Graphs, Machines & Macros (/topics/138-graphs-machines-macros/)

Graphs are visual representations of logic. They're at the core of Bolt.

There are two kinds of graphs:

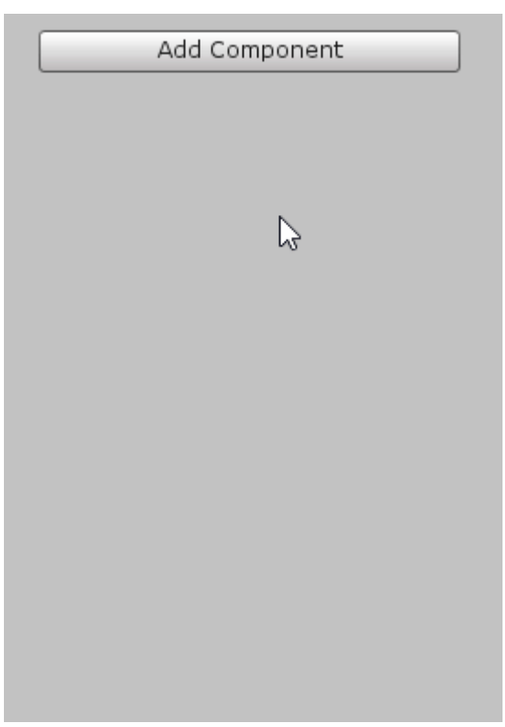
- **Flow Graphs** , in which you connect individual actions and values in a specific order. That order of execution is what we call the *flow*. If you have used Unreal Engine before, you might find they are similar to the Blueprints visual scripting language.
- **State Graphs** , in which you create different states and the transitions between them. Each state acts as a little program. If you have used PlayMaker or other FSM (Finite State Machine) systems before, you're familiar with state graphs.

Together, these two kinds of graphs allow you to create any game you have in mind.

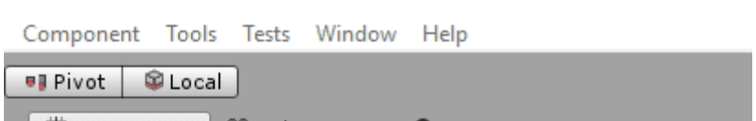
Each kind has its own full dedicated section in the manual, Flow Graphs (http://support.ludiq.io/forums/4-bolt/categories/41-flow-graphs/topics/) and State Graphs (http://support.ludiq.io/forums/4-bolt/categories/40-state-graphs/topics/). There is also a third section, Mixing It Up (http://support.ludiq.io/forums/4-bolt/categories/42-mixing-it-up/topics/), that shows how to combine them. For now, let's just introduce some basic concepts that are shared for both kinds of graphs.

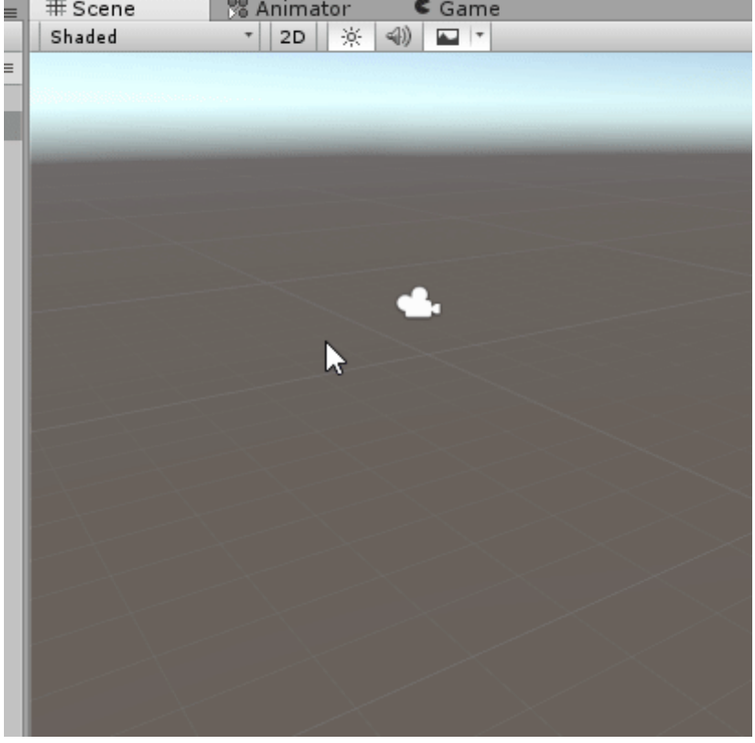
Machines

A **Machine** is a component that you add on a game object to *execute* the logic of a graph during play mode. There is one for each kind of graph: a **Flow Machine** and a **State Machine** . They are both located under the Bolt category.

