Basic Debugging

C# Programming

January 31

---

**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 use 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)

**Example**

```csharp
goto (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

---

**Color Pickers**

- You will probably want to add two `ColorDialog`s 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
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:

```csharp
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
}
```

**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

- 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

### 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

**Breakpoint example:**
```java
if (dr == DialogResult.OK) {
    throw new Exception();
    // NEW COLOR
}
```