nodeCanvas

For more thorough documentation please visit

www.nodecanvas.com

What is it...

NodeCanvas is a complete visual node Behaviour authoring solution for creating dynamic and interesting behaviours using the power of Behaviour Trees and Hierarchical State Machines with ease and offers much power to both designers and programmers to work together in an efficient and flexible way.

In this Quick Start guide we will just go through creating a very simple Behaviour Tree just for the shake of explaining some key concepts. By no means this is a real tutorial.

Let's Start...

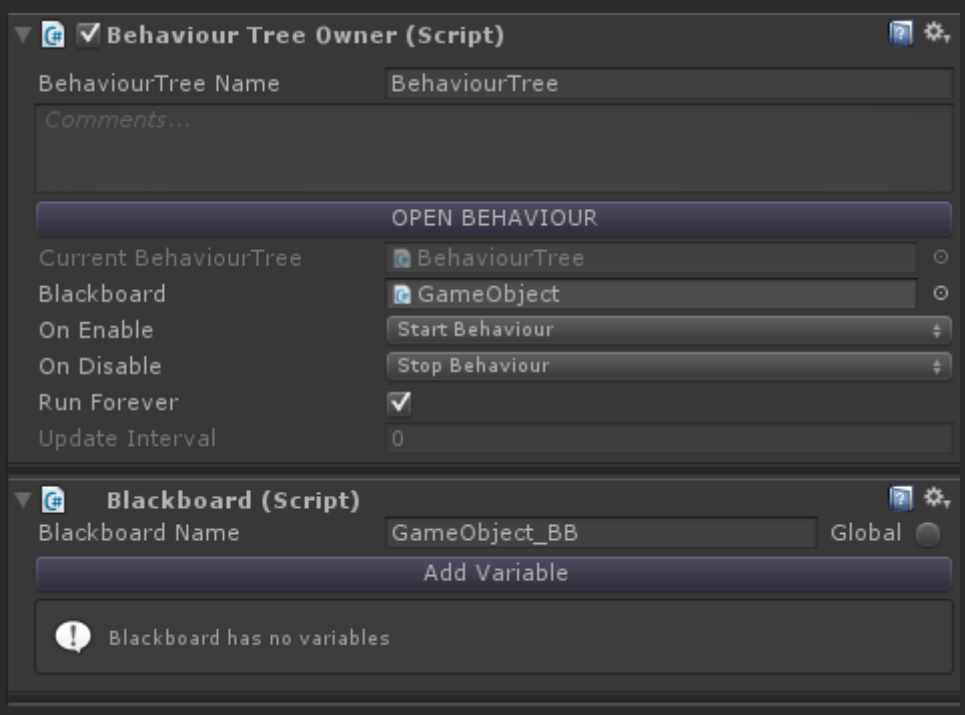
Place the 'Behaviour Tree Owner' Component on any game object you'd like to behave based on a Behaviour Tree system. The BehaviourTreeOwner is responsible for executing a Behaviour Tree and is not the Behaviour Tree itself.

As such, it must be assigned a Behaviour Tree. Click 'Create New' on that added component to create and assign one automatically. You will be prompted with an option for "Local" or "Asset" graph:

- **Local Graph** is a local instance which is bind to and saved with the scene. A prefab game object can not have such type.
- **Asset Graph** is an asset file, which can be reused amongst any other BehaviourTreeOwner.

Notice that you have the options to convert from one type to the other at any time.

Working with Graph Asset is recommended and works very similar to how Mecanim works in unity, where you add an Animator component and assign an Animator Controller. Only in this case, the Animator is our BehaviourTreeOwner and the Controller is our assigned Behaviour Tree asset.

