### User Interface Programming in C#:
#### Basics and Events

**Chris North**

**CS 3724: HCI**

---

### GUI Development: Goals

1. General GUI programming concepts
   - GUI components, layouts
   - Event-based programming
   - Graphics
   - Direct Manipulation, Animation
   - MVC architectures
   - Data-driven UIs

2. C#, .NET
   - Windows Forms
   - Events, delegates
   - GDI+
   - Threads
   - ADO.net

Goal: learn other languages quickly, same concepts
- VB, Xwin, Java 49, ...

---

### C# Background

- C# = VB + Java (best of both!)
- Basic statements identical to C++, Java
- Object-oriented only!
  - main() is inside a class
  - No global variables
- "interfaces"
- No pointers (object references only), safe
- No delete: automatic garbage collection
- Error Handling, exceptions (try, catch)
- GUI: Windows Forms
- Libraries: Data structs, databases, ...
- Component-based: ("assembly") reflection
  - No .h files
- Visual Studio
- .NET: CLR, multi-language, web, XML, services, ...

### C# Materials

- MSDN (integrates with VS)
- VS Dynamic Help
- Books
  - MS Visual C# .NET, MS Press
    - C# language
    - Windows Forms, GDI+
- MSDN online
### GUI Topics
- Components
- Events
- Graphics
- Manipulation
- Animation
- MVC

### Components API
- **Properties**
  - Like member fields
  - Get, set
  - E.g. Button1.Text = “Press Me”
- **Methods**
  - Like member functions
  - Tell component to do something
  - E.g. Button1.Show()
- **Events**
  - Like callback functions
  - Receive notifications from component
  - E.g. Button1.Click(e)
Interactive command line program

- User input commands
- Non-linear execution
- Unpredictable order
- Much idle time

```c
program:
main()
{
  decl data storage;
  initialization code;
  loop
    get command;
    switch(command)
    {
      command1:
        code;
      command2:
        code;
      ...
    }
}
```

Typical GUI program

- User input commands
- Non-linear execution
- Unpredictable order
- Much idle time
- Event callback proc

```c
GUI program:
main()
{
  decl data storage;
  initialization code;
  create GUI;
  register callbacks;
  main event loop;
  callback1() //button1 press
    {
      code;
    }
  callback2() //button2 press
    {
      code;
    }
  ...
```

GUI Events

- Mouse click
- Window system
- Input device

C# WinApp

- "delegates" = callbacks
- Function pointers

```c
C# WinApp:
Class
{
  decl data storage;
  constructor()
  {
    initialization code;
    create GUI controls;
    register callbacks;
  }
  main()
  {
    Run(new
    }
  callback1()
    {
      do stuff;
    }
  callback2()
    {
      do different stuff;
    }
  ...
```

Java: Listeners

```java
App1
OK
Cancel

App2
OK btn_click()
  {
    do stuff;
  }
CancelBtn_click()
  {
    do different stuff;
  }
App2Form_click()
  {
    do other stuff;
  }
```
Delegates

1. Register with a control to receive events
   - Give Control a function pointer to your callback function
     - `this.button1_Click += new EventHandler(this.button1_Click);`
2. Receive events from control
   - Control will call the function pointer
     - `private void button1_Click(object sender, EventArgs e){`

```
1.  button1.Click += button1_click();
2.  button1_Click();
```

Graphics

- Screen is like a painter’s canvas
- App must paint its window contents
  - GUI components paint themselves
  - Anything else: Programmer

1. How to paint?
2. When to paint?

Pixels

- Upside-down Cartesian

```
(0,0)          (width,0)

(0,height)    (width, height)
```

```
Y_{window} = height - Y_{cartesian}
```
Component Hierarchy

- Each component has its own subwindow
  - Subwindow = rectangular area within parent component
  - Has own coordinate system
- Clipping:
  - Can’t paint outside its subwindow
  - Can’t paint over child components?

