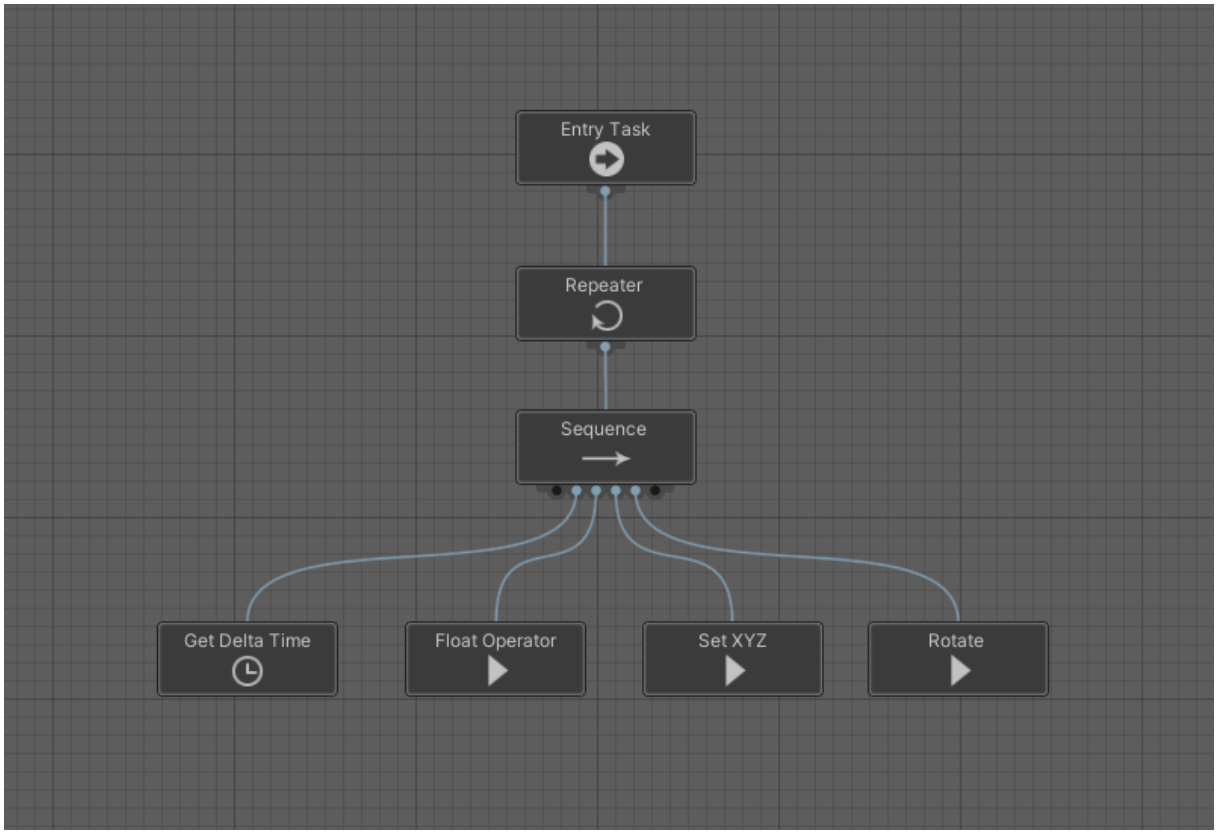
Behavior Trees

Getting Started

Behavior trees arise from the computer game industry as a powerful tool to model the behavior of non-player characters. They have been extensively used in high-profile video games such as Halo and Bioshock. Behavior Trees became popular by being able to create a complex behavior by only programming nodes and then designing a tree structure whose leaf nodes are actions and whose inner nodes determine the decision making.