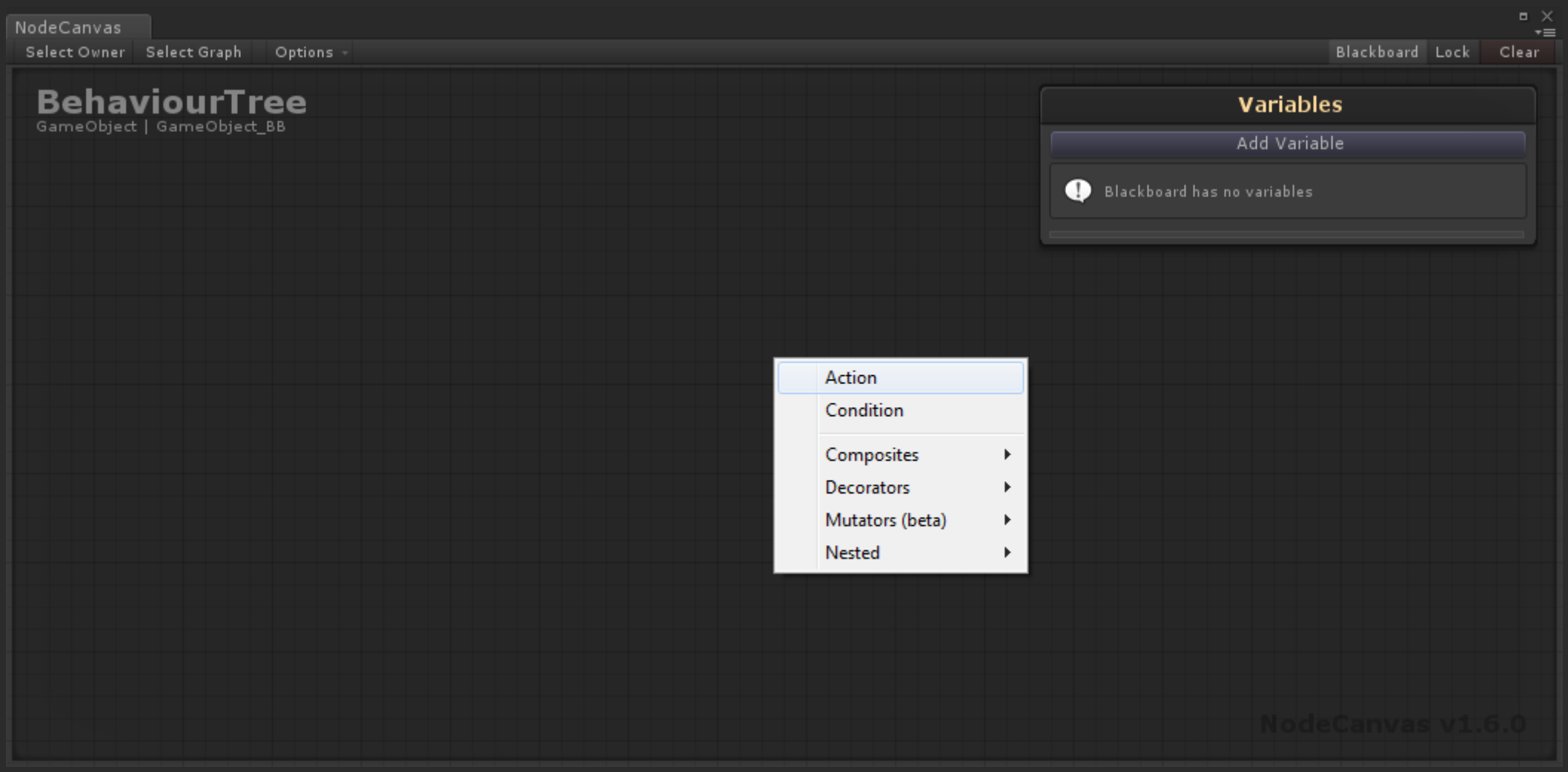


You will notice that a Blackboard Component is also automatically been added when the BehaviourTreeOwner was added. Blackboards are used to store variables which can be accessed from any action and condition tasks within the system to communicate data between one another, as well as from any of your own scripts. Again, very similar to Mecanim parameters, only in this case the parameters/variables are stores within a separate component. You can assign another blackboard if you like say for example to have a common blackboard between two or more different BehaviourTreeOwners, which is very desirable in some situations.

Since Blackboards are Component and you might be working with Asset Graphs, they prove very use full for parametrizing the Graph and having the ability to use and reference scene objects.