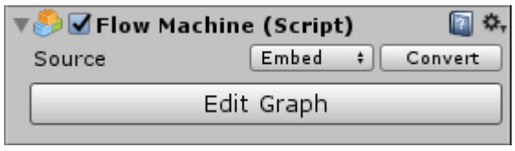


You can also find them in the top level Unity menu:





Both have the same options in the inspector, so we'll just create a flow machine for this example:



The Edit Graph button opens the graph of this machine in the graph window and the graph inspector. If you only have one machine on the same object, you won't need to use it, because Bolt automatically matches the graph to your selection.

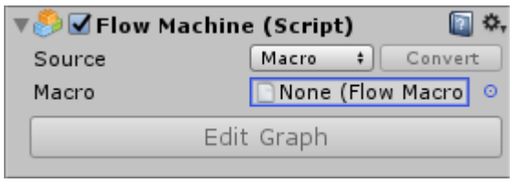
Let's take a look at the **Source**. You have two options: Embed or Macro.

	Embed	Macro
Relation	The graph is *embedded* in the machine itself.	The graph is a macro asset that is *referenced* by the machine.
Re-usability	You cannot re-use the graph for other machines, but it will be shared across prefab instances.	You can re-use the same macro for multiple machines, even if they're not on the same prefab.
Permanence	If you remove the machine component, the graph will be deleted. The graph will **not** be deleted if you switch the Source to Macro, it will only be hidden and unused unless you switch it back to Embed.	If you remove the machine component, the macro asset will still exist.
Scene References	The graph can refer to game objects from the current scene in its graph, as long as it's not saved as a prefab.	The graph cannot refer to game objects from the current scene, because it does not "belong" to any scene.

