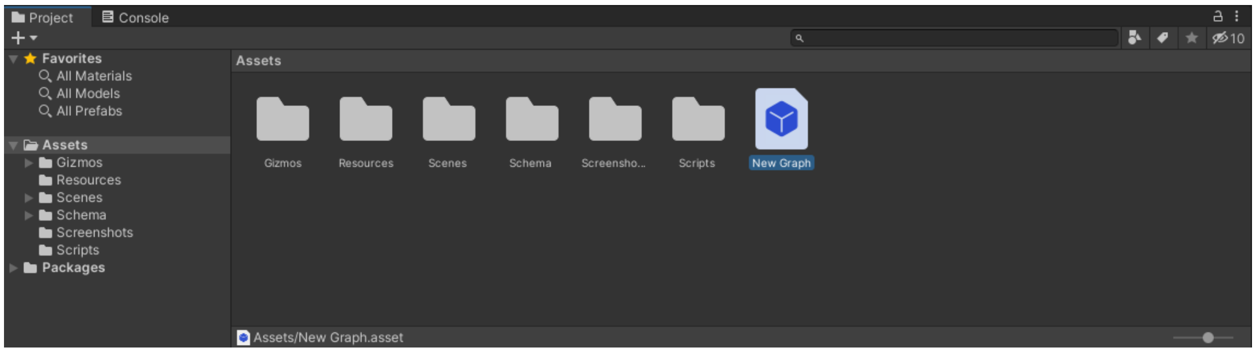
Schema

Behavior Tree Builder

Hey there, thanks for downloading Schema! This asset will help you build intelligent AI systems for video games in a visual node-based environment. This PDF will help you get started by creating a very basic behavior tree to get your footing in the editor and learning a little bit about how behavior trees work. They're a complex topic, so don't worry if it gets a little confusing at first. Over time, you'll get better and faster designing trees and spending less time coding complex AI and more time making your game.

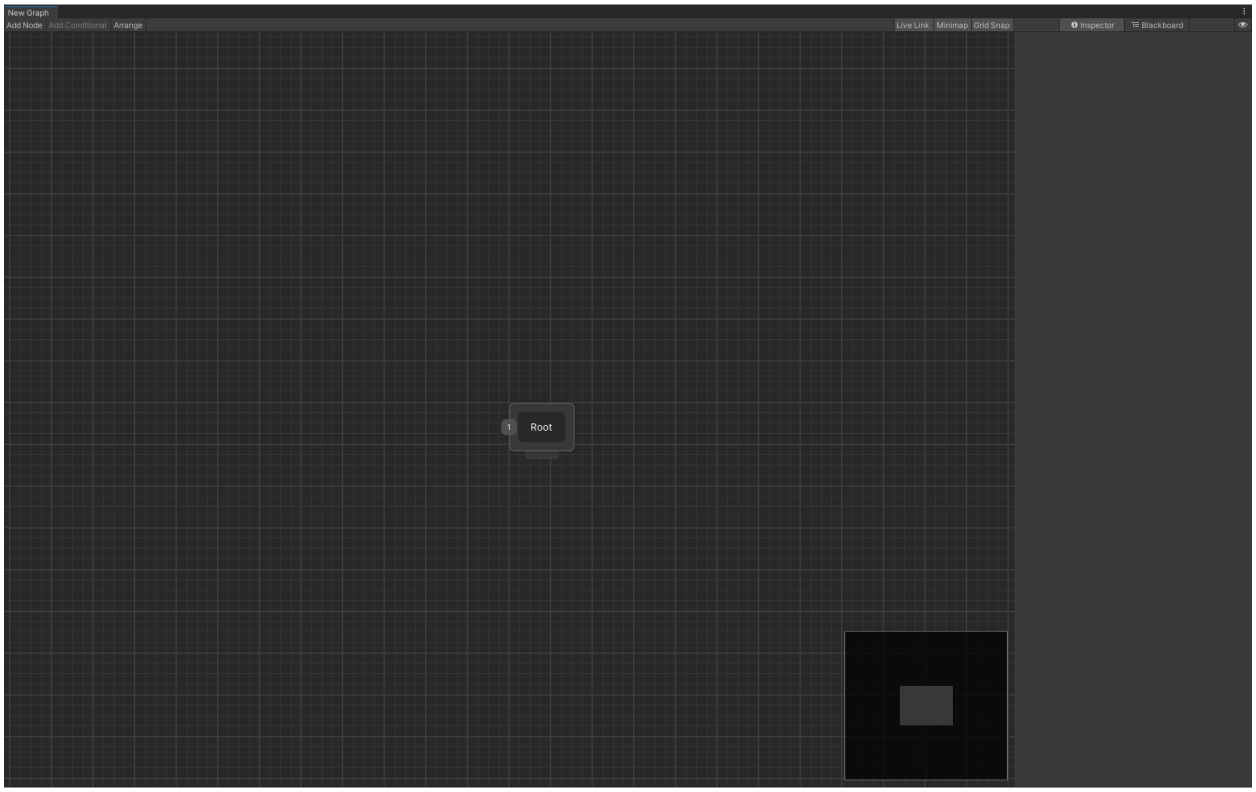
Let's create a basic behavior tree that logs the name of the GameObject every second. First, a quick introduction about how behavior trees are created. Behavior trees in Schema are stored in an asset, just like a Material or Prefab. These assets can be attached to Agents that actually execute the tree that you have made, with their own variables and context.

