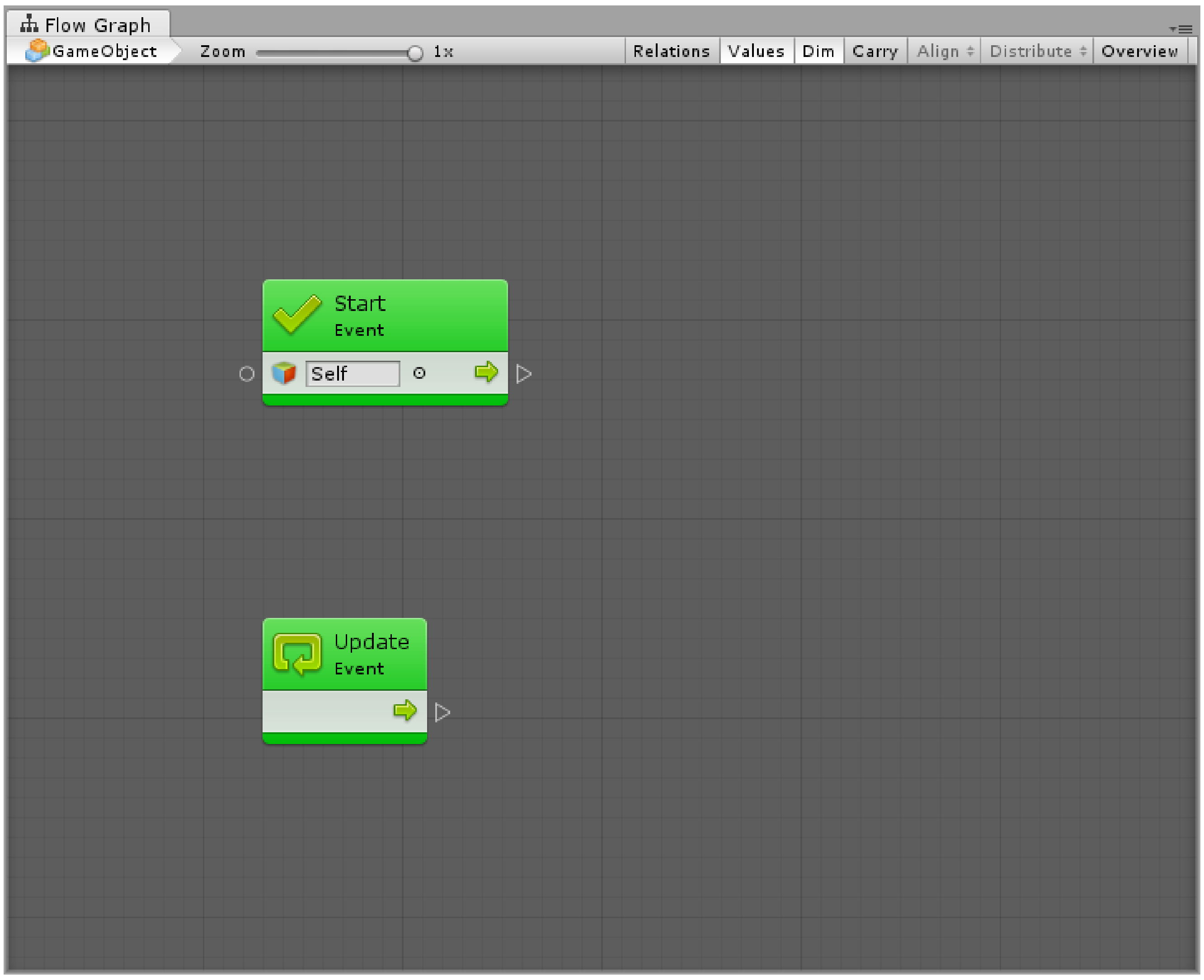
Units & Ports (/topics/149-units-ports/)

Before starting this section of the manual, we assume that you've read the Introduction (http://support.ludiq.io/forums/4-bolt/categories/37-introduction/topics/), that you're familiar with the Basic Concepts (http://support.ludiq.io/forums/4-bolt/categories/38-basic-concepts/topics/) and that you've created a Flow Machine (http://support.ludiq.io/topics/138-graphs-machines-graph-assets/).

