Behaviour Tree Graph — Unity Tool

Version 1.0.0 • 2025-10-09

Author: Aleksis Bojaruns (AleM)
Namespace: AleM.BehaviourTrees

This document is the official user guide for the Behaviour Tree Graph package. It includes setup, workflow, API, examples, and troubleshooting so reviewers and users can get productive fast.

1. Overview

Behaviour Tree Graph is a lightweight, production-friendly behaviour tree editor and runtime for Unity. It uses Unity's GraphView to author trees visually, then generates a runtime BehaviourTree at play mode for your AI agents.

- Visual node editor (Tools → Behaviour Tree Graph)
- Core nodes: Sequence, Selector, Leaf + Inverter decorator
- Random selector & cycle-through execution options
- ScriptableObject graph asset per agent (Resources)
- Tiny, readable runtime API (NodeBT, Leaf, Sequence, Selector, RSelector, Inverter)
- Example scene included: Scenes/ExampleBehaviourTree.unity

2. Requirements

- Unity 2020.3 LTS or newer (GraphView API present)
- TextMeshPro optional (not required)
- Scripting backend: .NET 4.x Equivalent

3. Installation

1. Import the package into your Unity project (Assets/BehaviourTree).
2. Open an example scene: Assets/BehaviourTree/Scenes/ExampleBehaviourTree.unity (optional).
3. Open the editor via menu: Tools → Behaviour Tree Graph.

4. Folder Layout

- Assets/BehaviourTree/Scripts — Runtime scripts (AleM.BehaviourTrees).
- Assets/BehaviourTree/Scripts/Editor — Graph editor & save utility.
- Assets/BehaviourTree/Resources — Styles (.uss) and saved graphs.
- Assets/BehaviourTree/Scenes — Example scene.
- Assets/BehaviourTree/Examples — Documentation images.

5. Quick Start (5 minutes)

4. Create an agent in your scene and add a component that inherits from BTAgent.
5. Open Tools → Behaviour Tree Graph. In the object field, assign your BTAgent instance.
6. Create nodes (Create Leaf / Create Sequence / Create Selector) and connect ports to form a tree.
7. For each Leaf, type the method name implemented on your agent (must return NodeBT.Status).
8. Click Save. A ScriptableObject will be created at Assets/BehaviourTree/Resources/<AgentName>_btgraph.asset.
9. Click "Set Container for Agent" to assign that asset back to your BTAgent.
10. Enter Play. The BTAgent builds the BehaviourTree from the container and starts ticking automatically.

6. Core Concepts

6.1 Nodes & Status

All nodes derive from NodeBT and return a Status each tick: SUCCESS, FAILURE, or RUNNING.

- Leaf — Calls a method on your agent. You implement the behaviour and return a Status.
- Sequence — Executes children left-to-right until one FAILS (then Sequence returns FAILURE). If all succeed, returns SUCCESS.
- Selector — Executes children until one SUCCEEDS (then Selector returns SUCCESS). If all fail, returns FAILURE.
- RSelector — Like Selector, but randomizes child order once per evaluation cycle.
- Inverter — Decorator that flips SUCCESS ↔ FAILURE. RUNNING passes through unchanged.

6.2 Execution Options

- Cycle Children — When enabled on Sequence/Selector, iterates over all children within a single update loop; otherwise evaluates one child per agent update (useful to throttle work).
- Random? — (Selector only) Enables RSelector behaviour (random child order).
- Invert/Ordered toggle — Button on the node header; when set to Inverted, the node's resulting status is inverted.

7. Editor Window

Open via Tools → Behaviour Tree Graph. The toolbar provides:

- Save — Serializes the current graph to Assets/BehaviourTree/Resources/<AgentName>_btgraph.asset
- Create Leaf / Create Sequence / Create Selector — Adds nodes at the graph center
- Set Container for Agent — Assigns the generated ScriptableObject to the selected BTAgent

Tips: Double-click on a node to focus it. Ports support multi-connections where allowed. The entry node is the root of the tree; ensure it feeds into your top-most composite (Sequence/Selector).