To get started, we need to create a behavior tree asset. To do this, right click anywhere in the Project window and select **Create > Schema > Behavior Tree**. This will create the asset we will edit. A new asset should appear in the project directory. Name it anything you want.



Our graph in the directory

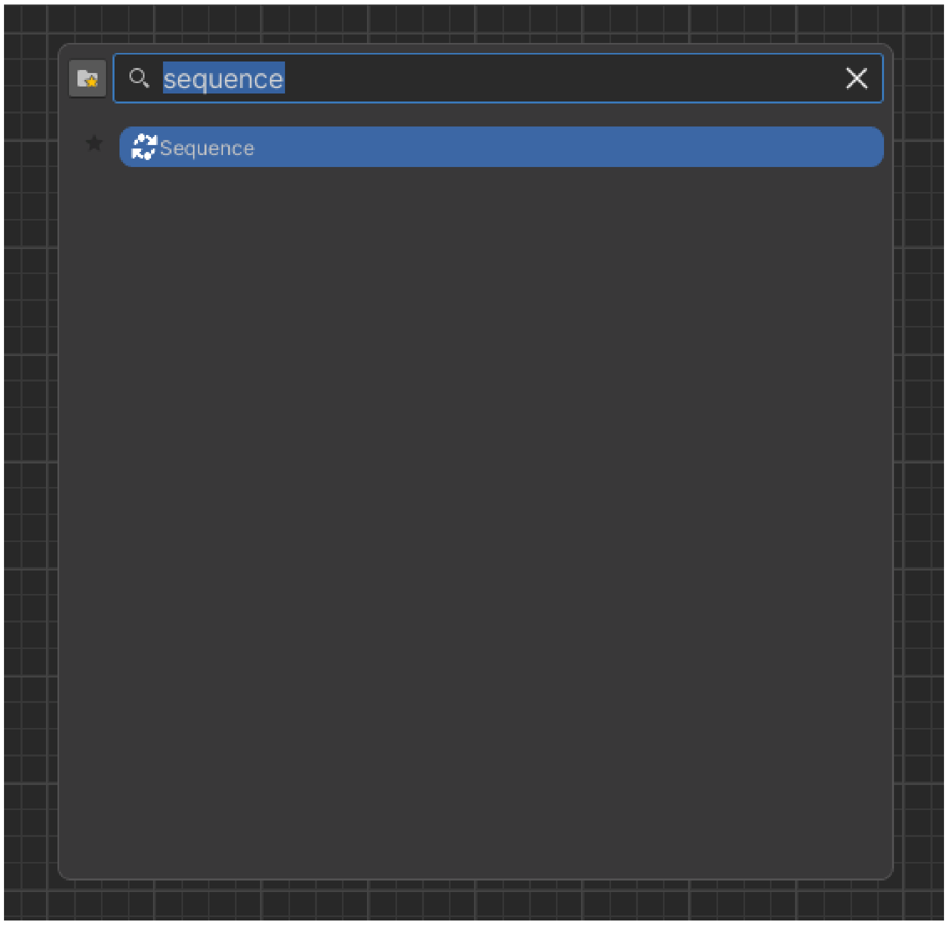
Double-click the new asset to open it in the Schema editor. This is where you will spend your time creating behavior trees. You will be greeted by a single node, **Root**. This node is where the execution of the tree begins.