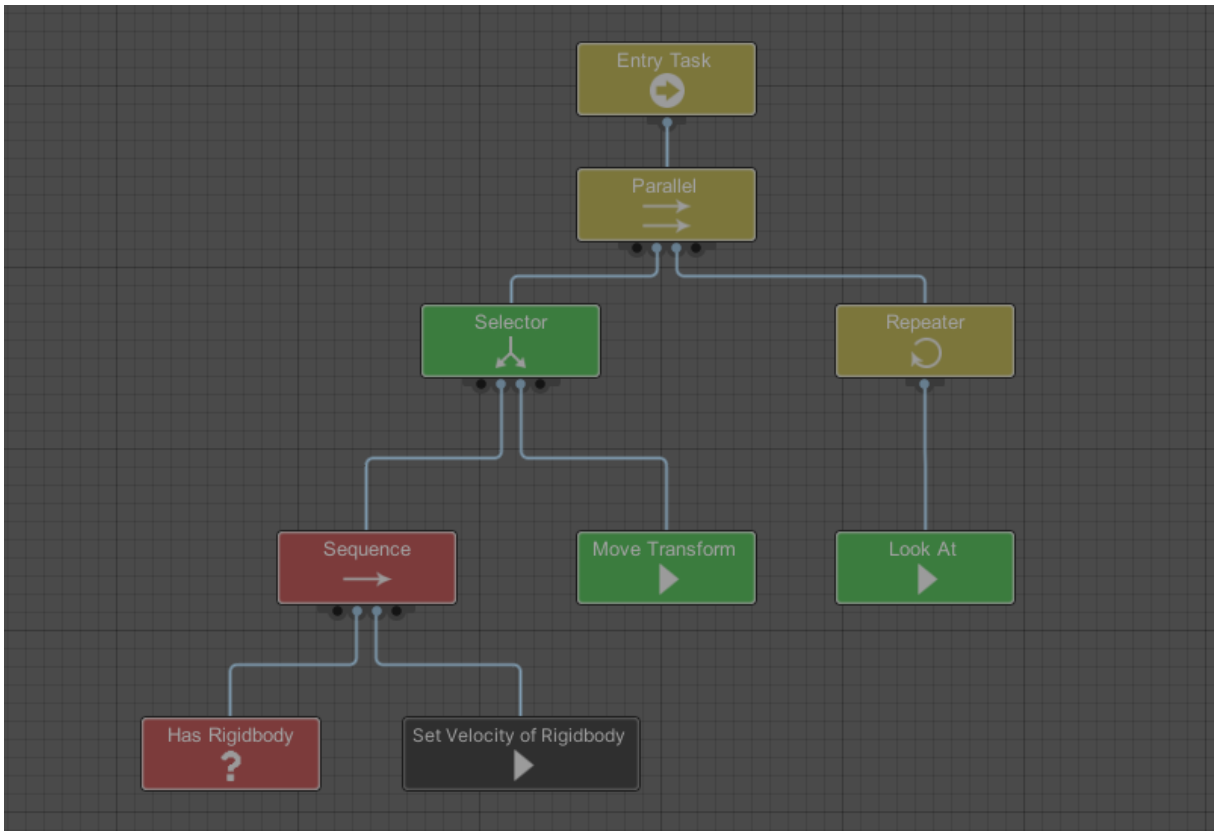
A behavior tree is graphically represented as a directed tree in which the nodes are classified as root, control flow nodes, or execution nodes. For each pair of connected nodes the outgoing node is called parent and the incoming node is called child. The root has no parent and exactly one child, the control flow nodes have one parent and at least one child, and the execution nodes have one parent and no children. Graphically, the children of a control flow node are placed below it, ordered from left to right.



Some core functionality is common to any type of node in a behavior tree. This is that they can return one of three statuses.

Success
Failure
Running

The first two, as their names suggest, inform their parent that their operation was a success or a failure. The third means that success or failure is not yet determined, and the node is still running. The node will be ticked again next time the tree is ticked, at which point it will again have the opportunity to succeed, fail or continue running. This functionality is the power of behavior trees, since it allows a node's processing to persist for many ticks of the game.



There are four diffrent types of nodes.

Actions
These are the lowest level node type, and are incapable of having any children. Actions are however the most powerful of node types, as these will be defined and implemented by your game to do the game specific or character specific actions required to make your tree actually do useful stuff.

Conditionals
Conditional nodes test some property of the game. For example in the tree

above it tests if the agent has a rigidbody component.

Composite
A composite node is a node that can have one or more children. The most commonly used composite node is the Sequence, which simply runs each child in sequence, returning failure at the point any of the children fail, and returning success if every child returned a successful status.

Decorators
A decorator node, like a composite node, can have a child node. Unlike a composite node, they can specifically only have a single child. Their function is either to transform the result they receive from their child node's status, to terminate the child, or repeat processing of the child, depending on the type of decorator node.

Installation

You can download and import **Behavior Trees** through the Module Manager. Please check the **Changelog and Dependencies** for version specific installation requirements. If you got it from the unity asset store, you are good to go.

Basic Node Types

Behavior nodes are largely self-contained. Relationships between behaviors are only stablished by explicitly adding them as children to composite a.k.a decider nodes. This rigid structure reduces coupling and therefore increases modularity and reuse of behaviors.