8. Implementing Leaf Methods

Leaves call a method on your agent by name. The method can be public or private, but it must be defined on the class inheriting BTAgent and return AleM.BehaviourTrees.NodeBT.Status.

```
using AleM.BehaviourTrees;
using UnityEngine;

public class WolfAgent : BTAgent
{
    // called automatically
    protected override void Start()
    {
        base.Start(); // builds the tree from the assigned BehaviourTreeContainer
    }

    // Leaf methods MUST return NodeBT.Status
    private NodeBT.Status Patrol()
    {
        // Move along waypoints...
        return NodeBT.Status.RUNNING; // keep going
    }

    private NodeBT.Status Chase()
    {
        // If target lost, fail; if reached, succeed
        return NodeBT.Status.SUCCESS;
    }

    private NodeBT.Status Eat()
    {
        // If no food, fail.
        return NodeBT.Status.FAILURE;
    }
}
```

In the editor, create Leaf nodes and set method names: Patrol, Chase, Eat. The editor validates that those methods exist on the selected BTAgent.

9. Runtime API (Summary)

```
• NodeBT
  - Status Process(): Override in nodes. Called repeatedly by the agent.
  - void AddChild(NodeBT n): Attach a child node.
  - Fields: status, children, currentChild, name, sortOrder

• BehaviourTree : NodeBT
  - Root node. Holds children and utilities (e.g., PrintTree())

• Sequence : NodeBT
  - Options: loopThroughAllChildren
  - Behaviour: SUCCESS when all children succeed; FAILURE on first failure
```

```
• Selector / RSelector : NodeBT
  - Options: loopThroughAllChildren; RSelector shuffles children once per cycle
  - Behaviour: SUCCESS when any child succeeds; FAILURE if all fail

• Inverter : NodeBT
  - Behaviour: Inverts SUCCESS/FAILURE of its single child; RUNNING passes through

• Leaf : NodeBT
  - Delegates:
    delegate Status Tick();
  - Constructed with a delegate bound to your agent’s method
```

10. Saving & Loading

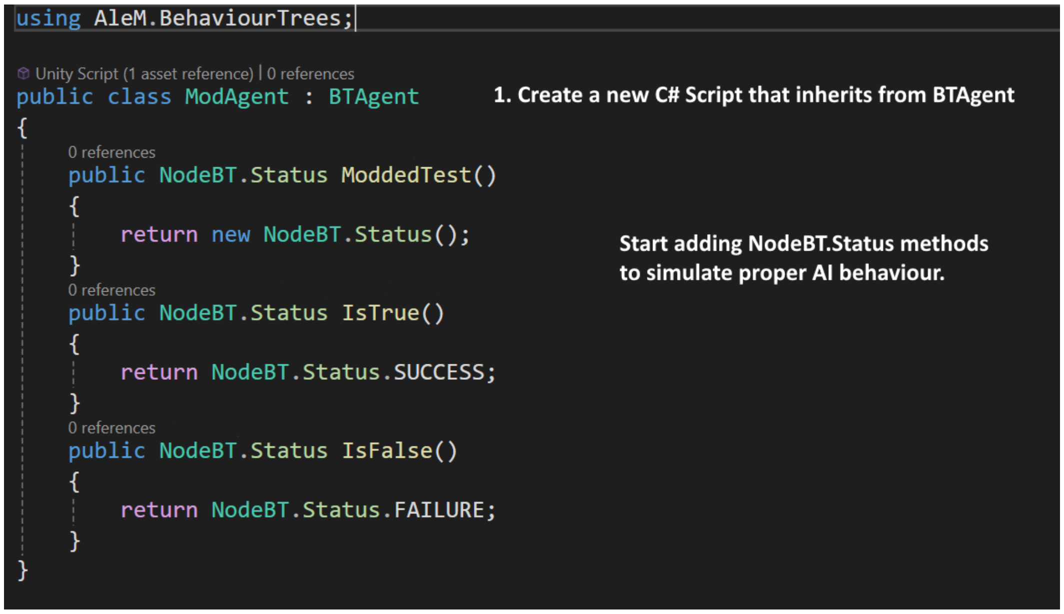
Click Save to generate a BehaviourTreeContainer asset in Assets/BehaviourTree/Resources named <AgentName>_btgraph. Use Set Container for Agent to assign this asset to the current BTAgent. At runtime, BTAgent.GenerateTree() builds a runnable BehaviourTree from the container.

11. Example Workflow

A simple patrol → chase → eat behaviour can be authored as:

- Create a Sequence as root child (Ordered).
- Add children Leaves: Patrol → Chase → Eat.
- Set method names accordingly on the Leaves.
- Save and Set Container for Agent, then Play.

