 300, 120, 90 );
  QLabel* mylabel = new QLabel( "Hello world", mywidget );
  mylabel->setGeometry( 10, 10, 80, 30 );
  QPushButton* myquitbutton = new QPushButton( "Quit", mywidget );
  myquitbutton->setGeometry( 10, 50, 100, 30 );
  QObject::connect( myquitbutton, SIGNAL(clicked()), &myapp,SLOT(quit()) );
  myapp.setMainWidget( mywidget );
  mywidget->show();
  return myapp.exec();
}
```

Pick Correlation

- Events are originally detected by the operating system, and then routed to application through "Pick Correlation". The application may continue to your the event to one of its children widgets.
- The principle is to encapsulate the event by routing it to the lowest level object.



Coordinates are relative to (0,0) in the upper-left corner of desktop/windows/widget..

Signals and Slots(cont)

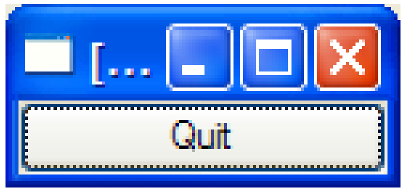
```
class PixmapRotator : public QWidget {
    Q_OBJECT
    public:
        PixmapRotator(QWidget *parent=0, const char *name=0);
    public slots:
        void setAngle(int degrees);
    signals:
        void angleChanged(int);
    private:
        int ang;
};
void PixmapRotator::setAngle( int degrees ) {
    degrees = degrees % 360;          // keep in range <-360, 360>
    if(ang == degrees)
        return;                       // actual state change?
        ang = degrees;                // a new angle
        emit angleChanged(ang);       // tell world ...
}
QObject::connect(scrollBar, SIGNAL(valueChanged(int)), rotator,
                                                   SLOT(setAngle(int)));
```

Making Connections in Qt

```
 #include <qapplication.h>
#include <qpushbutton.h>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QPushButton *button = new QPushButton("Quit", 0);
    QObject::connect(button, SIGNAL(clicked()),
                     &app, SLOT(quit()));
    app.setMainWidget(button);
    button->show();
    return app.exec();
}
```

- **Note that the connect function is a static function of QOBJECT.** *How do you know?*

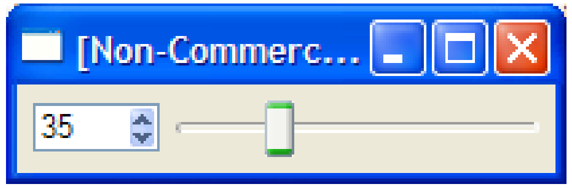
quit.exe



Making Connections with Qt

```
1.  #include <qapplication.h>
2.  #include <qhbox.h>
3.  #include <qslider.h>
4.  #include <qspinbox.h>

5.  int main(int argc, char *argv[])
6.  {
7.     QApplication app(argc, argv);

8.     QHBox *hbox = new QHBox(0);
9.     hbox->setCaption("Enter Your Age");
10.    hbox->setMargin(6);
11.    hbox->setSpacing(6);

12.    QSpinBox *spinBox = new QSpinBox(hbox);
13.    QSlider *slider = new QSlider(Qt::Horizontal, hbox);
14.    spinBox->setRange(0, 130);
15.    slider->setRange(0, 130);

16.    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
17.                slider, SLOT(setValue(int)));
18.    QObject::connect(slider, SIGNAL(valueChanged(int)),
19.                spinBox, SLOT(setValue(int)));
20.    spinBox->setValue(35);

21.    app.setMainWidget(hbox);
22.    hbox->show();

23.    return app.exec();
24. }
```

•Lines 1 – 4: You must include header files for each Qt object referenced in this file.
•Line 5 - 7: Declaring argc and argv and passing it through to the QApplication constructor is standard procedure. This allows all Qt programs to specificy the presentation style on the command line. Examples are age –style=windows, age –style=motif, and age –style=cde.
•Lines 8 – 11: Declare a *Layout Manager*. QHBox automatically arranged controls horizontally from left to right. Layout managers are used to alleviate the need for programmers to calculate actual coordinates for each control in the window. setCaption set the Title Bar. setMargin controls spacing around and in between child widgets.
•Lines 12 – 13: Create a spin box and slider with QHBox as the parent.
•Lines 14 – 15: Set the valid range for the spin box and the slider.
•Lines 16 – 19: Ensure that the spin box and slider are sychronized. Changing one always changes the other.
•Line 20 – Sets the initial value of the spin box to 35. Note that this causes the spin box to emit a valueChanged(int) signal, which is connected to slider's setValue(int) slot. The slider will emit a valueChanged(int) signal, which is connected to the spin box's setValue(int) slot. Since the value is already 35, the spin box does not emit another valueChanged(int) signal and avoids and infinite recursion loop.
Lines 21 – 23: Sets the main widget to the hbox, makes the hbox visible, and starts the window's main event loop.



age.exe

Other Features of Qt

- The Qt Paint Engine
- **QPainter** is highly optimized and contains several caching mechanisms to speed up drawing. Under X11, it caches GCs (graphics contexts), which often make it faster than native X11 programs.
- **QPainter** contains all the functionality one would expect from a professional 2D graphics library. The coordinate system of a QPainter can be transformed using the standard 2D transformations (translate, scale, rotate and shear).
- **Qt** supports Open GL for 3D graphics.
- Qt also contains a set of general purpose classes and a number of collection-classes to ease the development of multi-platform applications.
- Qt has platform independent support for the operating system dependent functions, such as time/date, files/directories and TCP/IP sockets.

Simulation Framework (e.g., Decision Support System)

- Time driven (similar to polling)
- Event driven

Time Driven Simulation

```
Assume B is static A tries to close the distance from B
Set an atomic time unit (tic)
Set initial system status
B is in location (x,y) (A is location (u, v))
Loop on tics (polling)
        Check system status
                Status might change based on some random parameters
        Make a decision
        Update system Status
        Check for exceptions
        Collect data
End loop
Analyze data
```

```
Assume A is static B tries to close the distance from B
Set an atomic time unit (tic)
Set initial system status
B is in location (x, y) (A is location (u, v))
Loop on tics (polling)
        Check system status
                distance from A (Status might change based on some random parameters)
        Make a decision
                choose the direction that minimizes distance
        Update system Status
                Set B coordinate to the selected neighbor
        Check for exceptions
        Collect data
End loop
Analyze data
```

Chasing

(x-1,y+1) – NE	(x,y+)	
(x-1, y)	(x,y) Static	
(x-1, y)		

3	2	1
4	8	0
5	6	7

Event Driven Simulation

```
Set an atomic time unit (tic)
Set initial system status
Loop on tics (polling)
        Check system status
                Status might change based on some random parameters
        Make a decision
        Update system Status
        Check for exceptions
        Collect data
End loop
Analyze data
```

OpenGl (CG)

- Goal generate images that look like “natural” images
    - Natural image is an image obtained by a film camera
- Alternatively drawing and visualization

Steps:

1) Chose and place the camera Define projection, define L R B T NF Place the camera
2) Define a “Model” scene
    1) Use a library of objects (mesh)
    2) Or, Get a model from other resources (mesh)
    3) Place objects in the scene
    4) Enable object manipulation (scale, translate, rotate, or, add objects from library)
        1) UI of player with the game or done by the AI engine
        2) Enforce “physics”
3) Set ligting and lighting properties
4) Set material properties (for interaction with light or other objects)
5) Enable rendering along with animation.