Painting Components

- Can paint any component
- Panel is good for custom graphics area

Painting in C#:

1. The GDI+ graphics library:
   using System.Drawing;

2. Get the “graphics context” of a component:
   Graphics g = myPanel.CreateGraphics();

3. Paint in it:
   g.DrawLine(pen, x1,y1, x2,y2);

Graphics Primitives:

- Line (pt1,pt2)
- Lines (pt list)
- Arc (rect)
- Curves, Bezier (pt list)
- Ellipse (rect)
- Rectangle (rect)
- Polygon (pt list)
- Image (img, x,y)
- String (string, x,y)
**Graphics Attributes**

- **Pen** (for lines)
  - Color, width, dash, end caps, joins,
- **Brush** (for filling)
  - Color, Solid, texture, pattern, gradient
- **Font, String Format** (for strings)
- **Bitmap/Metafile** (for images)
  - Bmp, gif, jpeg, png, tiff, wmf, ...

**Color**

- Combinations of Red, Green, Blue
- Alpha value = opacity
- Each in [0, 255]
- C#: Color.FromArgb(255, 150, 0)

**Hokie Orange**

**Re-Paint**

- Screen is like a painter’s canvas
  - All windows paint on the same surface!
  - Windows don’t “remember” what’s under them
- Need to re-paint when portions are newly exposed
- Receive *Repaint events*
  - Open, resize, bring to front
  - When other windows in front move, resize, close
Repaint event sent to: Desktop, MyApp
MyApp gets repaint event

MyApp Form gets repaint event

MyApp gets repaint event

MyApp Form forwards repaint event to Button

---

Repainting Static Graphics

- Repaint event:
  - Erase (fill with background color) - usually automatically done by the control
  - Draw graphics

---

In C#

- Receive “paint” event:
  (select the event in VisStudio)
  ```csharp
  this.Paint += new PaintEventHandler(this.Form1_Paint);
  private void Form1_Paint(object sender, PaintEventArgs e)
  {
    Graphics g = e.Graphics;
    g.DrawLine(new Pen(Color.Red,10), 10,10,300,300);
  }
  
  OR: Override the OnPaint method
  ```
  ```csharp
  override void OnPaint(PaintEventArgs e)
  {
    base.OnPaint(e); //preserve base class functionality
    Graphics g = e.Graphics;
    g.DrawLine(new Pen(Color.Red,10), 10,10,300,300);
  }
  ```
- Can call Refresh() to force a repaint
Typical program structure for Dynamic Graphics

- Store data structure of graphics items
  - E.g. user drawn picture in a paint program

- Paint event:
  - Erase window (draw background color)
  - Draw graphics for items in data structure

- Other user events that alter the graphics items:
  - modify the data structure
  - send repaint event by calling Refresh()

Program Structure

```csharp
C# WinApp:

Class
  declare data storage;
  constructor(){
    initialize data storage;
  }

  cntrl1_paintevent(e){
    draw graphics from data;
  }

  cntrl2_mouseEvent(e){
    manipulate data;
    cntrl1.refresh();
  }
```

Data structure for graphics items

- 2 approaches:
  - Store logical contents in a data structure, then draw graphics based on the data
    - E.g. drawing program: lines, shapes, colors, ...
    - E.g. visualization program: data table
  - Store visual contents as an off-screen image (bitmap)
    - E.g. pixels
    - Then use g.DrawImage( ) in paint event

Direct Manipulation

Definition: (Shneiderman)

- Visual objects
- Selection by pointing
- Rapid, incremental, reversible actions
- Immediate, continuous feedback
Typical interaction sequence

1. Hit Testing
   - Mouse down, mouse over
   - Which dot did user click on?
   - Using components:
     - Make each dot a simple component, like a Button
     - Hit testing automatic, each component is a subwindow
     - Receive events from components, check event source
     - rectangular items, not scalable, inherit from UserControl
   - Using custom graphics:
     - Get click (x,y) from MouseDown event
     - Iterate through data structure, test for hit
       - E.g.: if rect.contains(x,y)
     - Data structure for fast lookup?