Actions and Conditions

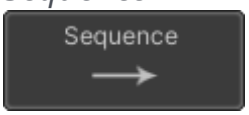
Action nodes, or actions for short, are the interface between behavior trees and the game world representation. Actions are leaf nodes of behavior trees. Action execution might change the behavior tree controlled agend and the game world state. When a ready action is visited during traversal it is activated and ticked. Some actions need multiple sumulation ticks until they terminate successfully or fail.

Actions that check if the actor or game is in a cetrain state are called conditions. Typically they finish during the same update tick in which they are activated without any side effects on the game state. Conditions query the game world state or agent state and return a success execution state if their condition check is fulfilled or fail otherwise.

Composites (Deciders)

Composite or decision nodes are inner nodes of the behavior tree and manage one or multiple child behaviors. When entered by the best first search through the behavior tree to update the agent, a decider chooses which of its child behaviors to traverse. As deciders react to the result execution states of their children, they can be understood to model the control flow of a behavior tree and schedule actions to tick.

Sequence



Sequence deciders run their child behaviors linearly, one after the other, as long as they complete successfully. When all children succeed, the sequence succeeds, too. If a child fails, then the whole sequence fails. In both cases, success or failure, the sequence node passes the execution state up to its parent node. If a child passes the running execution state to the sequence node it stores this child as the next one to visit and returns with a running state.

Selector

This composite pattern describes child nodes that are processed consecutively and in sequence only until one succeeds. As soon as one child node succeeds, the parent node succeeds immediately and stops processing child nodes. If all child nodes are attempted and all fail, the parent node fails. This pattern is useful for setting up AI agents to try multiple different tactics, or for creating fallback behaviors to handle unexpected outcomes.

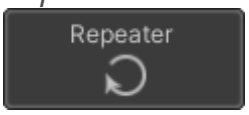
Parallel

Parallel or concurrent deciders tick all their children from left to right during each traversal.

Decorator

Decorator nodes only have a single child behavior. They represent some sort of functionality that can be added to another node and behaves regardless of how the other node works or what it does.

Repeater

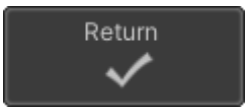


The repeater node will repeat execution of its children until a specified number of times.

Inverter

The inverter node will invert the return value of the child node after it has finished executing. If the child returns success, the inverter node will return failure. If the child returns failure, the inverter will return success.

Return



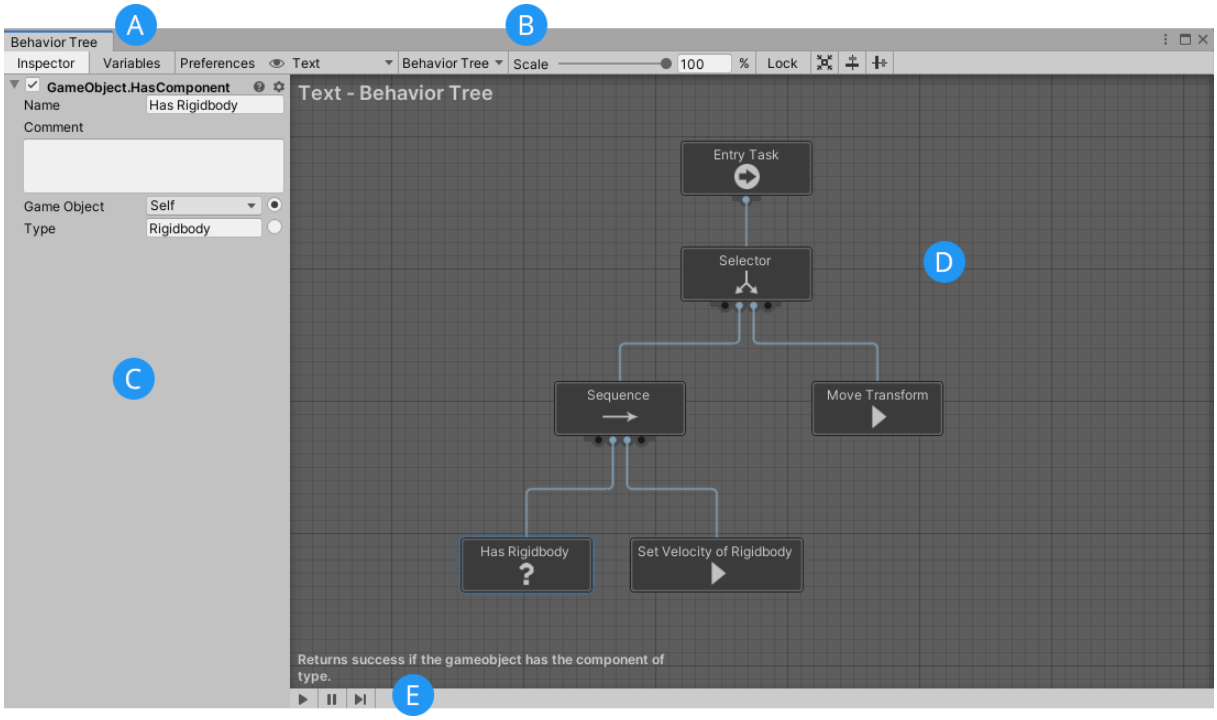
The return node will return the specified status regardless of the child status.