The default editor view

Let’s create the first node in our tree, the **Sequence**. The Sequence node will execute all of its child nodes in order, until one of them fails. If they all succeed, it will also succeed. The Sequence node is called a **Flow** node, because it controls the flow of the behavior tree. There are several other Flow nodes in Schema, and you can also create your own.

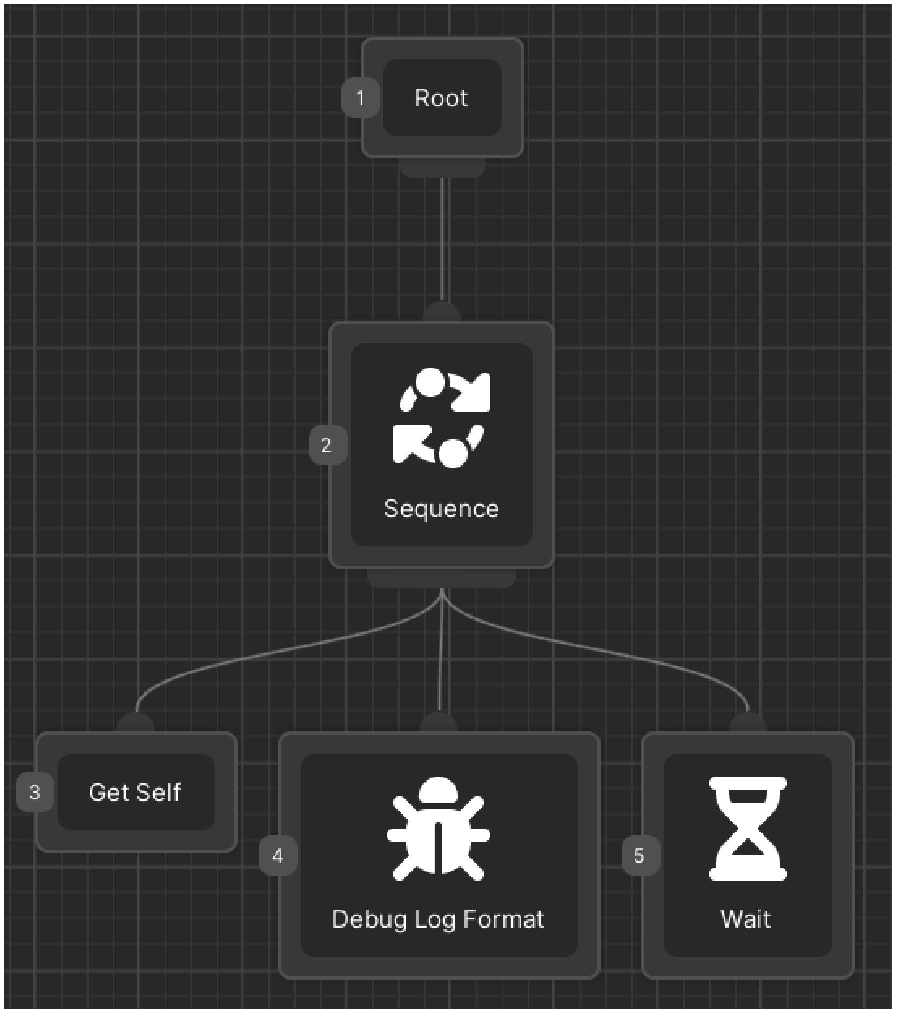
To attach a node to the Root node, click and drag the bottom nub and drop it anywhere. It will bring up a search menu. Search for “Sequence” and press Enter.