Last but not least, you can select an option for what happens OnEnable as well as what happens OnDisable. By default, OnEnable the behaviour will start and OnDisable the behaviour will stop. Furthermore, the 'Run Forever' option means that the behaviour tree will repeat when it is done, while the 'Update Interval' setting specifies how often the Behaviour Tree will be Ticked.

Leaving everything as is, click 'OPEN' to edit the Behaviour Tree.



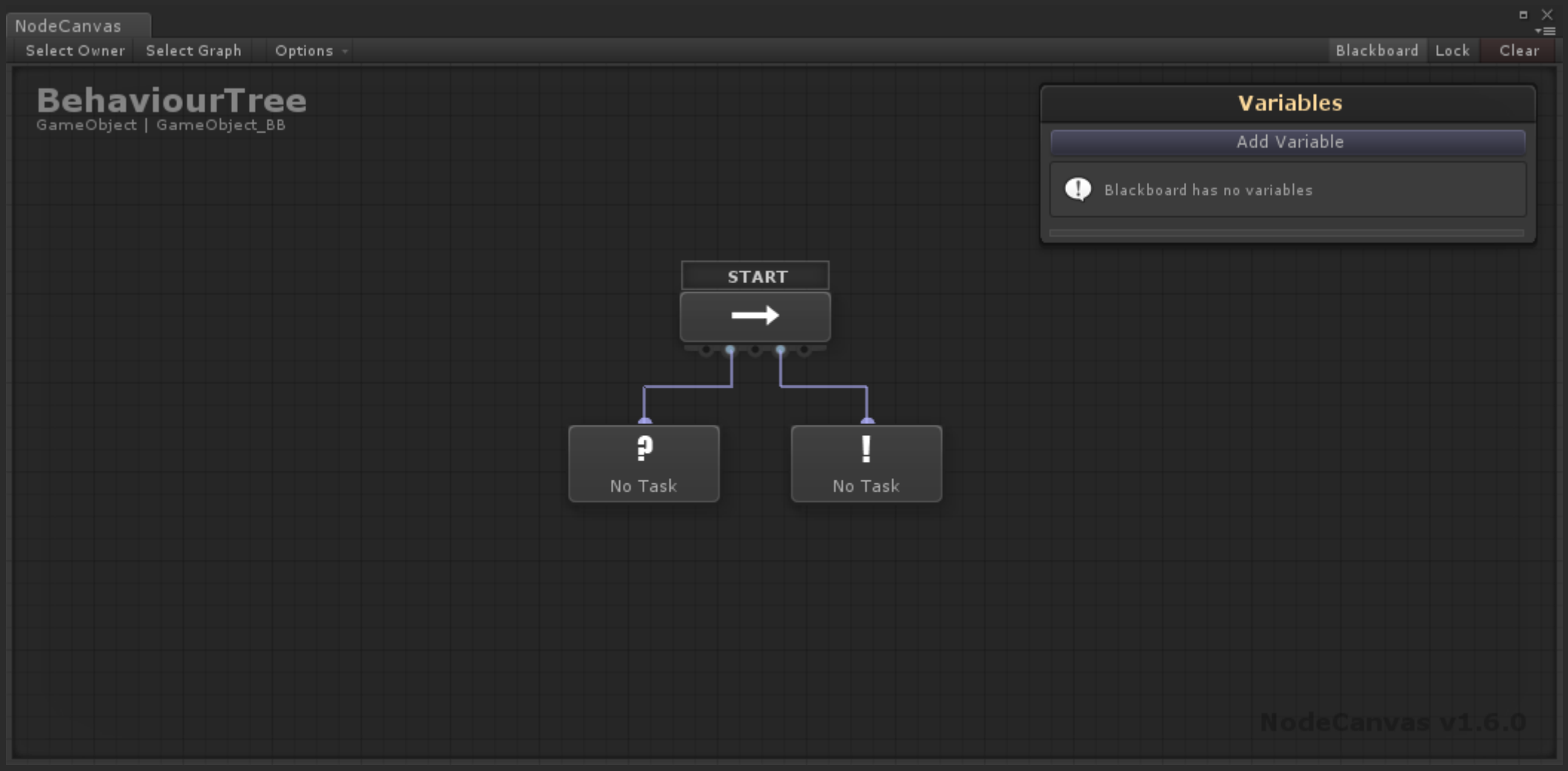
Right clicking on the canvas, will show a menu of all available nodes for this system (in this case Behaviour Tree). Selecting any of them, will add it in the canvas and at the mouse position.

