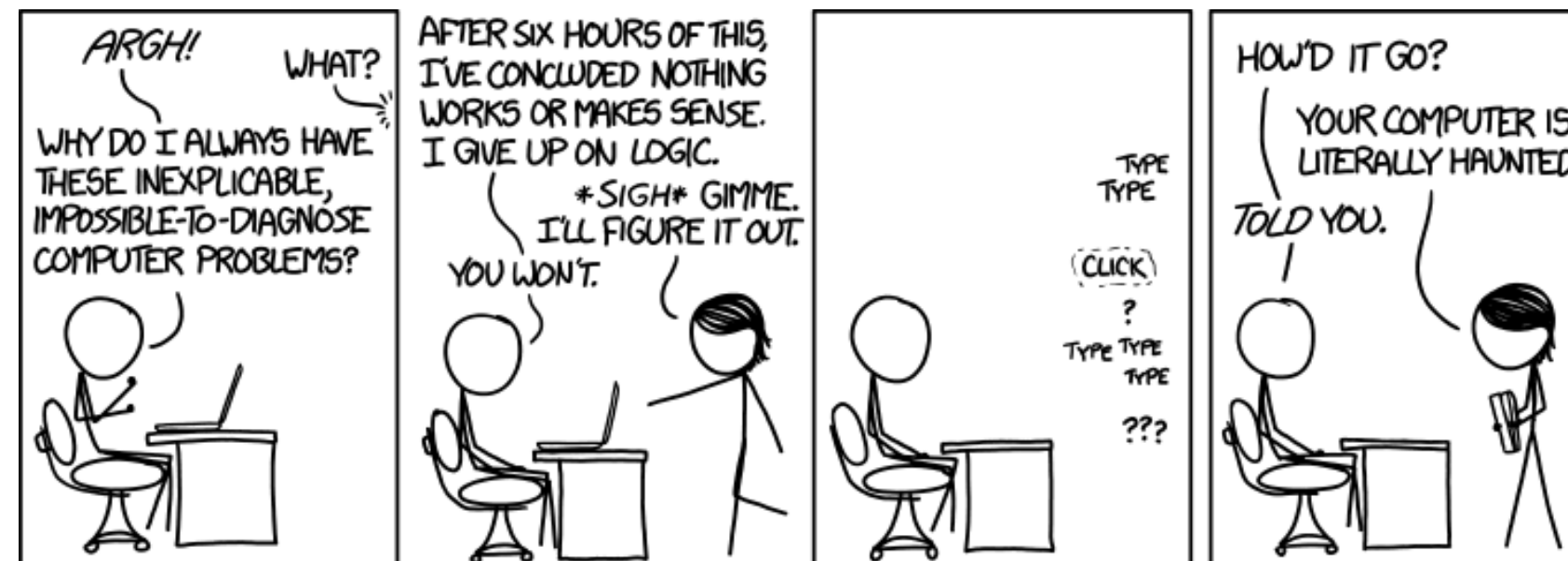


Debugging (and a bit of testing)