Until

Similar to the repeater node, the until node will repeat execution until the child returns a specified status.

Editor Overview

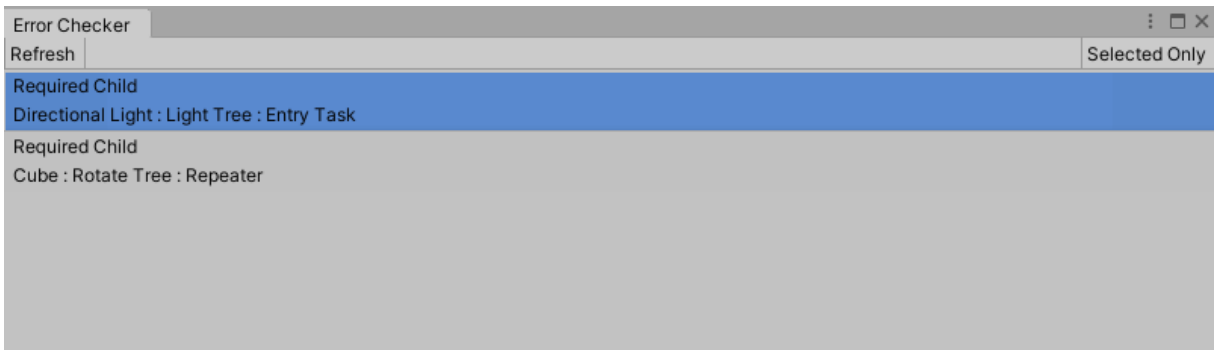
IIn the Unity Editor, you can access the Behavior Tree window through this menu: **Tools > Devion Games > Behavior Tree > Editor**.



- **(A) Inspector Toolbar**
  The Inspector toolbar allows you to switch between the inspector for nodes, variables and preferences.
- **(B) Operations Toolbar**
  The operations toolbar is mostly used for selecting behavior trees.
- **(C) Inspector**
- In the inspector you can configurate your behavior tree tasks, create new variables and setup preferences for the editor itself.
- **(D) Canvas**
  In the canvas you can create new tasks and arrange those tasks into a behavior tree.
- **(E) Bottom Toolbar**
  Quick access to start playing the unity editor.

