

# Basic Debugging

C# Programming

January 31

# Part I

## 2 Things for TurnRed

## Color Pickers

- For TurnRed, you need to allow the user to choose the two colors for the game squares
- The easiest way to do this is to make user of the `ColorDialog` control
- This control is not the kind that gets added to your form and is displayed all the time
- But this control picker dialog can be opened when appropriate (like when the user wants to change the game colors)

## Color Pickers

- You will probably want to add two `ColorDialogs` to your application, one corresponding to each of the two square colors
- `ColorDialog` has a `Color` property that stores the currently selected color
- To open up the dialog, invoke the `ShowDialog()` method
- If the user closed the dialog using the OK button, the `Color` property is updated to the newly selected color

## Example

```
private void button1_Click(object sender, EventArgs e) {  
    DialogResult dr = colorDialog1.ShowDialog();  
    if (dr == DialogResult.OK) {  
        // NEW COLOR SELECTED  
    }  
}
```

## Making use of the `sender` parameter

- If you find yourself writing a separate event handler for each square click that is mostly copy and paste code, there is probably a better way to organize the logic
- For example, you probably want the same method to be invoked no matter which square is clicked
- The grid location of the square clicked would need to be passed to this method

## Making use of the `sender` parameter

- For example, say your custom square class is called `TurnRedSquare` and has properties `I` and `J` representing a square's location in the grid
- You create an instance of the class for each square of the grid and set the `Click` event handler for **all** squares to be the same, say `OnSquareClick`
- Then in this handler, you can use the `sender` parameter to figure out which square was clicked:

```
private void OnSquareClick(object sender, EventArgs e) {  
    TurnRedSquare s = (TurnRedSquare)sender;  
    int i = s.I, j = s.J;  
    UpdateGrid(i, j); // SAME METHOD CALL FOR ALL SQUARES  
}
```

## Part II

# Basic Debugging

# Debugging

- When tracking down bugs, you probably insert `MessageBox.Show()` calls to examine some values at particular points
- This approach often gets messy, makes the development process inefficient, and is not at all scalable
- Instead, you should use debugging tools to get at the runtime information you need

## Assert statements

- One thing you can do is add assert statements that check to see if an invariant you define holds true
- The `System.Diagnostics.Debug` class has a method `Assert()` that takes an expression of type `bool`
- At runtime, if that expression evaluates to `true`, then the effect of the `Assert()` is nothing
- If `false`, a dialog box is displayed with the stack trace from the assert that failed

Assertion Failed: Abort=Quit, Retry=Debug, Ignore=Continue



```
at Form1.button1_Click(Object sender, EventArgs e) C:\Ravi's Documents\Penn\Spring07\C#\SampleCode\ColorPicker\Form1.cs(24)
at Control.OnClick(EventArgs e)
at Button.OnClick(EventArgs e)
at Button.OnMouseUp(MouseEventArgs mevent)
at Control.WmMouseUp(Message& m, MouseButton button, Int32 clicks)
at Control.WndProc(Message& m)
at ButtonBase.WndProc(Message& m)
at Button.WndProc(Message& m)
at ControlNativeWindow.OnMessage(Message& m)
at ControlNativeWindow.WndProc(Message& m)
at NativeWindow.Callback(IntPtr hWnd, Int32 msg, IntPtr wparam, IntPtr lparam)
at UnsafeNativeMethods.DispatchMessageW(MSG& msg)
at ComponentManager.System.Windows.Forms.UnsafeNativeMethods.IMsoComponentManager.FPushMessageLoop(Int32 dwComponentID, Int32 reason, Int32 pvLoopData)
at ThreadContext.RunMessageLoopInner(Int32 reason, ApplicationContext context)
at ThreadContext.RunMessageLoop(Int32 reason, ApplicationContext context)
at Application.Run(Form mainForm)
at Program.Main() C:\Ravi's Documents\Penn\Spring07\C#\SampleCode\ColorPicker\Program.cs(17)
```

Abort

Retry

Ignore

## Assert statements

- Clicking Abort terminates execution of the program
- Ignore resumes execution of the program
- Retry resumes execution of the program in Debug mode, so that you can inspect runtime values

## Assert statements

- If you write `Assert()` statements during the development process – for example, asserting that a reference is non-null before invoking a method on it – then a failed assert can be useful
- However, it is unlikely that you will want to include `Asserts` everywhere in your code
- And it doesn't make much sense to go back and include an `Assert` at a location where you know the program crashes
- Instead, it is useful to run the program in `Debug` mode directly, so you can inspect runtime values

## Debug mode

- Running your application in Debug mode (F5 instead of Ctrl-F5) is very useful when tracking down bugs
- Without this mode, an unhandled exception will result in termination of the program
- In Debug mode, the line that caused the exception will become highlighted, and execution will pause there
- You can inspect the state of all variables and the call stack in the windows at the bottom

```
{  
    throw new Exception();  
    // NEW COLOR  
}
```



### Exception was unhandled



Exception of type 'System.Exception' was thrown.

#### Troubleshooting tips:

[Get general help for this exception.](#)

[Search for more Help Online...](#)

#### Actions:

[View Detail...](#)

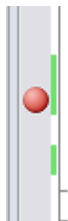
[Copy exception detail to the clipboard](#)

#### Locals

Name	Value
⊕ this	{ColorPicker.Form1, Text: Form1}
⊕ sender	Text = "button1"
⊕ e	X = 51 Y = 15 Button = Left
dr	OK
i	0
j	0

## Debug mode

- Before running your application, you can also set **breakpoints** in the code, which are places where execution will pause
- When stopped at a breakpoint, you can inspect runtime values, and even change them!
- To set breakpoints, click in the gutter to the left of the source code
- A red circle will appear, indicating that execution will pause before that statement is executed



```
DialogResult dr = colorDialog1.  
if (dr == DialogResult.OK)  
{  
    throw new Exception();  
    // NEW COLOR  
}  
}
```

## Debug mode

- Execution will pause at each breakpoint
- To resume from a breakpoint, click Continue (or F5)
- To terminate the program completely, click Stop Debugging (or Shift-F5)
  
- Using these basic debugging features makes tracking down bugs much easier and effective