<https://xkcd.com/1316/>

By Randall Munroe



We just don't fully understand what the program does under all inputs



We may think we do, but we almost always only have a partial understanding

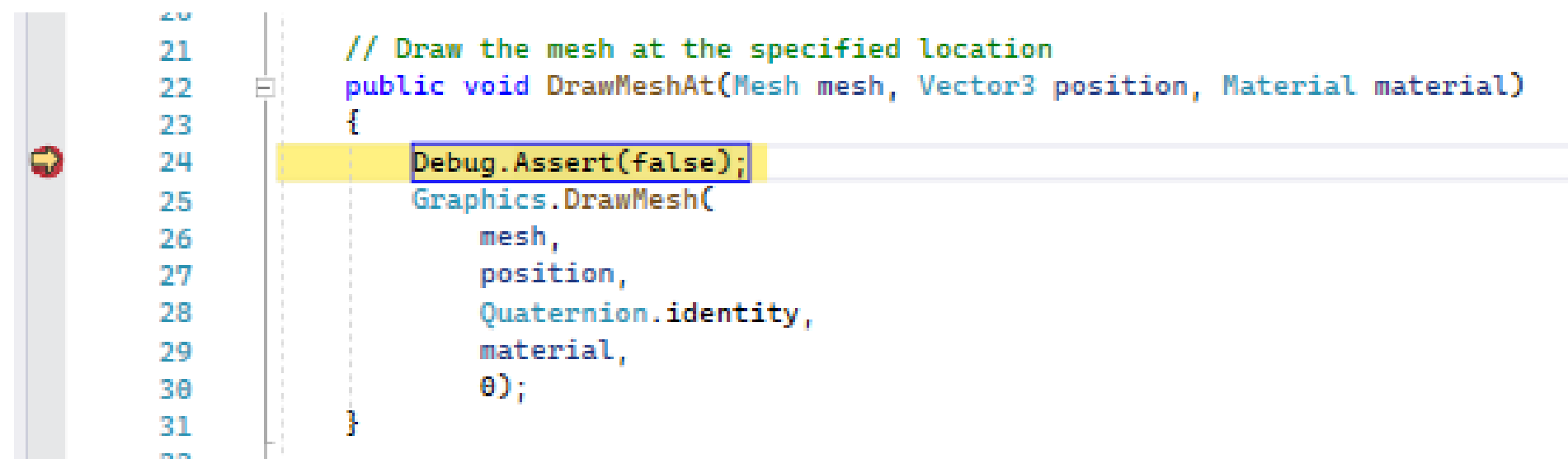
Programs do exactly what we ask!

Debugging

Bugs are errors, or unexpected behaviors within code. Debugging is the process of trying to figure out how a computer program working and why it is doing what it is doing.

Some basic tools for debugging are:

1. Trace Statements
2. Assertion
3. Breakpoints
4. Watches
5. Tests



```
20
21 // Draw the mesh at the specified location
22 public void DrawMeshAt(Mesh mesh, Vector3 position, Material material)
23 {
24     Debug.Assert(false);
25     Graphics.DrawMesh(
26         mesh,
27         position,
28         Quaternion.identity,
29         material,
30         0);
31 }
```

The screenshot shows a code editor with a C# method `DrawMeshAt`. A breakpoint is set on line 24, which contains the statement `Debug.Assert(false);`. The line is highlighted in yellow, and a red circle with a white arrow points to it. The method signature is `public void DrawMeshAt(Mesh mesh, Vector3 position, Material material)`. The method body contains a call to `Graphics.DrawMesh` with arguments `mesh`, `position`, `Quaternion.identity`, `material`, and `0`.

Debugger

- A debugger is a computer program that can attach itself to a running process, pause it, step through it line by line, and allow you to view the values associated with variables.
- Debuggers often provide a console window which displays the result from trace statements.
- When running a computer program from within the development environment usually the debugger is already attached.

Debug and Release Mode

Computer programs are often compiled in one of two modes: debug and release. More are possible (e.g., hybrid)

In debug mode a symbol file (.pdb) is generated which links binary executable to source files.

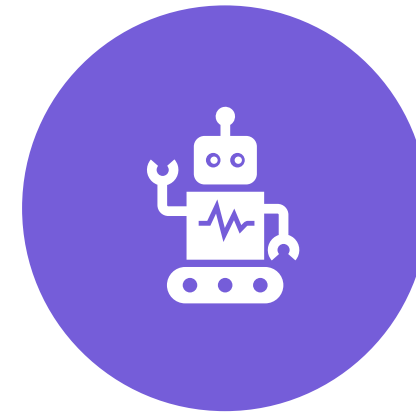
Release mode builds are usually optimized, certain instructions are removed from the executable, and a PDB is not generated.

Another name for a “debug” build is a “development” build.