At this point, you should have a flow graph with a Start and Update event:



Units

Units are the most basic element of computation in Bolt. They are represented as nodes with input and output ports in flow graphs. Units can do a do a wide variety of things, for example listen for an event, get the value of a variable, invoke methods on components and game objects, etc.

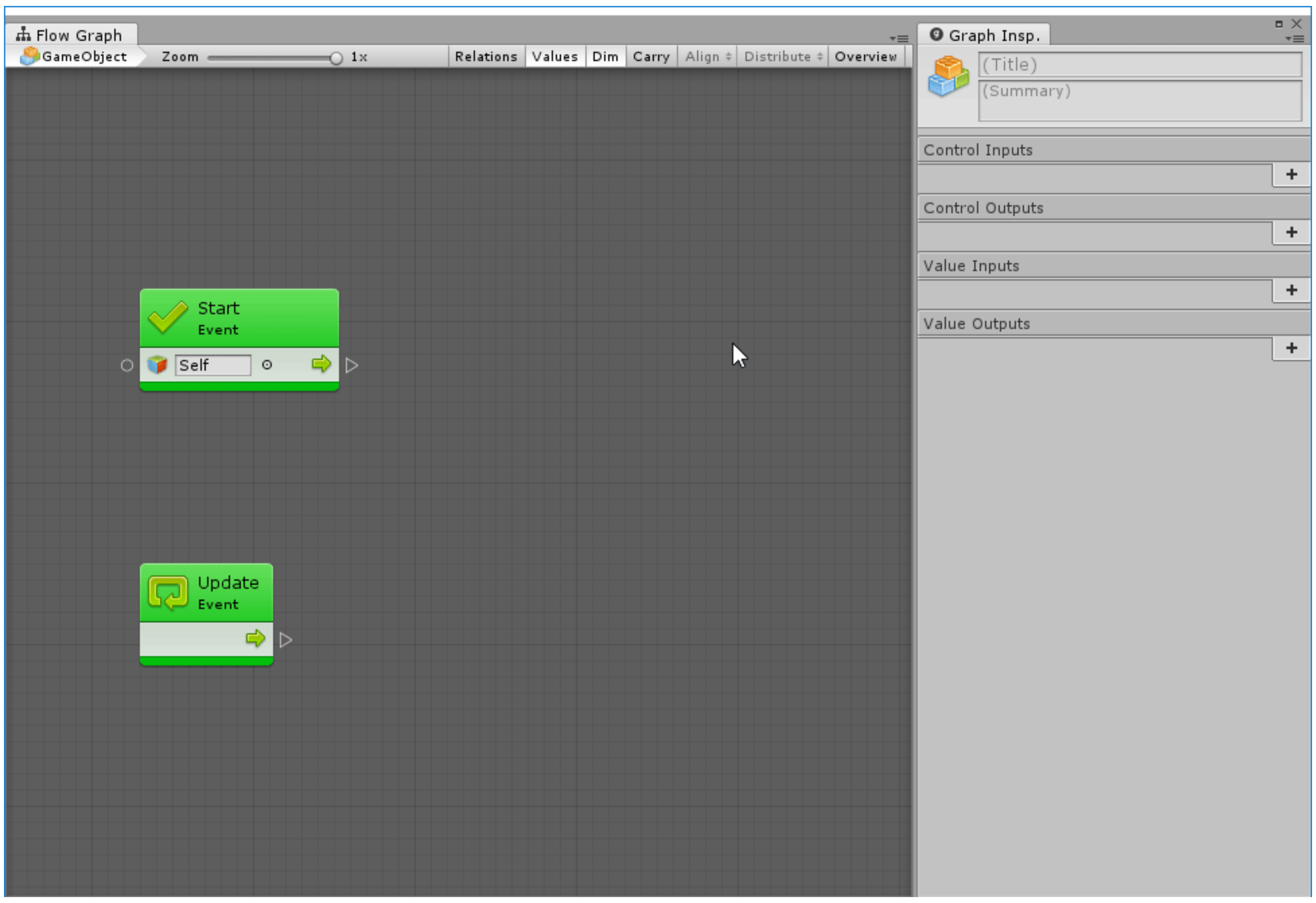
Units use connections to indicate in what order they should be called and to pass values from one another. We'll cover connections in the next article. For now, let's focus on units and their ports.