Error Checher

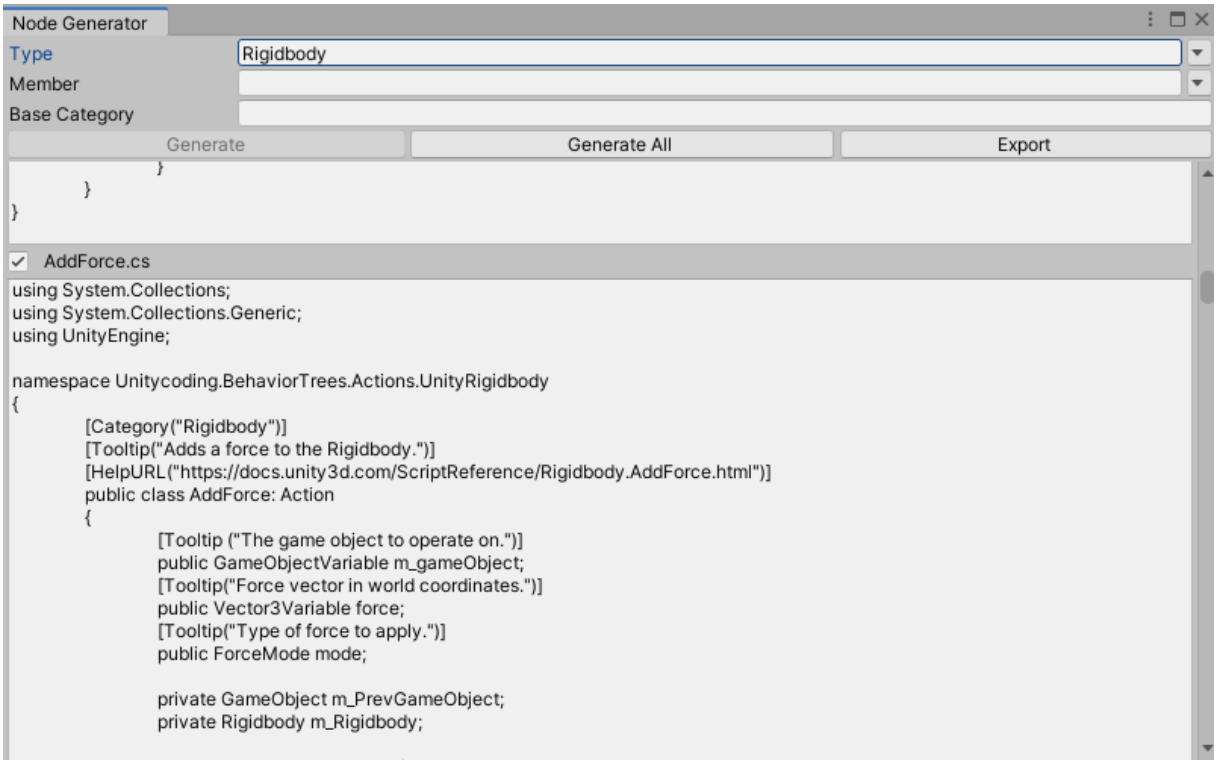
Open the Error Checker inside Unity: **Tools > Devion Games > Behavior Tree > Error Checker**



This tool helps you to find errors or misconfigurations in your behavior trees. When selecting an error it will automaticlly select the coresponding behavior tree and task with the error.

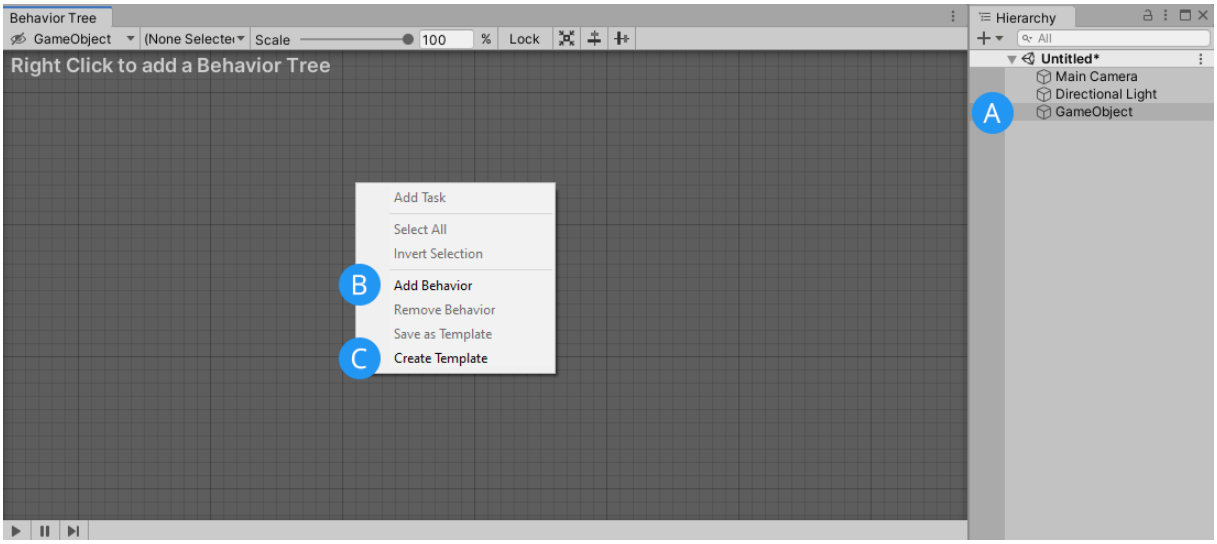
Task Generator

You can access the Task Generator in Unity at: **Tools > Devion Games > Behavior Tree > Task Generator.**