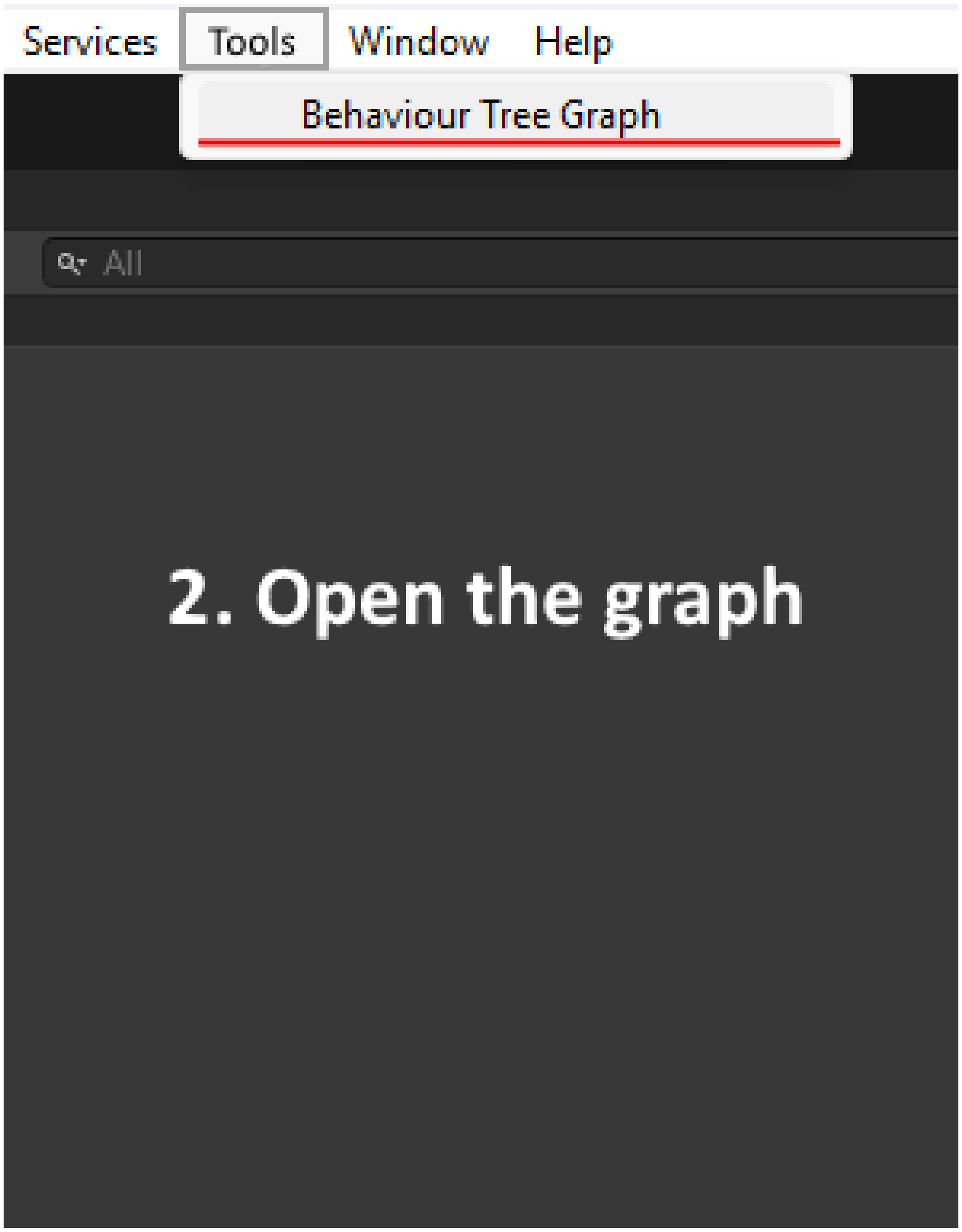
Selected screenshots:



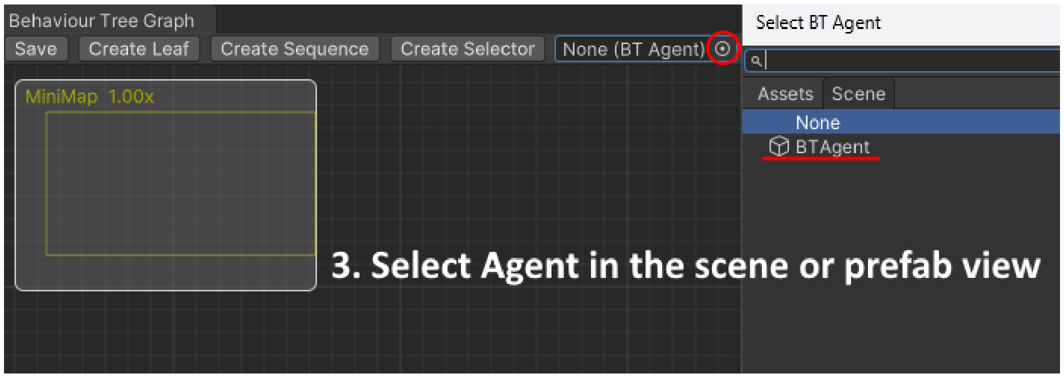
```
using AleM.BehaviourTrees;

Unity Script (1 asset reference) | 0 references
public class ModAgent : BTAgent
{
    0 references
    public NodeBT.Status ModdedTest()
    {
        return new NodeBT.Status();
    }
    0 references
    public NodeBT.Status IsTrue()
    {
        return NodeBT.Status.SUCCESS;
    }
    0 references
    public NodeBT.Status IsFalse()
    {
        return NodeBT.Status.FAILURE;
    }
}
```

1. Create a new C# Script that inherits from BTAgent

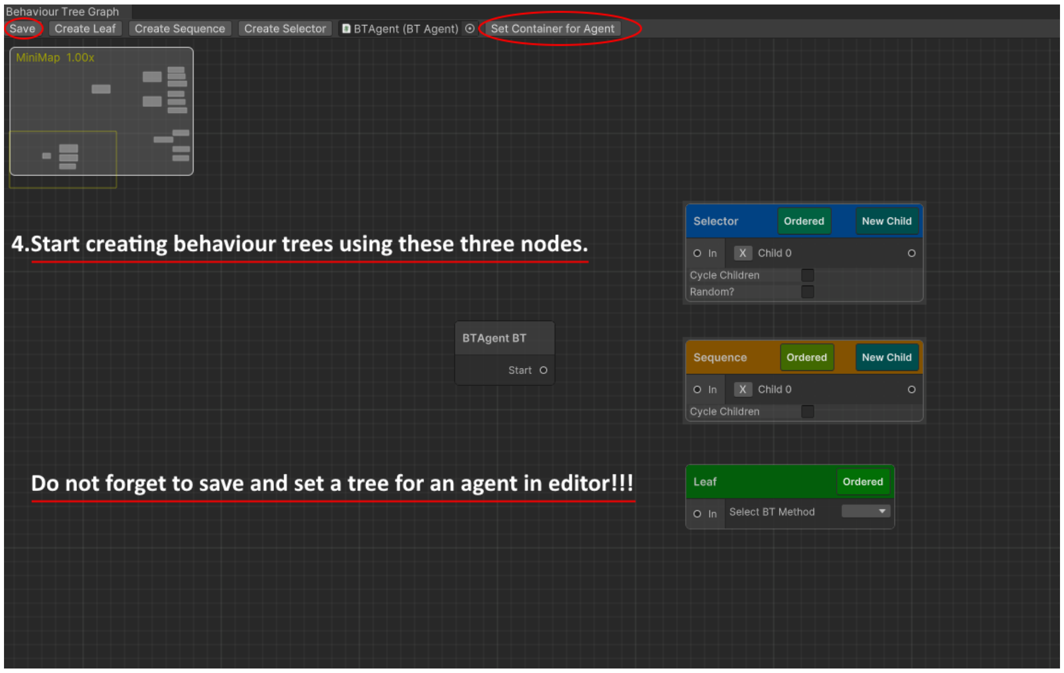
Start adding NodeBT.Status methods to simulate proper AI behaviour.