The Schema search menu

To add nodes to the Sequence, repeat the process above but with the bottom nub of the Sequence node. We'll need to add three **Action** nodes: **GetSelf**, **DebugLogFormat**, and **Wait**. Action nodes are where the tree does things. There are over a hundred actions present in Schema already, but chances are, you'll need to create your own. To learn how, visit the documentation.

You can move the nodes around to shuffle the order, or detach them from their parents by clicking on the top nub and releasing the connection somewhere else. Work on your tree until it looks like this:

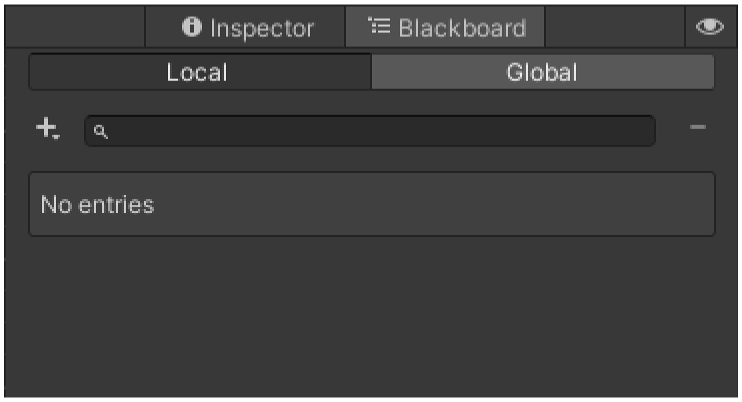


Our behavior tree

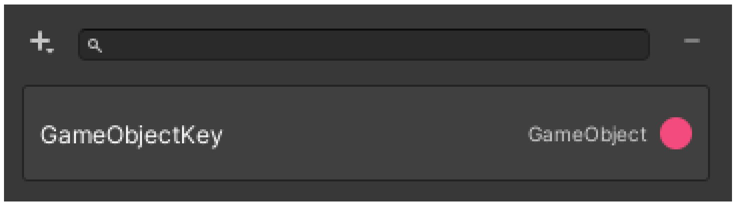
Quick tip! To make your trees look way better, click the “Arrange” button on the toolbar. It will algorithmically sort your nodes so they no longer look like spaghetti. This makes working through the logic of the trees much easier.

Now, for some configuration. We need to create a GameObject variable to store the GameObject whose name we want to log into the console. In this case, we want to log the name of the GameObject executing this tree, hence the GetSelf node.

To create a variable, click the “Blackboard” tab on the top right inspector menu.



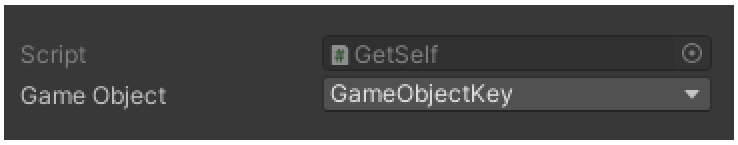
Create a new variable by selecting the plus icon and clicking the “GameObject” option. This variable can now be used throughout our tree to store the value of the GameObject.