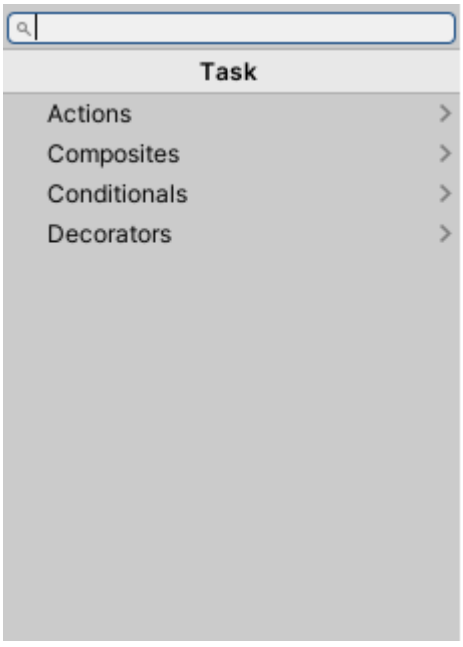


The **Task Generator** helps you to create or fully generate and save C# node code. This editor is able to create default unity nodes and third party nodes to use in the bahvior tree.

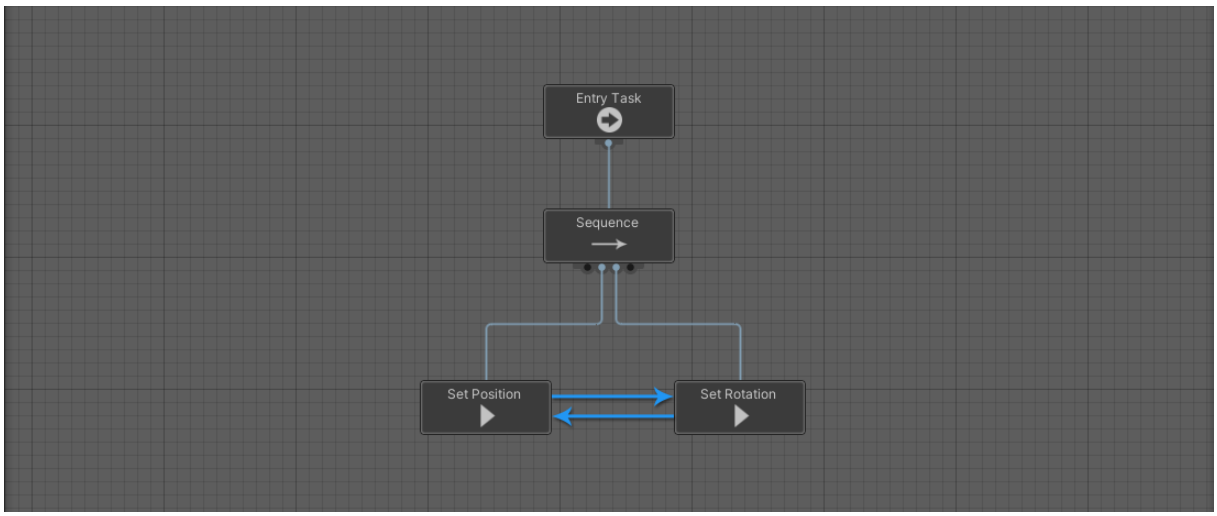
Create Behavior Tree



- **(A) Select a Game Objec**t in your scene, where you would like to add the behavior tree. You can select game objects in the scene view or in the hierarchy.
- **(B)** Right click on the canvas and select **Add Behavior**. This will add a Behavior component to your selected game object.
- **(C)** Alternatively you can create a **Behavior Tree Template**. This is a scriptable object asset that will be saved in your project. Templates can be reused on multiple objects in your scene, having a global control of them.