2. Open the graph

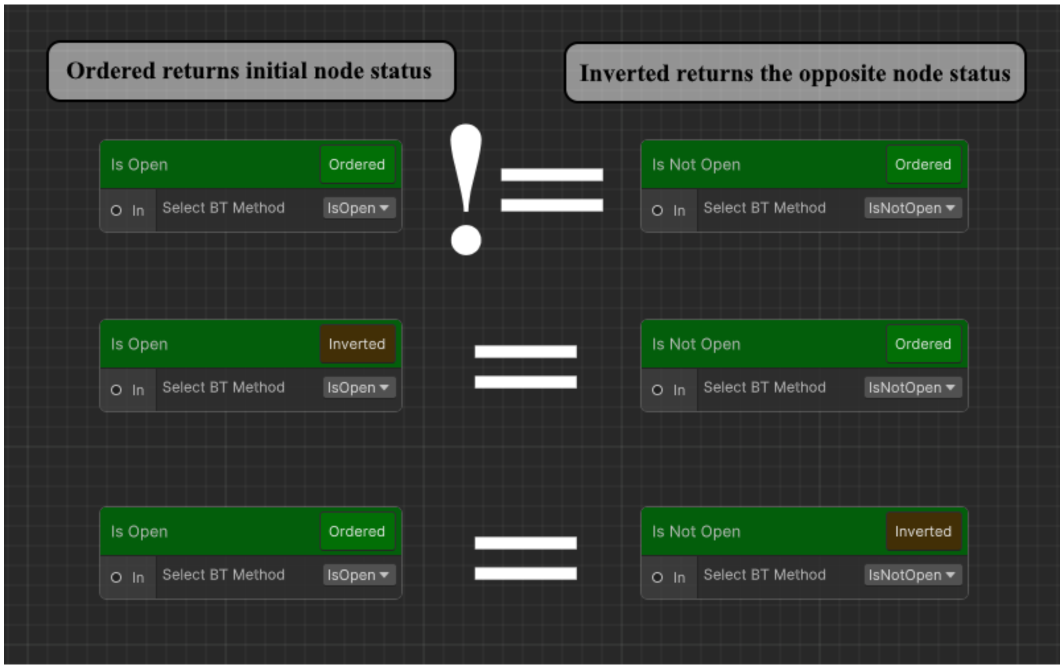


3. Select Agent in the scene or prefab view