Created GameObject variable

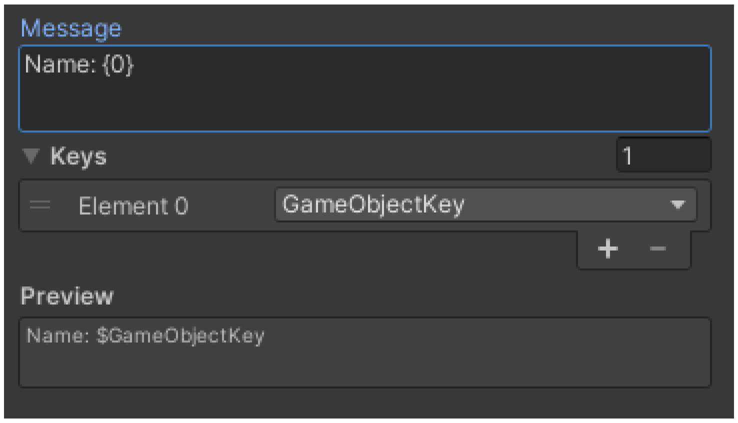
Click on the Inspector tab to return to it. The inspector functions exactly like the Unity inspector, and is used to modify values for the nodes and conditionals in the tree.

Select the GetSelf node by left clicking and change its Game Object variable to the one we just created. The GetSelf node will now store the value of the GameObject executing this tree in that variable.



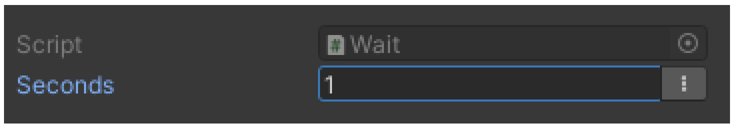
Script: GetSelf
Game Object: GameObjectKey

Now, select the DebugLogFormat node and change its inspector values to match the following:


Message
Name: {0}
Keys 1
Element 0 GameObjectKey
Preview
Name: $GameObjectKey

The DebugLogFormat node is a wrapper for the Debug.LogFormat method built into Unity. It will interpolate the values of values selected in the list into the string that you provide at the top. This node will log the name of the GameObject variable to the console.

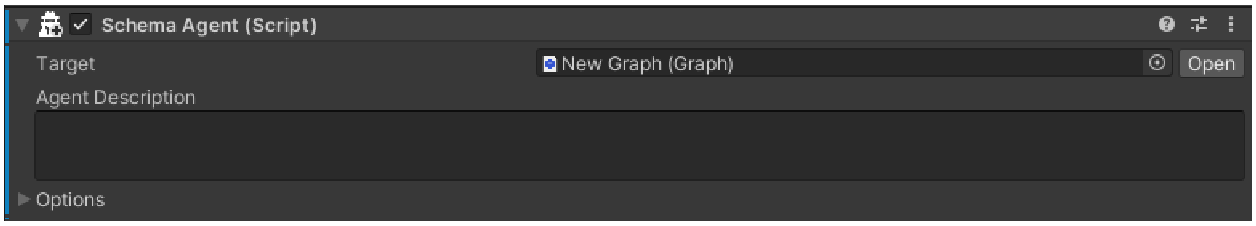
Finally, select the Wait node to change the duration of the pause. The Wait node will simply wait a given number of seconds before resuming execution of the tree.


Script Wait
Seconds 1

You may notice the button to the right of the field. This allows you to use a blackboard value instead of a constant. This is incredibly useful for creating dynamic trees, and even supports properties of another variable. For example, you could Wait for the same amount as the object’s x position. There’s tons of ways you can use this, and most nodes will support this functionality.

With that, our behavior tree should be complete. Although it contains all of the information for Schema to execute the tree, we still need to attach it to a GameObject that acts as an agent that executes this tree.

Select a GameObject in your scene and add the “SchemaAgent” MonoBehaviour to it. This Component is what actually executes your tree. The inspector looks like this:



Attach your tree to the GameObject field, and hit play. If you did everything right, you should see the GameObject executing the tree logged to the console every second.

And those are the basics of using Schema! Things not covered in this guide include conditional execution, aborting executing nodes, modifiers, and dynamic variables. To learn about these concepts, you can visit the documentation.

If you need to get in touch, contact me here: support@schema-ai.com or visit the Discord.