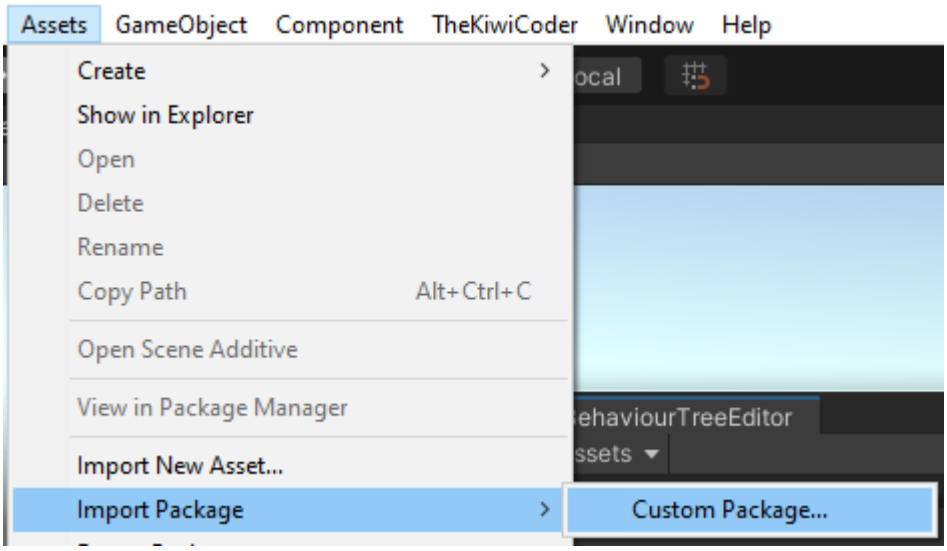
Behaviour Tree Editor

Table of Contents

Installing the package

After downloading the zip file, import the package from the assets menu: `Assets->ImportPackage->CustomPackage` and select the behaviourtree.unitypackage file.