2. Dynamic Updating
   - Dragging, stretching, …
   - MouseMove events
   - Using components:
     - mouseDown store x,y click in component
     - mouseMove
       - Calculate x,y delta
       - Move component by delta
   - Using graphics:
     - (need to erase it, repaint other graphics, repaint new item)
     - Calculate delta, calculate new item location, store
     - Call Refresh( )
     - Draw new graphics in Paint event

Problem

- Dynamic manipulation on top of other graphics
  - Need to preserve (redraw) other graphics
  - Examples: MacDraw, powerpoint
- Simple solution:
  - Call refresh( ) or invalidate( ) while dragging
  - Paint( ) event restores other graphics
  - But: if lots of graphics, too slow & flashing!
Problem: Flashing

- Ugly flashing when repaint:
  - Paint background
  - Redraw shapes

Solution: Double buffering

- Double buffered repaint:
  - Draw all graphics in temporary off-screen image
    - Paint background color
    - Paint shapes
  - Then paint image to Window

- Bonus: C# can do this for you!
  - Form1.DoubleBuffered = true;
  
Rubber Band (XOR painting)

- Want multi-selection by stretching a rubber band
- Draw rubber band by inverting pixel colors
  - drawing with XOR once inverts background colors
  - drawing with XOR twice returns to original look
    -- No need to refresh(), fast!

  // in mouseMove event:
  // erase previous rect: (must use screen coords, not window coords)
  ControlPaint.DrawReversibleFrame(rect, Color.Black, FrameStyle.Dashed);
  // <update rect here based on mouse x,y>
  ControlPaint.DrawReversibleFrame(rect, Color.Black, FrameStyle.Dashed);

// SetStyle(ControlStyles.DoubleBuffer | // VS 2003
  ControlStyles.UserPaint |
  ControlStyles.AllPaintingInWmPaint, true);
Drag-n-Drop

- Drag and Drop API for GUI controls
  - Supports data transfer
  
  DestControl.AllowDrop = True;

  SourceControl_MouseEvent:
  
  this.DoDragDrop(data, DragDropEffects.Copy);

  DestControl_DragOver(e):
  
  e.Effect = DragDropEffects.Copy;

  DestControl_DragDrop(e):
  
  do something with e.Data.GetData(typeof(String));

Animation

- Update components/graphics in a loop:

  for(int i = 0; i < 100; i++)
  
  button2.Left += 10;

  for(int i = 0; i < 100; i++)
  
  myGraphicX += 10;

  refresh();

  but? Loops blocks other events.

Event-based Animation

- Use a Timer control

  - Non-visible control, fires a Tick event at specified intervals
    
    Timer1.Interval = 10 //milliseconds

  - Timer1.Enabled = true //starts animation

  - in Timer1_Tick() event:
    
    Update graphics
    Refresh()

  - Timer1.Enabled = false //stops animation

- Use doublebuffering

Software Architecture so far…

Program State

- data structures

Paint event

- display data

Interaction events

- modify data
Model-View-Controller (MVC)

- **Model**
  - Program State
  - data structures
- **View**
  - Paint event
  - display data
- **Controller**
  - Interaction events
  - modify data

Advantages?

- Multiple views for a model
  - Multi-view applications (overview+detail, brushing,…)
  - Different users
  - Different UI platforms (mobile, client-side, server-side,…)
  - Alternate designs
- Multiple models
- Software re-use of parts
- Plug-n-play
- Maintenance

Multiple Views
14

**E.g. C# TreeView Control**

- TreeView control
- `treeView1.Nodes`
- Nodes collection
- Java: model listeners

**C# DataBase Controls**

- DataGrid control
  - scroll, sort, edit, ...
- `DataSet` class
  - tables
  - columns
  - rows

**GUI Topics**

- Components
- Events
- Graphics
- Manipulation
- Animation
- MVC