Macros

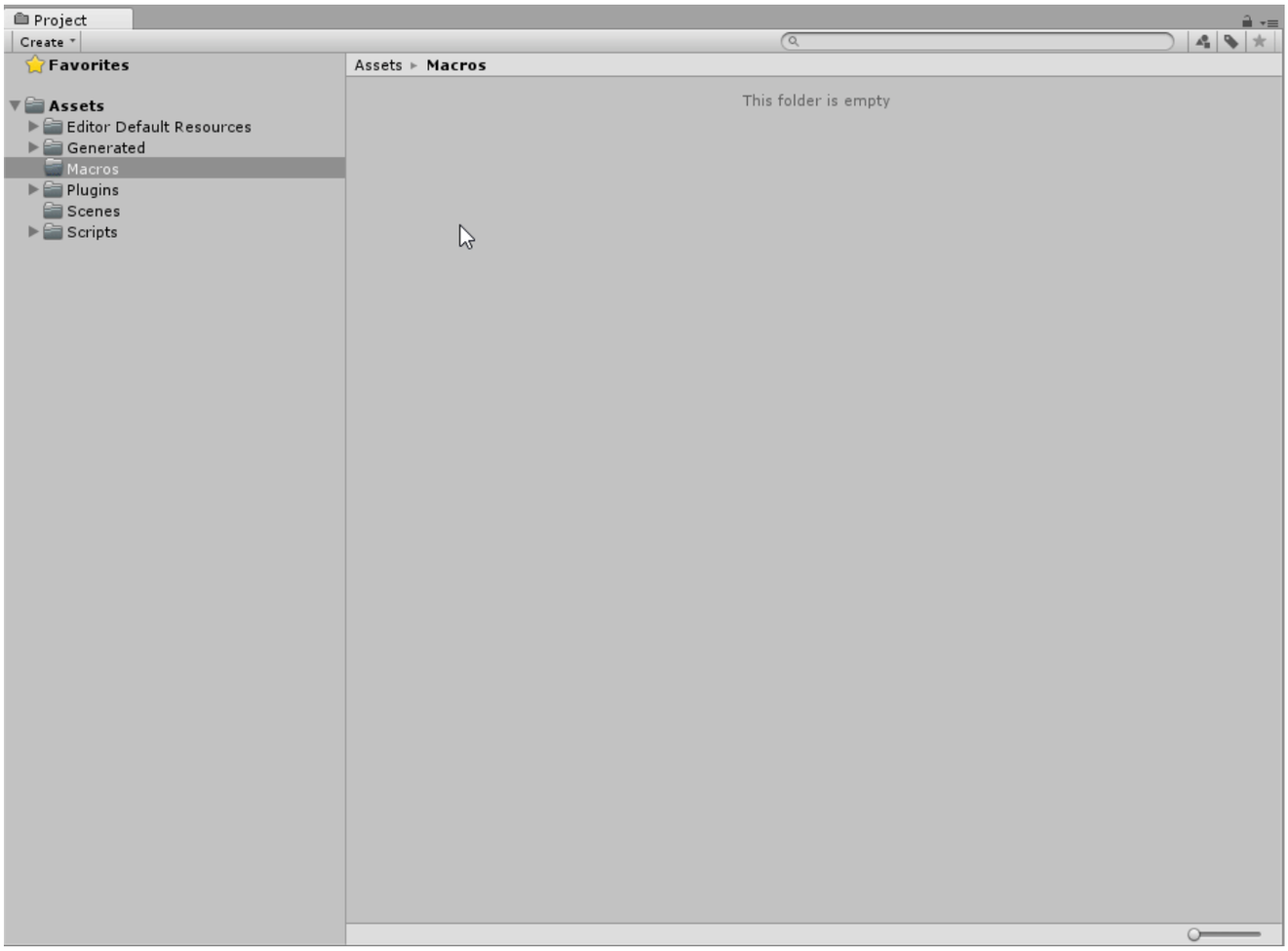
A **Macro** is a re-usable graph that can be referenced by multiple different machines that have their Source set to Macro.

If you switch the Source option of our machine to Macro , you'll see that you now have to tell the machine *which* macro it should use:

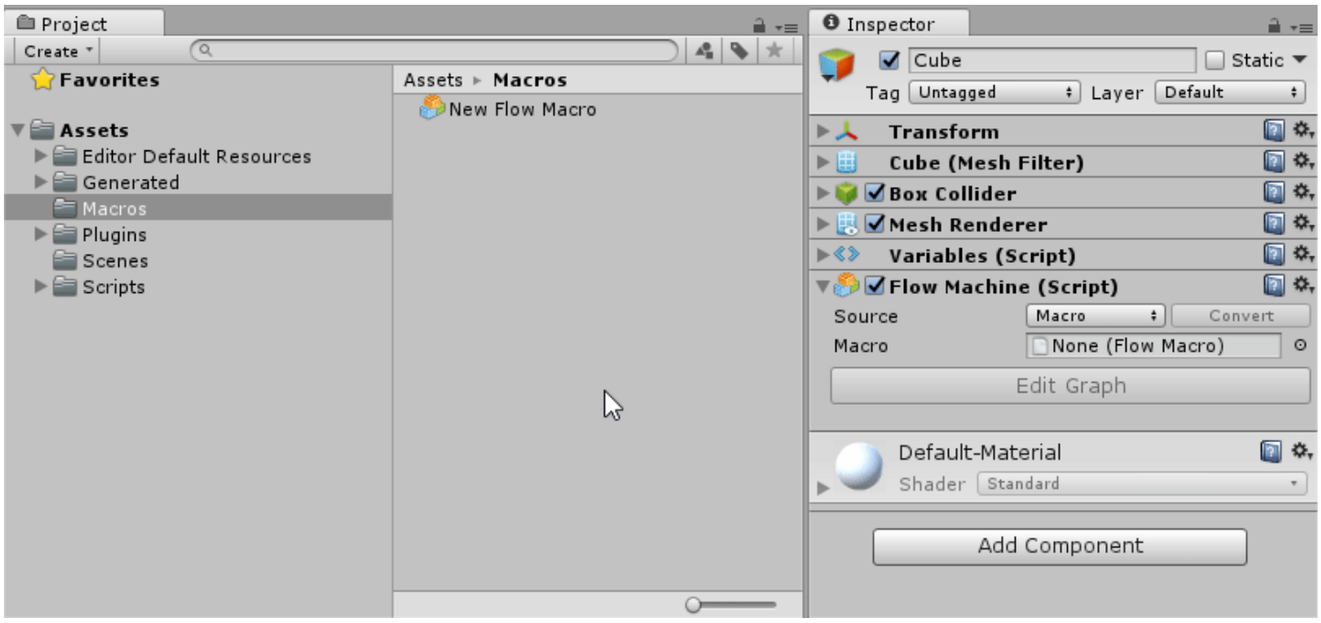


As you would expect, a Flow Machine requires a **Flow Macro** and a State Machine requires a **State Macro** .

To create a macro, right-click in an empty folder in your Project panel and choose Create > Bolt > (Flow / State) Macro . We recommend putting your macros in a top-level Macros folder, but that organization is entirely up to you and has no impact on functionality.



Then, you can simply drag-and-drop your new graph or use the Unity object picker to assign it to the machine:



Converting the Source

At first, you might find it a bit confusing to choose whether you should use a component or asset graph. Don't worry: Bolt makes it simple to change your mind at any moment. It's common and normal to convert from one kind of source to the other.

From Embed to Macro

For example, you might start working on a component graph for movement on an enemy game object, but then realize you'd want the same logic to apply to friendly NPC's. Therefore, you'll want to convert your component graph to an asset graph for re-usability.

To do so, simply click on the 

button and choose a path and file name for the new graph asset. Bolt will copy all the items in your component graph to the graph asset (except scene references, which are not supported in asset graphs). The machine will then automatically switch to asset mode and reference your new graph.

Note that your component graph isn't deleted in the process. You can always just switch the source back to component to return to it.

From Macro to Embed

For example, if you were using a shared graph for AI behaviour, but you then realize that this one enemy has special behaviour, you might want to convert your asset graph to a component graph to modify it independently from the others.

To do so, simply click on the 

button. A dialog will warn your that this conversion will **permanently overwrite** your current component graph, so make sure you're OK with this before moving on.

Prefabs

Bolt supports prefabs, but there are some caveats. There is an article on Prefabs (http://support.ludiq.io/topics/159-prefabs/) in the Advanced Topics section to fully understand the limitations.

To summarize, if you use a machine with a **embed** graph as a prefab, the edits you make on the prefab definition **will not be automatically propagated** to the prefab instance. In other words, it's OK to use any machine as a prefab, but if you instantiate a component machine prefab **in edit mode** (not during play mode), you'll have to manually use Revert to Prefab on each instance every time you make a change on the definition. In these cases, you should use a macro instead.

When to use each kind of graph

We're almost done with this basic concepts section, and soon you start learning about Flow Graphs (http://support.ludiq.io/forums/4-bolt/categories/41-flow-graphs/topics/) or State Graphs (http://support.ludiq.io/forums/4-bolt/categories/40-state-graphs/topics/). We will start with flow graphs, then move on to state graphs.

- You should use a **Flow Graph** for most of the so-called "low-level" logic, like branching, loops, math, etc. They are much more versatile than state graphs, but they don't benefit from the built-in notion of state. Flow graphs best answer the question "when this happens, what should I do, and in what order?". You can make an entire game with only flow graphs.
- You should use a **State Graph** when you want to create so-called "high-level" logic, like AI behaviours, scene or level structure, or anything that requires the notion of *state*. For example, an enemy that has a "patrol", "chase" and "attack" states, or a door that has "locked", "unlocked", and "open" states. State Graphs best answer the question "what is my current behaviour, and when should that change?".

Finally, remember that you can combine the two kinds of graphs. For example, each state node in a state graph is actually a flow graph. This will be discussed in detail in the Mixing It Up (http://support.ludiq.io/forums/4-bolt/categories/42-mixing-it-up/topics/) section of the manual.