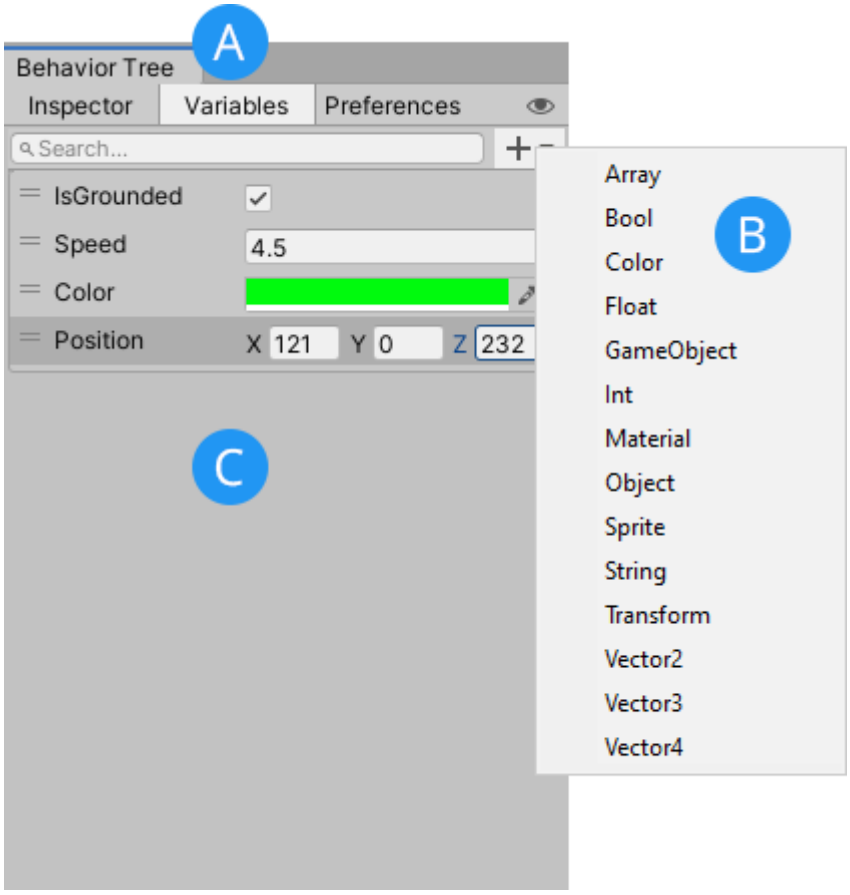


Now you can follow the **Add Task** button to add new nodes to your behavior tree or by clicking the space hotkey.



Move nodes to left or right to set the execution order. Please note that behavior trees are executed from top to down and left to right.

Create Variables

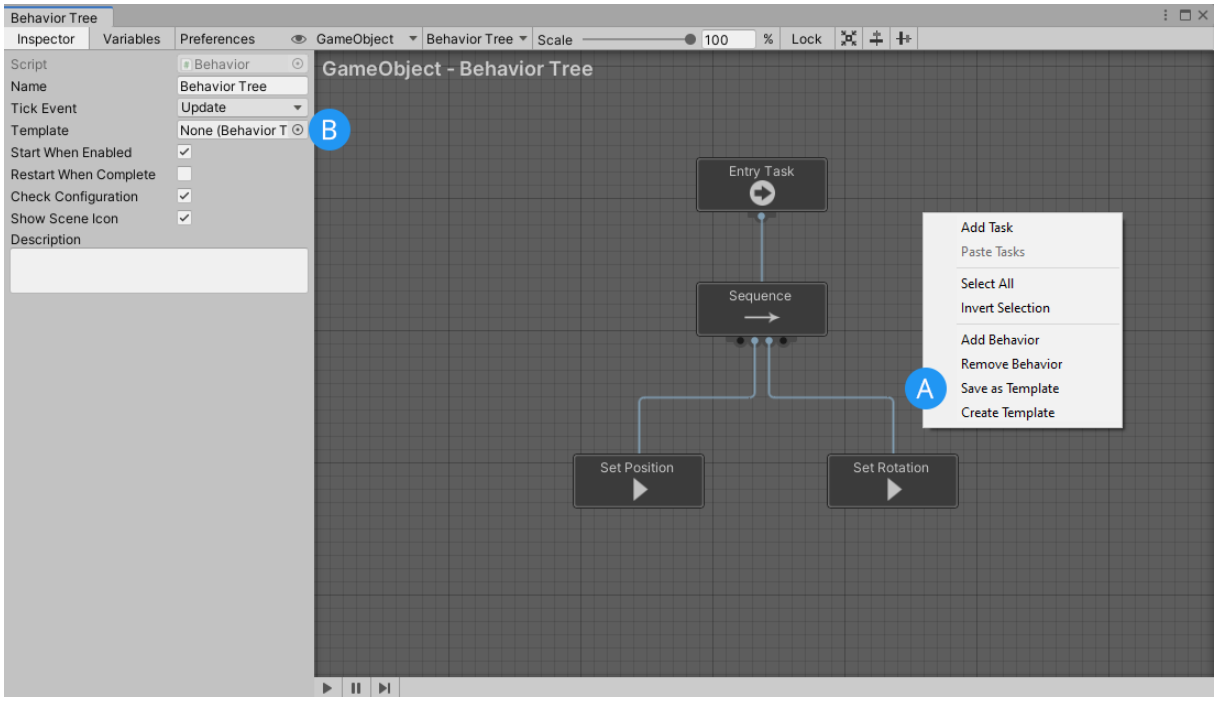


- **(A)** Navigate to the variables tab.
- **(B)** Click on the plus button and select the variable type to create it.
- **(C)** Enter default values in the inspector.