Code not run during Release Build



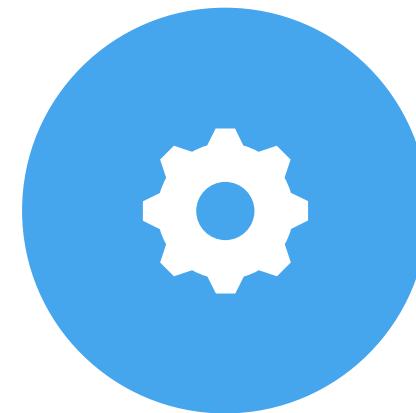
Code between `#if DEBUG` and `#endif`



Code that is conditional on `Debugger.IsAttached`



`Debug.Assert()`



`Debug.WriteLine()`

The background features a dark, grayscale image of a ruler and a document. The ruler is positioned diagonally across the frame. On the document, the number '5' is visible. The entire image is framed by a thick purple border.


**ONLY DISTRIBUTE AND MEASURE
RELEASE BUILDS**

Tracing


- A function call, which outputs information to a console window, is called a trace statement.
- `Debug.WriteLine()` outputs text to the Visual Studio output window.
- You can use `Console.WriteLine()`, but be careful about leaving in debug information during releases
- When using Unity, you can use `Debug.Log()` which outputs to the Unity console.
- There is also a [Trace](#) class, which is intended for instrumenting release builds
- See also the [Trace and Debug Topic](#) in C# documentation

Debug Assertions

An assertion statement is a function call to which takes a conditional expression that is expected to be true.



If the condition evaluates to false, and a debugger is attached the program will be paused.



All assert calls will be conditionally included only in a development/debug builds



Assert statements are great ways to document and test your assumptions about code.

Test Assertions

1

In unit tests we also have assertions

2

These are run during both release builds and debug builds

3

Test assertions should be in a separate test project and not distributed

4

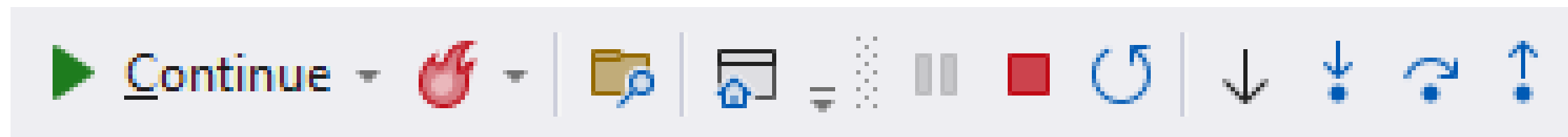
If an assertion fails during a test run, the test is considered non-passing