You will see that the first node been added to the canvas is marked as **Start**. That means that it will be the first node to be executed; the entry point. You can specify another if you like by right clicking on a node and selecting 'Set Start'.

Before we continue, here are the basic editor controls:

- To connect nodes together click and drag from an empty node port and release on top of the target node.
- You can disconnect nodes by right clicking on a connected port.
- You can delete nodes or connections after selecting them, by hitting 'Delete'.
- You can pan all nodes around by middle click and dragging.
- **You can pan a node and all of it's children together by holding Shift and dragging a node.**

Note that Behaviour Tree nodes will get auto sorted depending on their position to the parent, from left to right.



So here we've added a **Sequencer** as well as a **Condition** and an **Action** node. You will notice that when adding an Action or a Condition node we don't specify beforehand the actual action or condition that will take place, but instead we simply adding a placeholder wrapper for an action or a condition. So we are free to design the behaviour irrelevantly of whether or not an action or condition has been implemented yet and be very modular about it. This is very powerful in the long run.

So these two nodes get to be assigned an actual Action and Condition respectively, collectively known as Tasks.

By pressing 'Add Action Task' (or Condition Task) a context menu of all available Action Tasks in the project will show up categorized respectively. When a task has been added, the node will read it's summary information as of what it will do or check, as well as the node inspector panel will now show that Task's controls from there after.

Lets take a closer look at the node Inspector now and in regards to the current Action Task assigned.

First of all, we can simply remove the Action Task and assign another without deleting the node. Simply press that "X" button and the "Add Action Task" button will show up again for the possibility to add another. Also clicking on the script Icon on the far right will open your IDE to edit the Task script.

Next comes the 'Target Agent' control which also reads the required agent Type in parentheses, as well as it has a toggle control on the far right. This tells us that for this action to execute correctly, the agent's game object needs to have that type of component (or a component with the required interface in some cases). By default the 'Target Agent' is set to 'Use Self', which refers to our BehaviourTreeOwner we've added before.

Clicking on the toggle control on the far right, you can Override the Target Agent for this Task, so instead of using Self it can be set to either use a direct assigned reference, or even better be set to read from a Blackboard Variable.