Creating Units

By default, there are **over 23 000** available units in Bolt. They include the entire Unity scripting API, as well as all the methods and classes from your custom scripts or third party plugins. Finally, there are some additional utility units for math, logic, variables, loops, branching, events and coroutines.

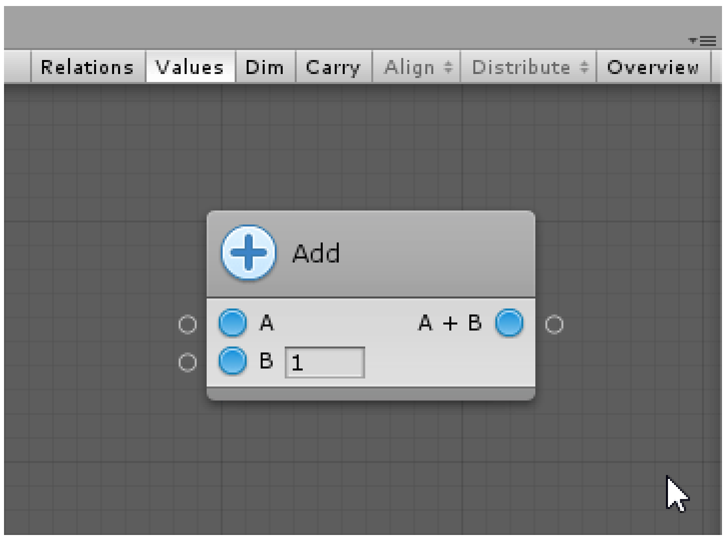
Fortunately, these units are well organized in a simple, searchable creation menu called the fuzzy finder .

To display the fuzzy finder, simply right-click anywhere in the empty grid. You can then browse through the categories or search in the top field to quickly find a unit. Here, for example, we're adding a simple scalar addition (float + float) unit with both methods:

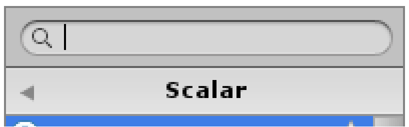


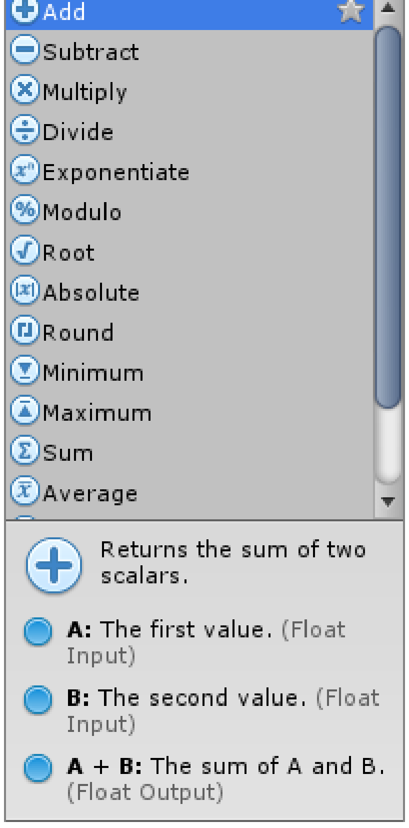
The first thing you'll notice is that your new units appear as dimmed out. This is because Bolt warns you that their value is never used. Indeed, we're not using the result of the addition anywhere, so these nodes are currently "useless".