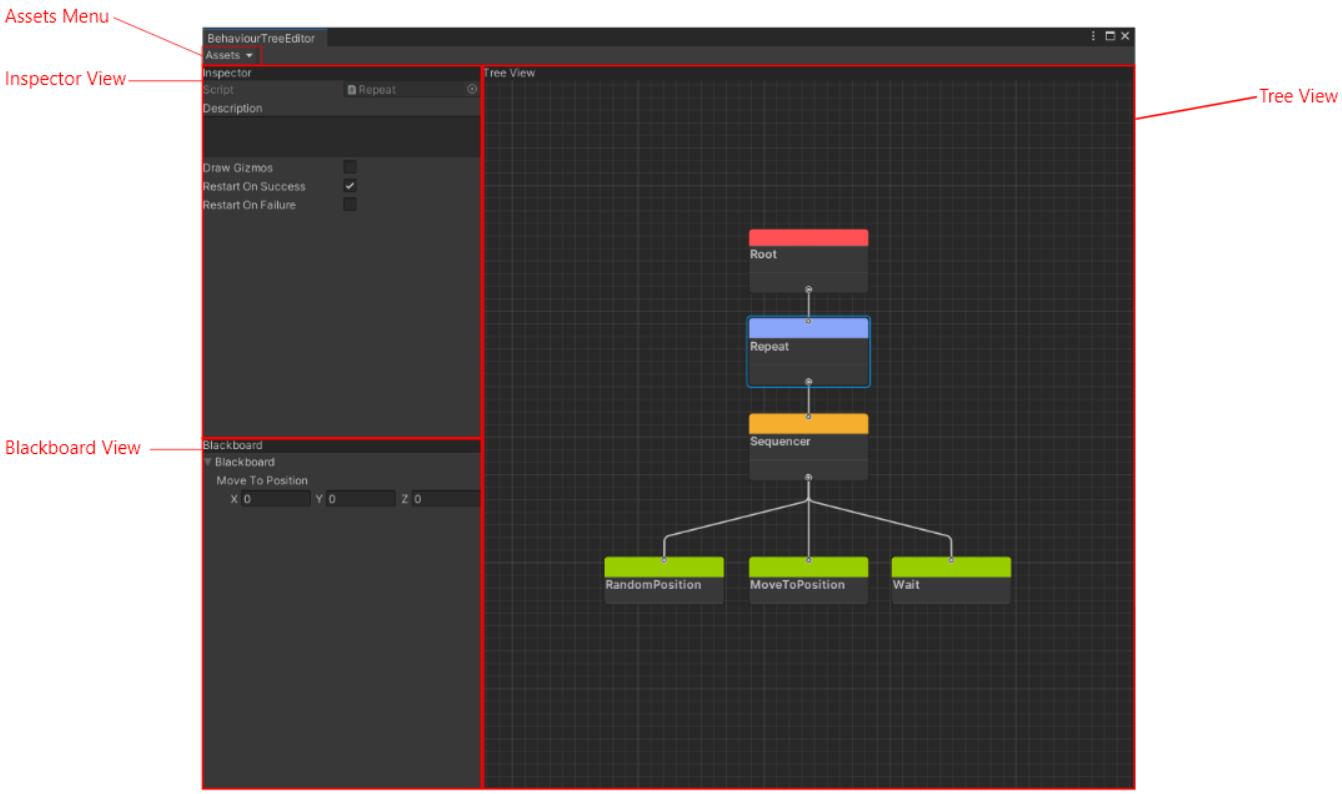


All files will be installed to `Assets/TheKiwiCoder/BehaviourTree`

An example scene and behaviour tree can be found at `Assets/TheKiwiCoder/BehaviourTree/Example`

> ***NOTE:*** The version used for testing at time of writing is Unity 2020.3.13f1 LTS.

Overview



Behaviour trees are great at creating both simple and complex ai behaviours. One of the main advantages of behaviour trees is their ability to breakdown complex behaviours into very simple units called Actions, which can then be combined, reused, and rearranged in various ways to create new behaviours.

A behaviour tree executes top to bottom starting at the root node, and traversing down into it's children. Each node in the tree can return one of three states; failure, success, or running. It is up to the parent node to decide how to interpret these return states, and decide what to do next if a failure occurs.

Action Nodes

Action nodes make up the bulk of the tree. They are always at the leaf of the tree, and have no children. These nodes are responsible for changing the state of the world, and ultimately represent the agent's capabilities. Anything an agent can do should be represented as an action. You will create many new types of actions to represent the behaviours that are specific to the agents in your game.

The granularity of actions can either be large or small, but it's often a good idea to keep the actions as small as possible to increase reuse across the tree. Once you have built a decent library of actions for your AI they can be composed together to create complex ai behaviours.

Some sample action node types included in the package are:

- Log - Logs a message to the console
- Wait - Waits for a period of time
- Breakpoint - Pauses the editor when executed
- RandomFailure - Returns failure to it's parent with a random change value between 0 and 1
- RandomPosition - Generates a random position within a min max range and assigns the value to the blackboard.
- MoveToPosition - Reads a position from the blackboard, and sets the navmesh agent destination to this position.

Composite Nodes

Composite nodes represent the control flow of the tree, and can have many children. It's up to each composite node type to decide which order the children should be executed in. Composite nodes also monitor the failure state of it's children. There are many ways to handle failures and execution order, and this is what differentiates each composite node from another. The two main composite node types that reoccur in behaviour trees are sequencer and selector (a.k.a fallback), however many other composite node types are usually necessary to create complex logic. One example is the parallel node which is capable of executing multiple children at once.