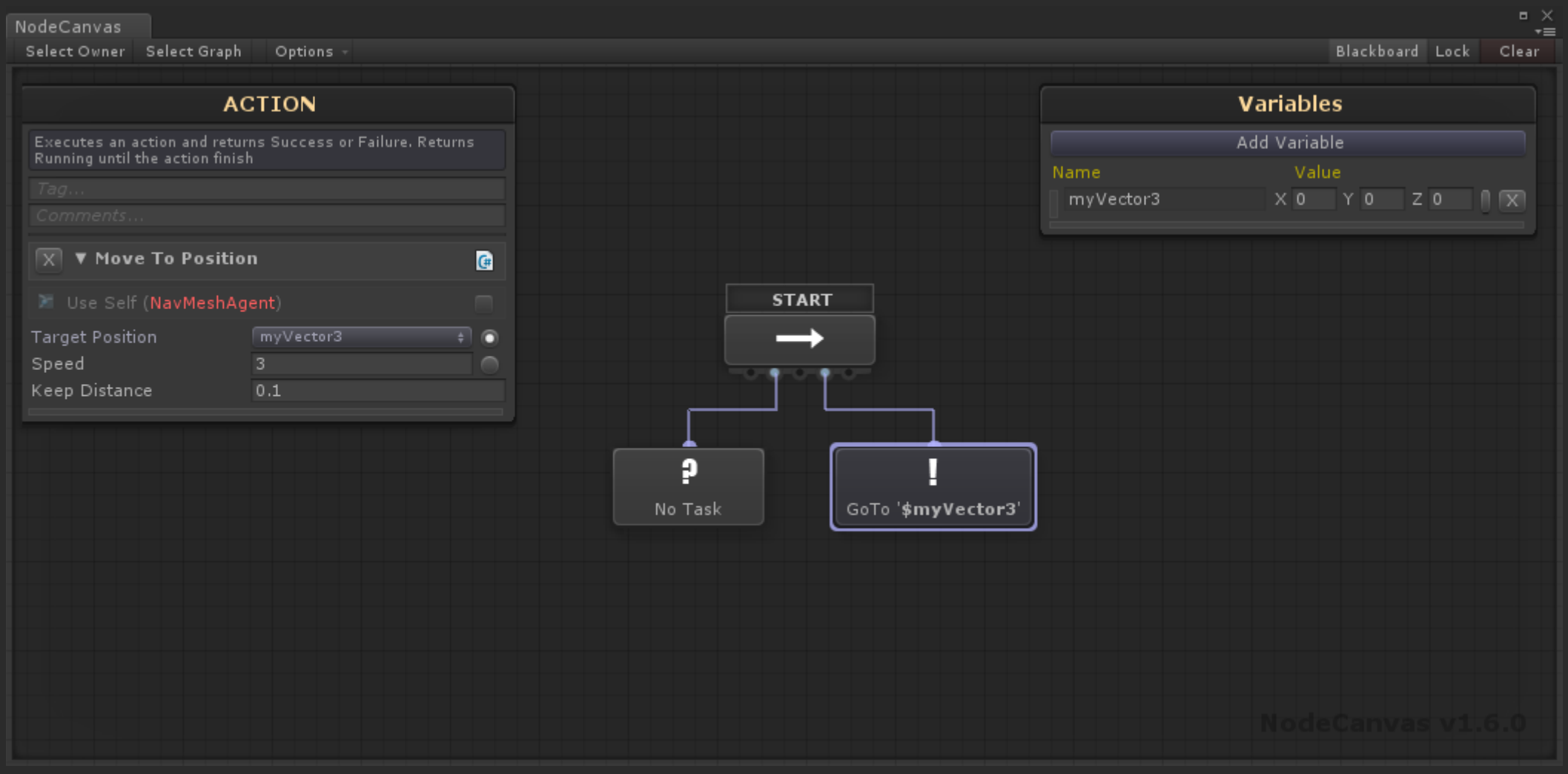
Next come the settings for the actual action. In this case the target position for the agent to move, the speed at which to move and the distance to keep from the target position. The first two of the settings have this radio button on the far right.

Whenever that radio button is shown, it means that it is possible to select a value from the blackboard variables instead of directly providing one. Pressing the radio button for the Target Position will turn the controls into a drop down variable selection.

The drop down (currently reading |NONE|) will show all available variables on the blackboard and of the type required by the specific property. Currently we have no such variables on the blackboard so lets create one.