This is a very useful predictive debugging feature, but since we're not going to be connecting nodes until the next article, you might want to disable it to see what you're doing for now. You can toggle dimming with the `Dim` button in the toolbar:



The fuzzy finder gives you a preview documentation of each unit before you even create it. For example, for the add node, we can know what it does and what ports it has straight from the fuzzy finder:




Add
Subtract
Multiply
Divide
Exponentiate
Modulo
Root
Absolute
Round
Minimum
Maximum
Sum
Average

Returns the sum of two scalars.

A: The first value. (Float Input)

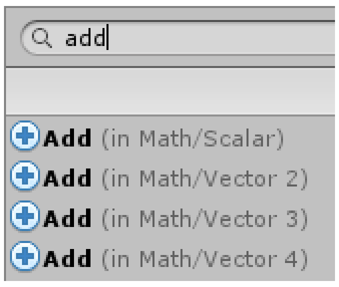
B: The second value. (Float Input)

A + B: The sum of A and B. (Float Output)

Overloads

Some units have multiple variations, which are called overloads .

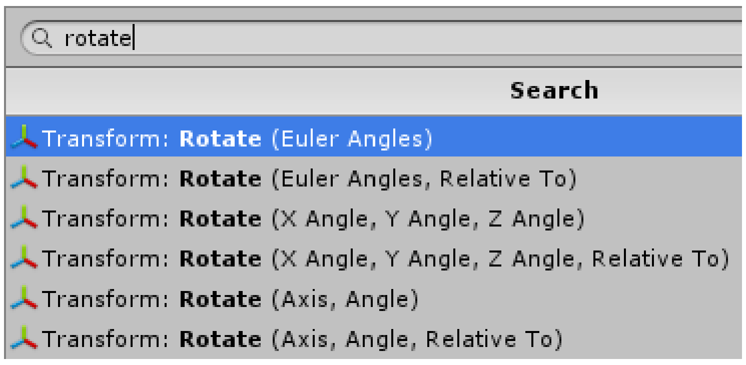
For example, there are 4 Add units: one for scalars, and one for 2D vectors, 3D vectors and 4D vectors. In this case, you can use their category to distinguish them.


add

Add (in Math/Scalar)
Add (in Math/Vector 2)
Add (in Math/Vector 3)
Add (in Math/Vector 4)

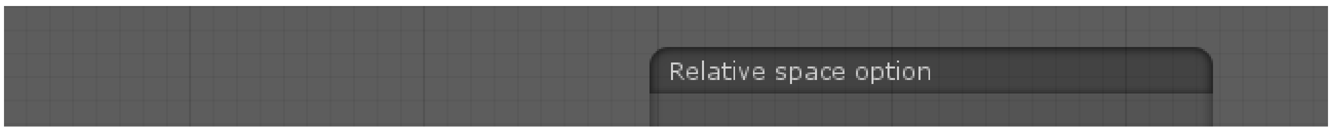
Some method units have parameter overloads. Usually, these variations are for convenience and each will do roughly the same thing. Some overloads allow for more specific configuration than others.