The composite node types included in the package are:

Composite Node Type	Execution Order	Success	Failure

Composite Node Type	Execution Order	Success	Failure
Sequencer	Executes children one at a time left to right	When all children return success	When one child returns failure
Selector	Executes children one at a time left to right	When one child returns success	When all children return failure
Random Selector	Randomly selects one child to execute	When the child returns success	When the child returns failure
Parallel	Executes all children 'at once' concurrently. Multiple children can be in the running state at the same time.	When all children return success	When one child returns failure. Remaining children are aborted.
Interrupt Selector	Similar to Selector, but children are constantly reevaluated each tick. If a child with higher priority succeeds, the current running child is aborted.	When all children return success	When one child returns failure

Decorator Nodes

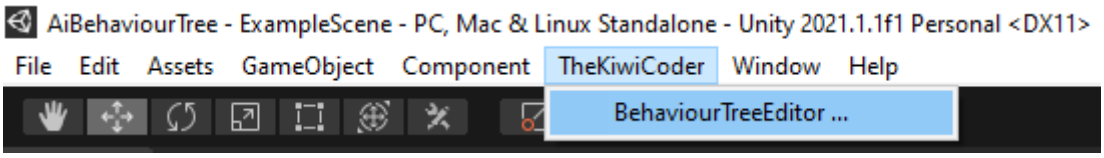
Decorator nodes have exactly one child, and are used hide or modify the return state of it's child. For example they are very useful to flip a childs return status, similar the the '!' operator in most programming languages. However they are also able to have complex logic inside them such as restarting a child when it returns failure a number of times. Or aborting a child that has been running for too long.

The Decorator node types included in the package are:

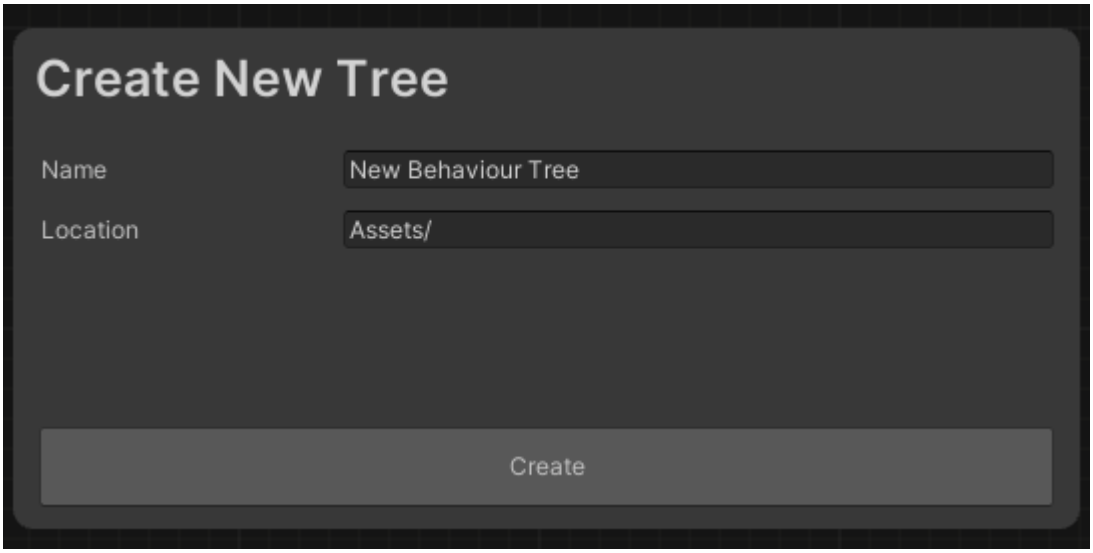
- Failure - Always returns failure to the parent
- Succeed - Always returns success to the parent
- Inverter - Inverts the childs return state from success to failure and vice versa
- Repeat - If it's child fails, it will try and again, and return Running to it's parent.
- Timeout - Aborts it's child if it hasn't finished after a given timeout period.

The Editor

After installing the package, The BehaviourTreeEditor can be accessed via the menu



The first time opening up the behaviour tree editor a prompt will appear to create a new behaviour tree. Pick a name for the tree and select a location, then press Create. A Behaviour tree can also be created via the standard project window create menu.



Navigation

Nodes can be selected directory, or box selected and dragged around the canvas.

