

Squiggle Documentation

The purpose of Squiggle is to assist with debugging code. You can track variables to see how they change over time, using the `DebugGraph` class to output to the Squiggle window or by attaching the `ComponentMonitor` to a gameobject and selecting variables to watch.

Open the Graph Console window by clicking the “Window-> Squiggle” menu button.

Example graphs can be found by clicking the “?” button in the top right of the Squiggle window.

Graphing Values

Quickly Visualise Data: `DebugGraph.Log(Value)`

A graph will form as data changes over time. Supported types are Floating Point, Integer, Vector2, Vector3, Color, Color32 and Boolean.

Create Multiple Graphs: `DebugGraph.Log(Name, Value)`

Unnamed Data will be logged on a single chart, data logged with a name is logged to a chart of that name.

Color Your Data: `DebugGraph.Log(Name, Value, Color)`

Every logged value can have a different color, this enables you to create a multi-colored graph which makes it easy to find what you are looking for.

Timestamp Your Data: `DebugGraph.Log(Name, Value, Color, Time)`

`Time.realtimeSinceStartup` is used as the default timestamp, but you can specify your own if you need multiple samples per time step. For example, this could be used to discover the most expensive part of a function or to visualise the order in which code is executed. Tip: use colors to identify different sections of code.

Create a Strings Graph: `DebugGraph.Write(...)`

Using `DebugGraph.Write` instead of `DebugGraph.Log` allows you to output any object as a string. Each output string will be represented by a vertical line on the graph.

Trace a Path: `DebugGraph.Draw(...)`

Using `DebugGraph.Draw` instead of `DebugGraph.Log` allows you to trace a line following a Vector2 value.

Create a Color Gradient:

Any logged Color or Color32 value, will automatically be represented as both a color gradient and the graph of its RGBA values.

Squiggle Documentation

Comparing Values

Quickly Compare Data: `DebugGraph.MultiLog(Color, Value)`

Each color logged will form its own graph as data changes over time. This allows you to compare multiple values on the same chart.

Multiple Comparisons: `DebugGraph.MultiLog(Name, Color, Value)`

Unnamed Data will be logged on a single chart, data logged with a name is logged to a chart of that name.

Timestamp Your Data: `DebugGraph.MultiLog(Name, Value, Color, Time)`

`Time.realtimeSinceStartup` is used as the default timestamp, but you can specify your own if you need multiple samples per time step.

Component Monitor

Watch Component Variables: Attach the ComponentMonitor to a gameobject, then click on the [+] button and select a variable to track.

Stop Watching Component Variables: Click on the [-] button to the left of the variable you want to stop watching.

Select Sample Time: Choose Update, FixedUpdate or LateUpdate from the SampleTime dropdown.

Squiggle Window

View Logged values: Move the mouse along a graph to view the values, click on the graph to freeze/unfreeze the timeline sampler.

View Minimum, Maximum and Middle values: These are displayed on the left hand side of a graph console window.


Zoom: While holding Ctrl or Cmd, click on a graph with the left mouse button then drag up/down. (Holding Shift will zoom all graphs)

Pan: While holding Ctrl or Cmd, click on a graph with the left mouse button then drag left/right. (Holding Shift will pan all graphs)

Stretch: Change the height of a Graph by dragging the grey bar below a graph.

Squiggle Documentation

Toolbar Buttons

 **Record:** When Record is enabled all graph data will be remembered. When Record is disabled, only a short preview of graph data will be shown.


Clear: Removes all recorded data from the graph console.

Shrink: Reduces all graphs to their minimum height.

Restore: Returns all graphs to their initial zoom/pan state.

Snap To End: Moves the timeline sampler to the end of a graph, this enables you to view the most recent value input to the graphs.

 **Settings:** Opens the Squiggle settings window.

 **Export Data:** Opens the graph exporting window, which provides options for exporting graph data to file.

?: Opens the help window, which includes example graphs.

Debug Graph Renderer

Render Graphs In Game: Attach the DebugGraphRenderer to a gameobject, then enter the name of the graph to be rendered in the Graph Name field.

Custom Graph Material: Custom Materials can be assigned to the DebugGraphRenderer. There are three texture coordinates to available. 0: Graph Point (X, Y), 1: Graph Distance (X, Y) 2: Normalized Point (X, Y)

Advanced Usage (Mystery.Graphing namespace)

Custom Graphing: `DebugGraph.AddCustomGraph(Name, IDebugGraph)`

You can create your own graphs by inheriting `LinearDebugGraph<X, Y>` or by implementing the `IDebugGraph` interface. You can then display these in the Graph Console Window by using the `AddCustomGraph` function.

Custom Charting: `DebugGraph.AddCustomGraph(IGraphConsole)`

You can programmatically control your own charts by creating an instance of `SingleGraphConsole<X, Y>`, `MultiGraphConsole<X, Y>`, by inheriting `GraphConsole` or by implementing the `IGraphConsole` interface. You can then display these in the Graph Console Window by using the `AddCustomGraph` function.

Questions and Suggestions

Email: mark@mysterytech.net

Twitter: [@mysterytechs](https://twitter.com/mysterytechs)