5

If an uncaught exception occurs, then the test is also considered non-passing

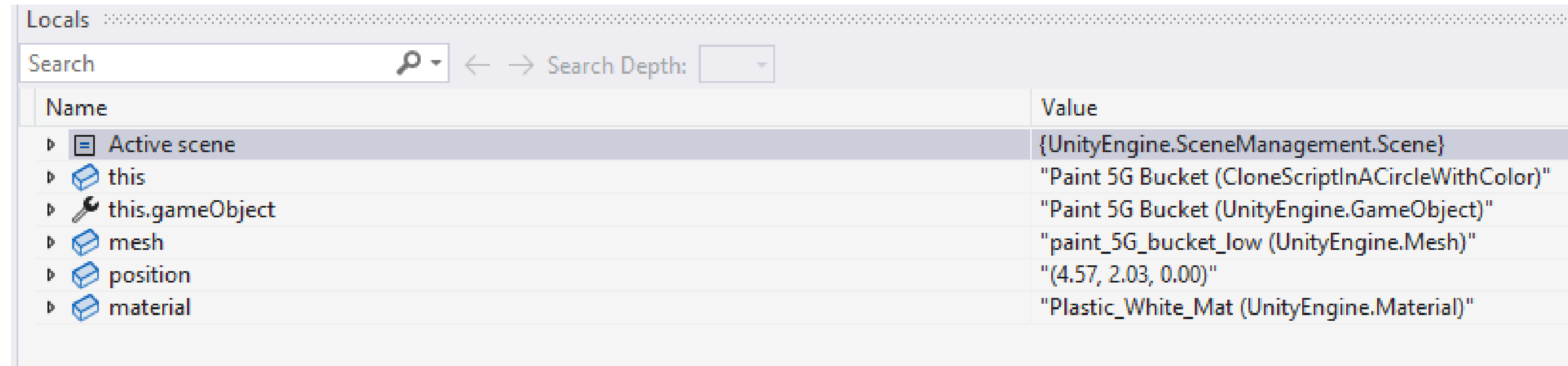
Breakpoints

- A breakpoint is a marker assigned to an instruction within a computer program that triggers the debugger to pause.
- In C# you can also use the `Debugger.Break()` to programmatically pause any attached debugger.
- If no debugger is attached, you can also use the `Debugger.Launch()` function to launch and attach a debugger.
- Once you hit a breakpoint, you can step through code one line at a time.



Watches

- A watch is a window that shows the value associated with a variable or expression.
- We can view all local variables, or specific expressions.
- For more information on using watches [see this article](#).



Debugging Advice

- Computers and compilers almost never make mistakes.
- Virtually of the time, problems arise from an incomplete understanding of what we have asked the computer to do.
- When you are stuck, the problem often boils down to unidentified and incorrect assumptions.
- You have assumed something about how either the compiler or program works, and you are not aware of what assumptions you have made.
- Write out your assumptions as assertion statements or comments

Start Debugging launches Startup Project

A solution can have multiple executables

Pressing F5 (start debugger) launches the start-up project

Uses the current debug profile

You can have multiple profiles per project

Launch Profiles

🔍 📄 ✖ 🗑

📁 HelloCommandLineArgs

Command line arguments
Command line arguments to pass to the executable. You may break arguments into multiple lines.

C:\Users\cdigg\git\cs321\class-notes

Working directory
Path to the working directory where the process will be started.

Browse...

Use remote machine
☐ Indicates that the debugger should attach to a process on a remote machine.

Environment variables
The environment variables to set prior to running the process.

Name	Value
<input type="text"/>	<input type="text"/>

DEBUG PROFILES

Debug Profiles

Set	Set	Set	Launch
Command line arguments	Working directory	Environment variables	Project or executable

Testing

Tests are a kind of debug tool



```
graph TD; A[Tests are a kind of debug tool] --> B[They help you narrow in on problems and validate assumptions]; B --> C[They fail more often than you might think!]; C --> D[Not all tests have to pass ... sometimes they just help you understand limits];
```

They help you narrow in on problems and validate assumptions

They fail more often than you might think!

Not all tests have to pass ... sometimes they just help you understand limits

Test Driven Development (TDD)



Starting with tests and filling out implementations is a powerful problem-solving technique



I do not recommend it as a complete software development methodology, just a tool

```
public static int Fibonacci(int n)
{
    throw new NotImplementedException();
}

[Test]
public static void TestFibonacci1()
{
    var seq = new[] { 0, 1, 1, 2, 3, 5, 8, 13, 21 };
    for (var i = 0; i < seq.Length; i++)
    {
        Assert.AreEqual(seq[i], Fibonacci(i));
    }
}

[Test]
public static void TestFibonacci2()
{
    for (var i = 2; i < 99; i++)
    {
        Assert.AreEqual(Fibonacci(i - 2) + Fibonacci(i - 1), Fibonacci(i));
    }
}
```

```
public static int Factorial(int n)
{
    throw new NotImplementedException();
}

[Test]
public static void TestFactorial1()
{
    var seq = new[] { 1, 1, 2, 6, 24, 120, 720 };
    for (var i = 0; i < seq.Length; i++)
    {
        Assert.AreEqual(seq[i], Factorial(i));
    }
}

[Test]
[TestCase(99)]
public static void TestFactorial2(int max)
{
    for (var i = 0; i < max; i++)
    {
        // This is effectively the definition of Factorial
        Assert.AreEqual(Factorial(i - 1) * i, Factorial(i));
    }
}
```