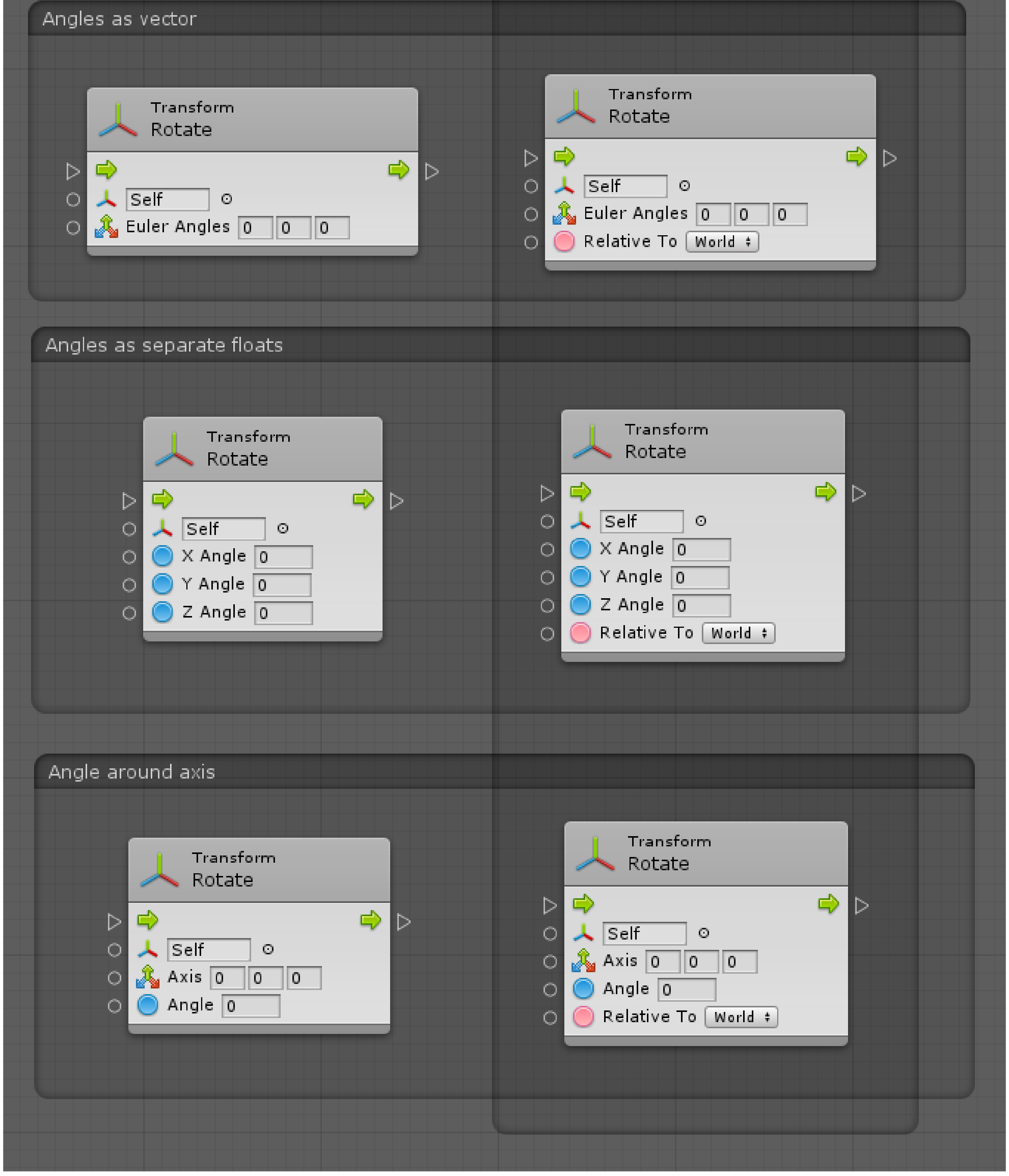
For example, the Rotate Transform unit has 6 overloads. Two of them take the angles as a single euler angle vector, two other take it as 3 separate float components in X / Y / Z, and the last two take it as an angle relative to the axis. In each pair, one allows to specify the relative space, while the other just assumes that you're specifying angles in world space. Here's a screenshot of all 6 overloads for the rotate unit:


rotate

Search

Transform: **Rotate** (Euler Angles)
Transform: **Rotate** (Euler Angles, Relative To)
Transform: **Rotate** (X Angle, Y Angle, Z Angle)
Transform: **Rotate** (X Angle, Y Angle, Z Angle, Relative To)
Transform: **Rotate** (Axis, Angle)
Transform: **Rotate** (Axis, Angle, Relative To)