Global Variables can be created in the same way. Navigate to **Tools > Devion Games > Behavior Trees > Global Variables**.

Use Templates

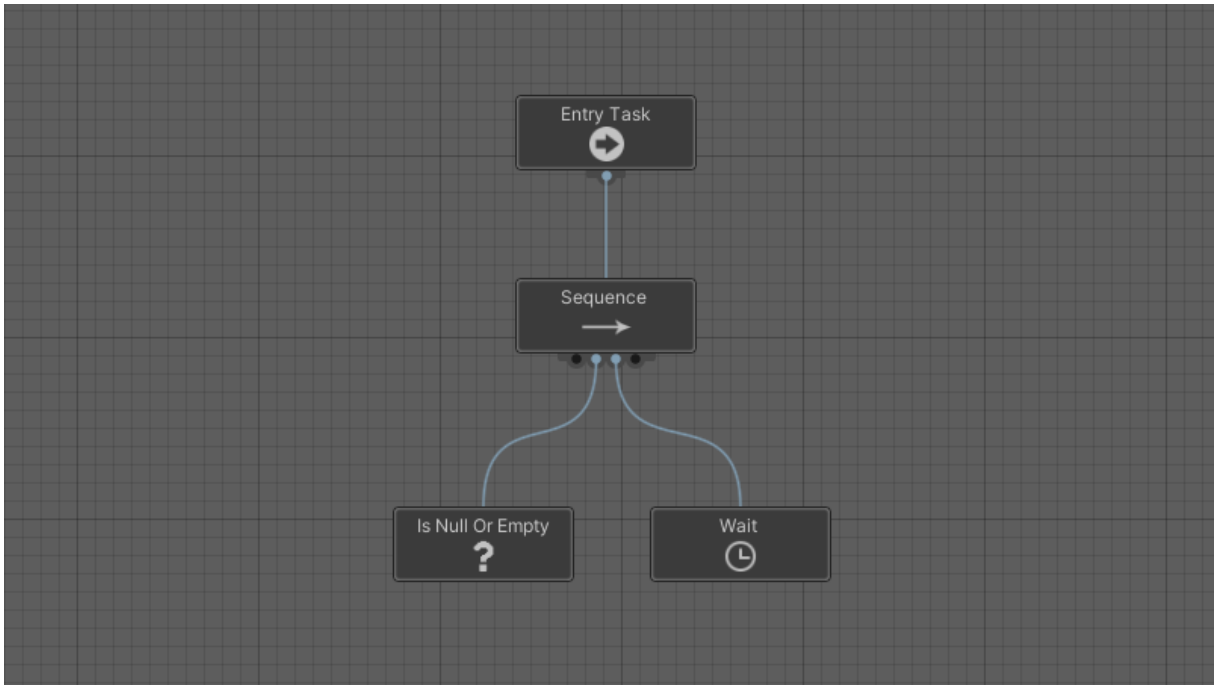
Templates are a very powerful feature. Any behavior tree can be saved as a Template that can then be re-used.



- **(A)** Context Click in the canvas and select **Save As Template**. A file dialog will open and you can save the template to a folder in your project.
- **(B)** Go to the Inspector and deselect any nodes so the **Behavior** component shows up. Here you can select your saved Template from the assets.

Abort Tasks

Conditional Aborts allow you to interrupt tasks from running if a condition is met. A basic example is the following behavior tree.



If the condition **IsNullOrEmpty** is met the sequence will start running the next child, the **Wait** task. The Wait task has a wait duration of x seconds. While the wait task is running, lets say that the conditional task changes its state and now returns failure. If aborts are enabled, the conditional task will issue an abort and stop the Wait task from running.

Self Abort
The conditional task can only abort an sction task if they both have the same parent composite task.

Lower Priority Abort
The conditional task will abort the right branch of its parent.