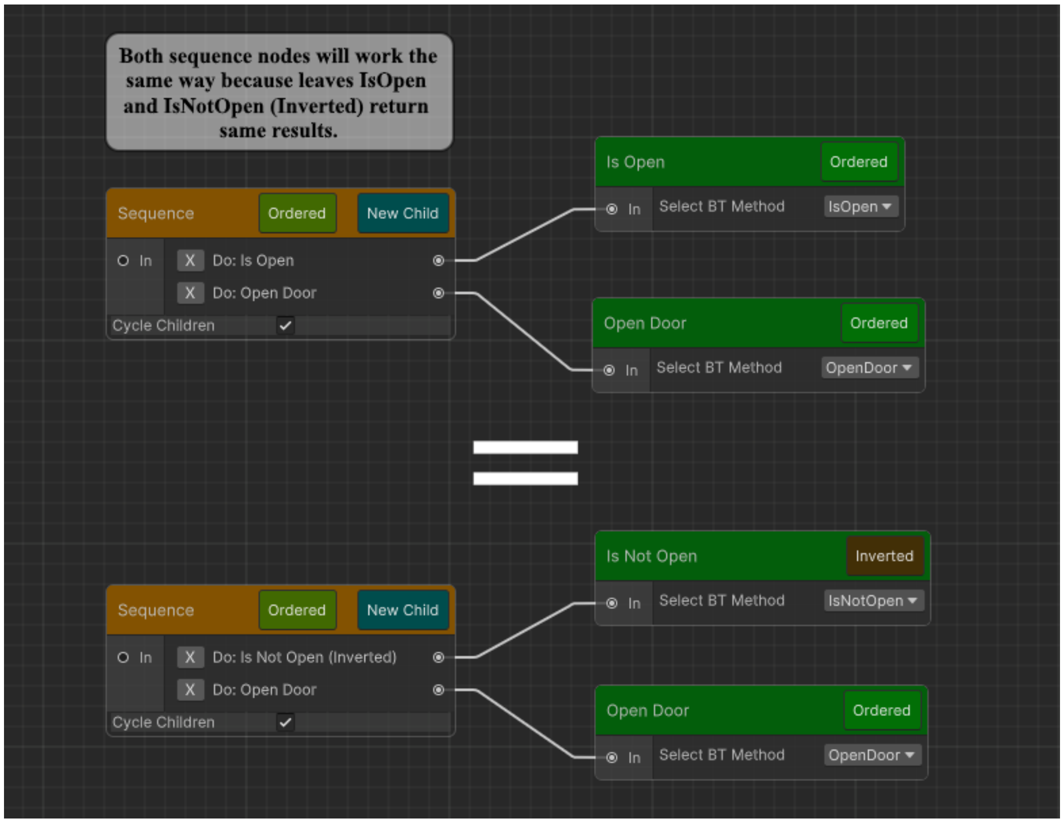


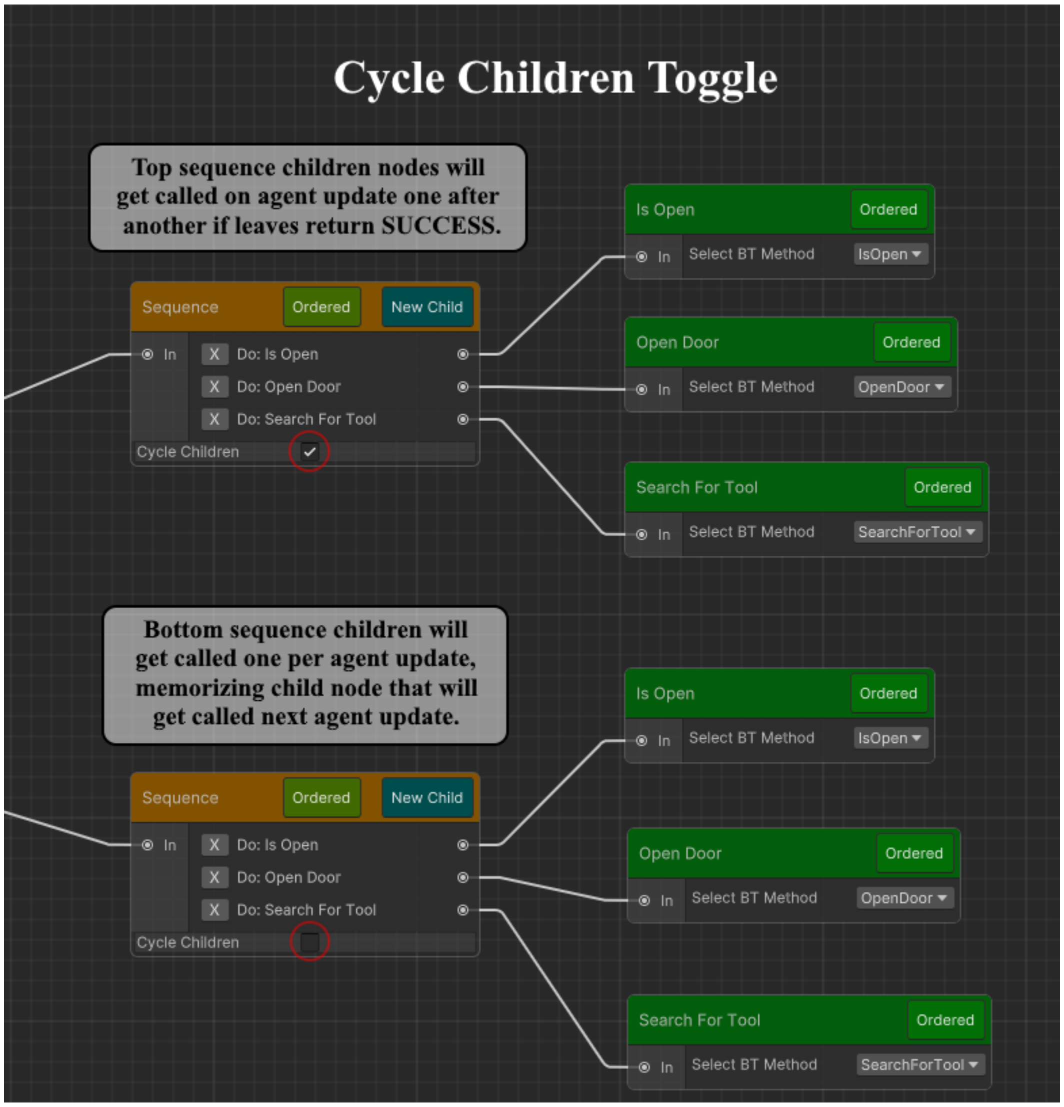
4.Start creating behaviour trees using these three nodes.