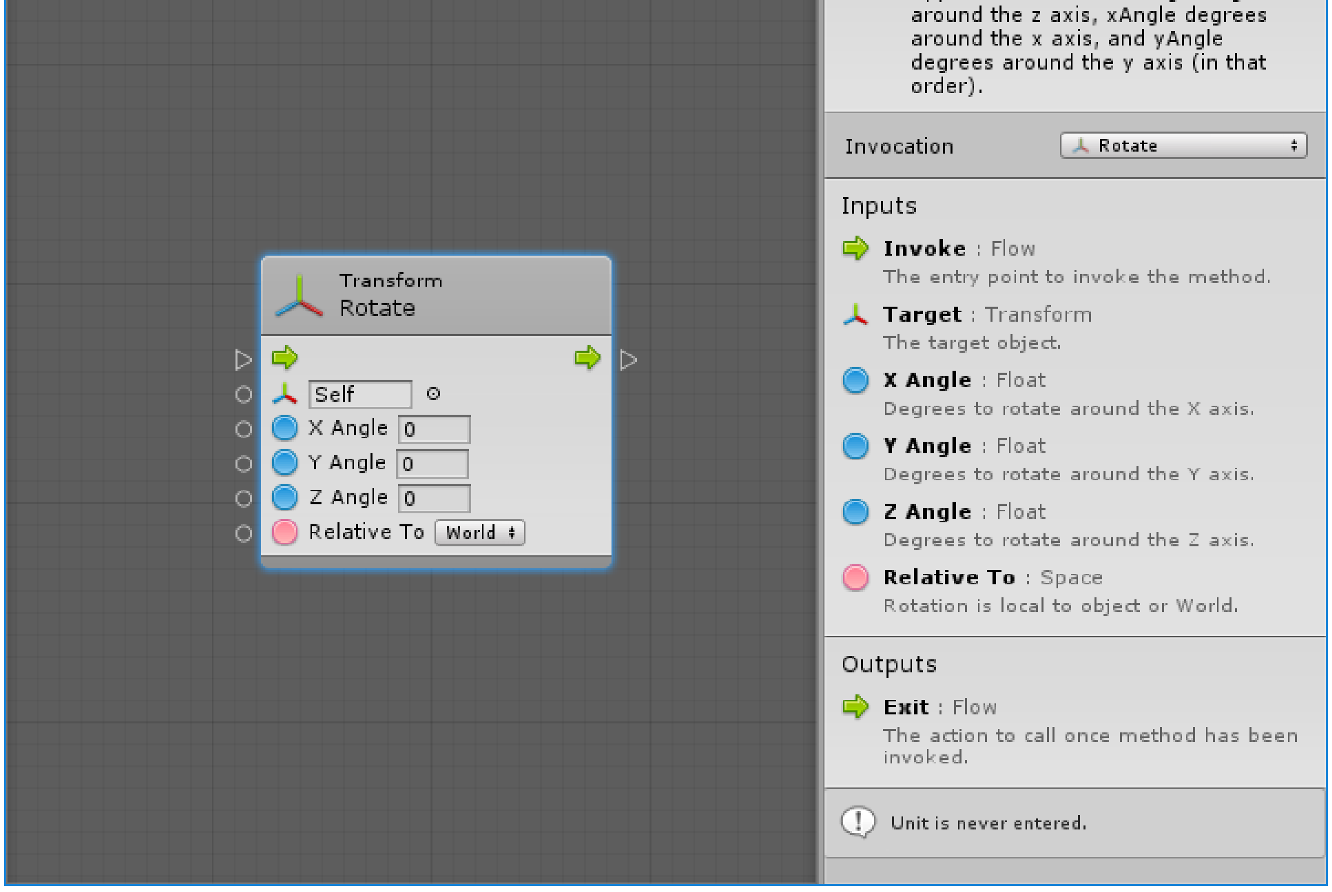
It might take some trial and error to find the right overload at first, but you'll quickly get used to the available options. You can use the built-in documentation or the Unity manual to help you distinguish between each variation.

Take a moment to explore the unit options and browse around the fuzzy finder. But don't worry if you're overwhelmed at first: we'll have a look at every kind of unit over the next few articles.

Reading Units

Let's look at the anatomy of a unit. In this example, we created a Rotate Transform unit, which you can find under Codebase > Unity Engine > Transform > Rotate (X, Y, Z, Relative To) .