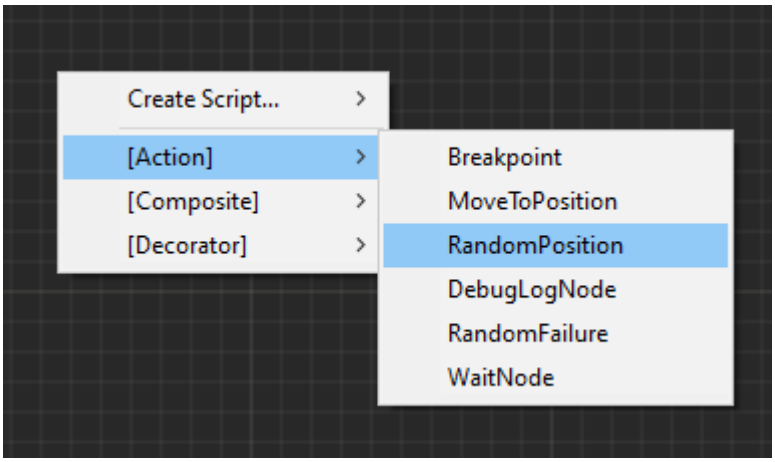
To select and entire `subtree`, double click on it's parent and the children will be added to the selection. Click in empty space to deselect again.

Keyboard Shortcuts:

KeyCode	Action
DEL	Deletes the currently selected nodes
A	Frames all nodes on the canvas
O	Frames the canvas origin
[	Frames the child node of the current selection

Adding New Nodes

New nodes can be added to the tree via the context menu by right clicking anywhere in the canvas and selecting which type of node to add. The nodes are grouped into three sub menus, Actions, Composites, and Decorators.

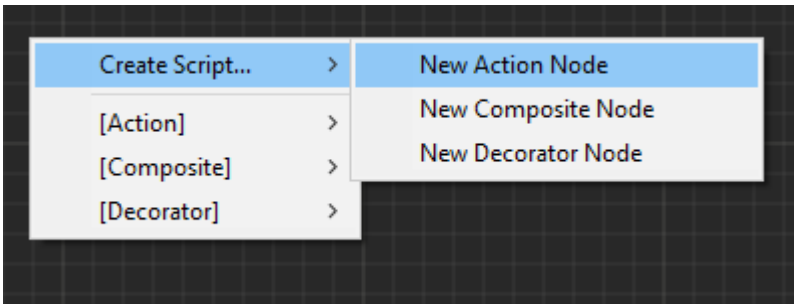


Creating New Nodes

The built in node types will only get you so far. The real power of behaviour trees comes when you design and create your own node types. Nodes are just standard c# scripts which inherit from one of three base types, `ActionNode`, `CompositeNode`, or `DecoratorNode`.

Creating a new script can be done by hand, and so long as it inherits from the previous base types mentioned, then they will automatically be picked up and registered by the editor and will appear in the context menu.

To speed up the process, new node scripts can be created from the context menu too. These use the script templates found at `Assets/TheKiwiCoder/BehaviourTree/ScriptTemplates`



Linking Nodes

To add a node as a child of another node, drag the `output` of the parent node to the `input` of the child node. Note only `CompositeNode` types can have multiple children.

Inspector View

The inspector view displays all public properties of the currently selected node. To display a node's properties in the inspector, be sure to select the middle of the node over the node's title. All nodes have a description field which can be set in the inspector. Just start typing in the description box and the text will appear under the node's title.

Blackboard View

The blackboard view displays all public properties of the blackboard. These values can be read and written to by individual nodes from any level of the tree. Note this blackboard is not generic, and it is expected to be modified to include the properties that make sense for your game. An example property exists for illustration purposes.

- MoveToPosition - Vector3 written to by the `RandomPosition` node, and read from by the `MoveToPosition` node

Assets Menu

The assets menu dropdown shows all behaviour trees located in the project. Use this to quickly jump between different trees in your project. There is an additional menu option to create a new behaviour tree from here too.

Settings Menu

There are various 'hardcoded' settings for the behaviour tree editor. These can be accessed via the standard project settings menu under the 'Behaviour Tree' category.