Do not forget to save and set a tree for an agent in editor!!!


Ordered returns initial node status

Inverted returns the opposite node status

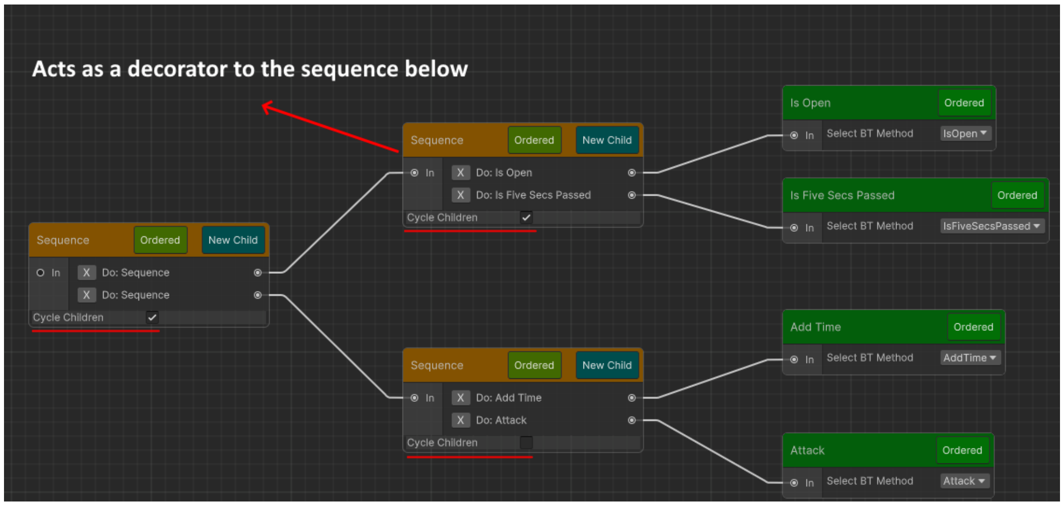

Both sequence nodes will work the same way because leaves IsOpen and IsNotOpen (Inverted) return same results.



Cycle Children Toggle

Top sequence children nodes will get called on agent update one after another if leaves return SUCCESS.

Bottom sequence children will get called one per agent update, memorizing child node that will get called next agent update.



Acts as a decorator to the sequence below

12. Best Practices

- Keep leaf methods short and pure. Use components for movement, perception, etc.
- Prefer RUNNING for ongoing actions (movement) to avoid hot loops.
- Use Inverter to negate conditions instead of duplicating methods.
- Throttle evaluation with Cycle Children off if your trees are heavy.
- Group conditions under a Sequence; actions under a Selector when appropriate.

13. Troubleshooting

- Leaf method not found: Ensure the method name matches exactly and exists on the BTAgent; public or private is fine, but it must return NodeBT.Status and take no parameters.
- Nothing happens in Play: Confirm your BTAgent has a BehaviourTreeContainer assigned (use Set Container for Agent) and your root is connected to composites/leaves.
- Graph didn't save: Check the Console for errors; the graph asset is saved under Assets/BehaviourTree/Resources.
- Wrong method bound: Open editor, select Leaf, correct the Method field, Save again.
- Performance spikes: Disable Cycle Children on heavy composites to spread evaluation across frames.

14. FAQ

- Can I call async code? No. Keep leaf methods synchronous and use RUNNING for multi-frame work.
- How do I share one tree across many agents? Save one graph, assign the same container to multiple BTAgents.
- Can I extend with custom nodes? Yes. Derive from NodeBT and add your node; you can also extend the editor to add a creation button.
- Is there a random selector? Yes—toggle Random? on a Selector to use RSelector.

15. Changelog

- 1.0.0 — Initial public release.

16. License

This package is provided as part of a Unity Asset Store submission. Redistribution follows the Unity Asset Store EULA. For custom licensing, contact the author.

17. Support

For issues and feature requests, contact: <replace-with-your-email> or create an issue in your chosen tracker. Include Unity version, OS, and reproduction steps.

Appendix A — Menu Path

Tools → Behaviour Tree Graph

Appendix B — Example Scene

Assets/BehaviourTree/Scenes/ExampleBehaviourTree.unity