Rotate
Applies a rotation of zAngle degrees


around the z axis, xAngle degrees around the x axis, and yAngle degrees around the y axis (in that order).

Invocation: Rotate

Inputs

Invoke : Flow
The entry point to invoke the method.

Target : Transform
The target object.

X Angle : Float
Degrees to rotate around the X axis.

Y Angle : Float
Degrees to rotate around the Y axis.

Z Angle : Float
Degrees to rotate around the Z axis.

Relative To : Space
Rotation is local to object or World.

Outputs

Exit : Flow
The action to call once method has been invoked.

Unit is never entered.

The top part of a unit is its header. It's a quick summary of what the unit does. In this case, it tells us that it is invoking the Rotate (https://docs.unity3d.com/ScriptReference/Transform.Rotate.html) method on a Transform (https://docs.unity3d.com/ScriptReference/Transform.html) component.

You can tell the unit is selected because of the slight blue glow around its edge. When a unit is selected, its options and documentation will show in the Graph Inspector, which is placed on the right of the window in this screenshot.

Ports

Ports are hooks that you can use to connect nodes together.

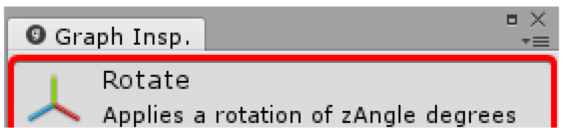
On the left side, you'll find the **Input Ports** .

On the right side, you'll find **Output Ports** .

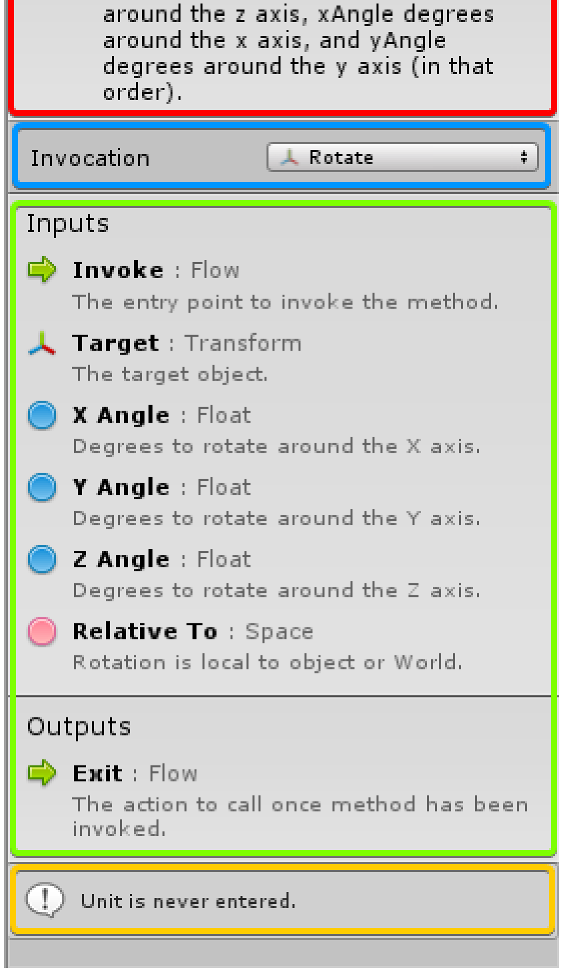
- ▷ **Control Ports** are used to connect the ➡ **Flow** . Think of the flow as the order in which nodes should be executed. Flow always goes from left to right, hence the direction of the little arrow.
- ○ **Value Ports** are used to connect... well, values. Each value port has a Type (http://support.ludiq.io/topics/132-types/) that must be matched when connecting nodes.

Unit Inspector

Let's break down the unit inspector:


Graph Insp.

Rotate
Applies a rotation of zAngle degrees


around the z axis, xAngle degrees around the x axis, and yAngle degrees around the y axis (in that order).

Invocation: Rotate

Inputs

Invoke : Flow
The entry point to invoke the method.