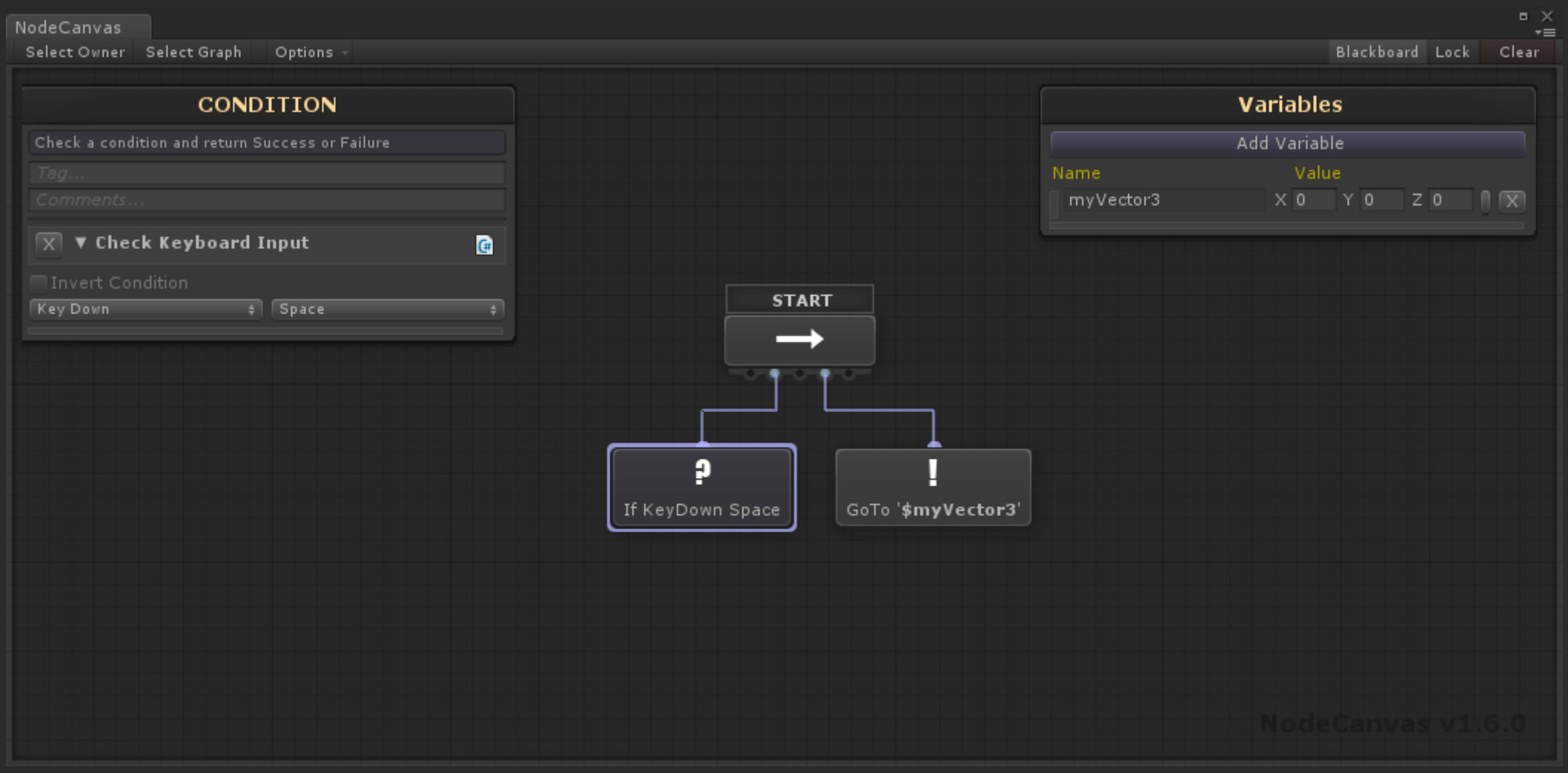
We can either go back to our BehaviourTreeOwner by pressing 'Select Owner' on the top left of the editor, or click the 'Blackboard' button on the top of the editor toolbar to show the blackboard reference within the canvas. In either case, clicking 'Add Variable' in the Blackboard inspector and selecting Vector3 in the popup menu, will add such a variable in the blackboard. We can (and should) of course specify a name as well.

Back to our action settings, the variable will now show up to be selected. You will also notice that the node will now read that selected variable's name instead of the actual value, but with an '$' symbol on the start and in **bold** to designate that this is a blackboard variable assignment, so that everything is instantly readable.