Target : Transform
The target object.

X Angle : Float
Degrees to rotate around the X axis.

Y Angle : Float
Degrees to rotate around the Y axis.

Z Angle : Float
Degrees to rotate around the Z axis.

Relative To : Space
Rotation is local to object or World.

Outputs

Exit : Flow
The action to call once method has been invoked.

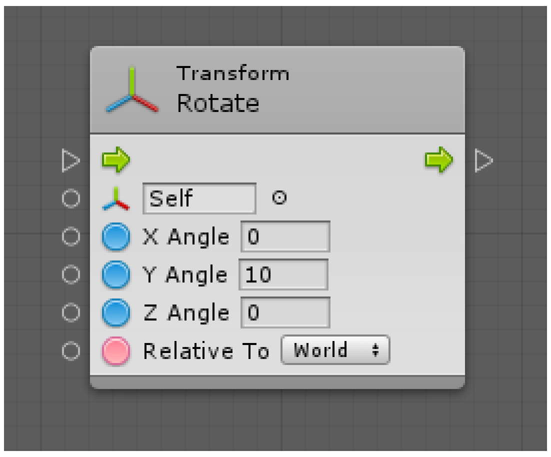
Unit is never entered.

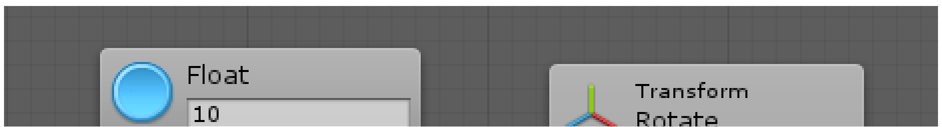
1. At the top, in the **red rectangle**, you can see the title and summary for the unit, which gives you a quick overview of what it does.
2. Below, in the **blue rectangle**, you have the unit's settings . They vary from unit to unit, and some units don't even need settings. In this case, we could change the method that we are invoking if we wanted.
3. In the **green rectangle**, you have the documentation for each port . First its name (e.g. "X Angle"), then its type (e.g. "Float"), and finally its summary ("Degrees to rotate around the X axis").
4. Finally, in the **yellow rectangle** at the bottom, Bolt will display all the warnings for the unit. For example, here, Bolt warns us that the unit is never entered, because we never connected the "Invoke" control input port. If we had dimming enabled, this unit would therefore be dimmed out.

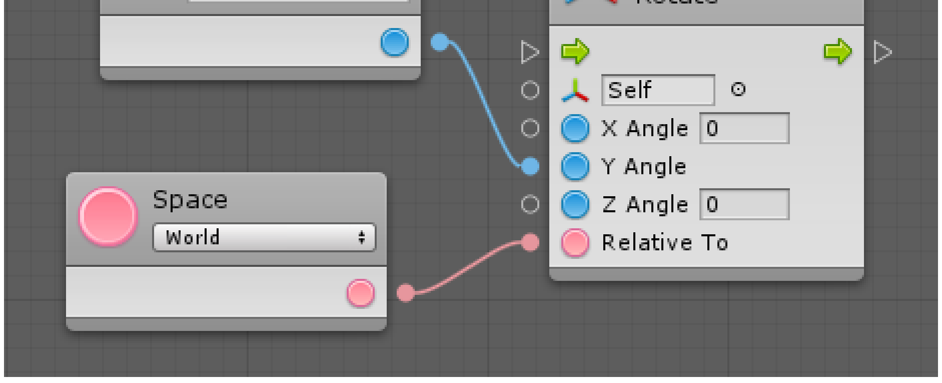
Inline Values

You'll have noticed by now that some value input ports have small fields next to them. These are called **Inline Values** . If the port is not connected, the value of this field will be used instead. Most common types support inline values, but not all types do. Inline values are useful to keep your graphs tidy by avoiding the creation of literal nodes for every value input port.

For example, these two graphs are exactly equivalent:







Now that we're familiar with units, let's have a look at how to create these connections!