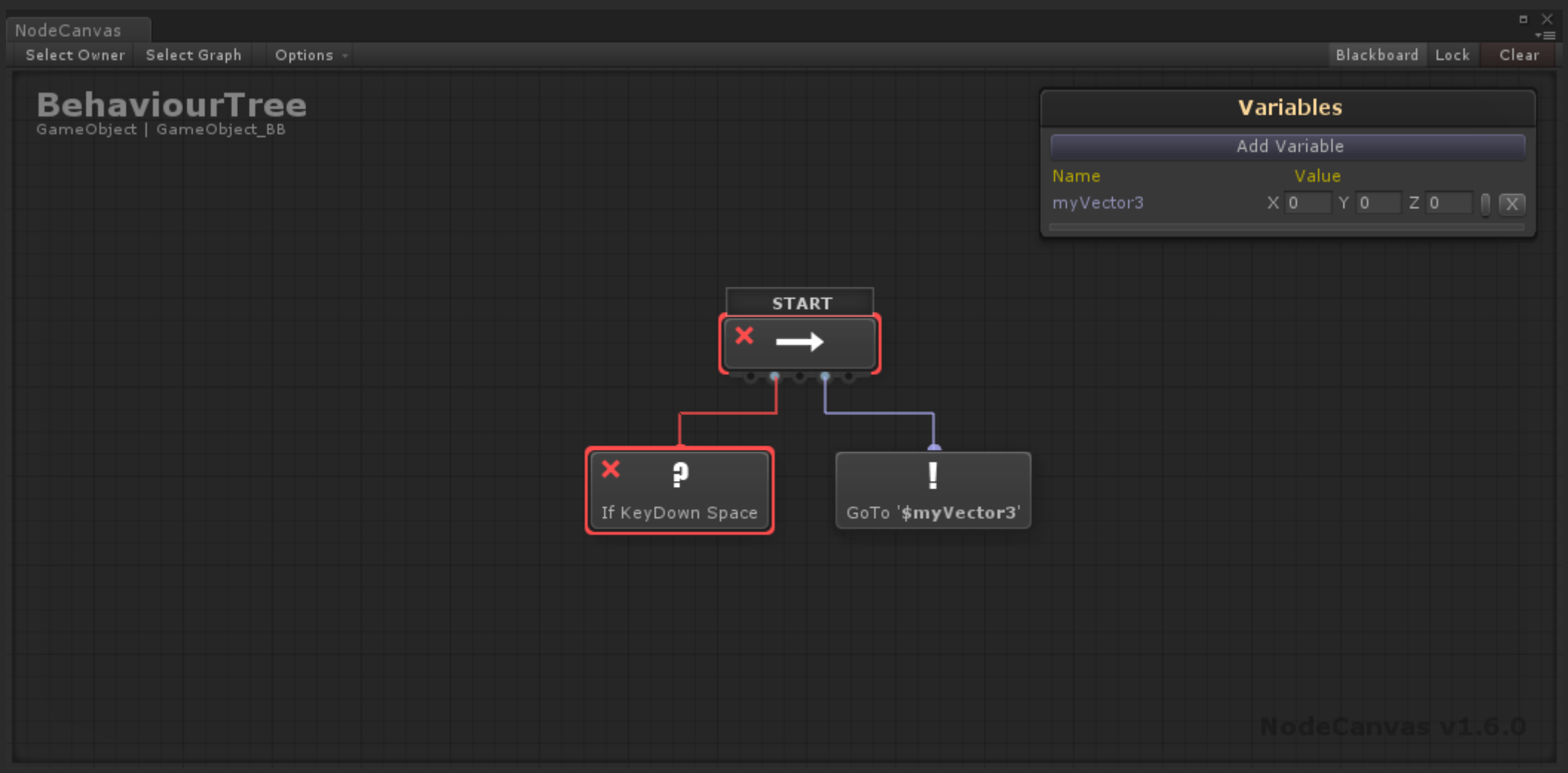


For the shake of this demo, lets add a 'Check Keyboard' Condition Task to the condition node, so that if we press a key, then the action will happen.

We do this in the very same way we've added the action. Select the Condition node, click 'Add Condition Task' and select the Check Keyboard condition under the Input category.



We can now hit play and watch it happen while visually debugging the Behaviour Tree as well:



You will notice that the nodes will now change colours based on their return status along with an icon that represents that status, as well as the connection colours which also represent that status. All of them together creating an instant feedback as of what, when and why something happens.

- So in this example, the Sequencer will check the first child as it should, which will return Failure (red) since the space key is not pressed down.
- As soon as the space key is pressed down, the Condition will return Success (green) and as such the Sequencer will continue to the next child as it should.
- Since the Action 'Move To Position' requires some time to complete, it will enter a running (yellow) state until it is finished at which point it will return Success and the Behaviour Tree will reset and repeat (considering that the Run Forever option is checked on the BehaviourTreeOwner as described in the start of this document)

That's it for this very brief Quick Start to NodeCanvas and Behaviour Trees, but of course there are so much more!!
Please do visit www.nodecanvas.com to read much more thorough documentation, node reference, learn how to create you own Actions and Conditions as well as how to use any other NodeCanvas Module System.
Please understand that this is a very brief quick start guide.