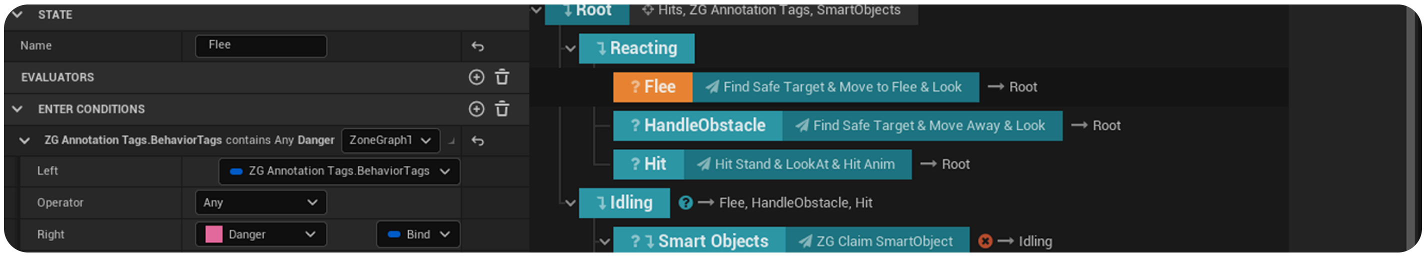
StateTree Debugger Quick Start Guide

Quick start guide of the StateTree Debugger in Unreal Engine.


STATE
Name
Flee
EVALUATORS
ENTER CONDITIONS
ZG Annotation Tags.BehaviorTags contains Any Danger
ZoneGraph1
Left
ZG Annotation Tags.BehaviorTags
Operator
Any
Right
Danger
Bind
Root
Hits, ZG Annotation Tags, SmartObjects
Reacting
Flee
Find Safe Target & Move to Flee & Look
Root
HandleObstacle
Find Safe Target & Move Away & Look
Root
Hit
Hit Stand & LookAt & Hit Anim
Root
Idling
Flee, HandleObstacle, Hit
Smart Objects
ZG Claim SmartObject
Idling

Introduction

The **StateTree Debugger** monitors and records StateTree runtime behavior to help developers understand and diagnose potential problems in their StateTrees.

The system was built with two main goals in mind - to provide a visual representation of the active States in the tree, and to provide live monitoring of runtime values for States, Tasks, and Conditions.

The system offers live debugging for Editor Sessions (e.g. Play in Editor) and Remote Sessions (e.g. Standalone, Client, Server). In addition, users can perform deferred analysis by saving and loading recorded trace files.

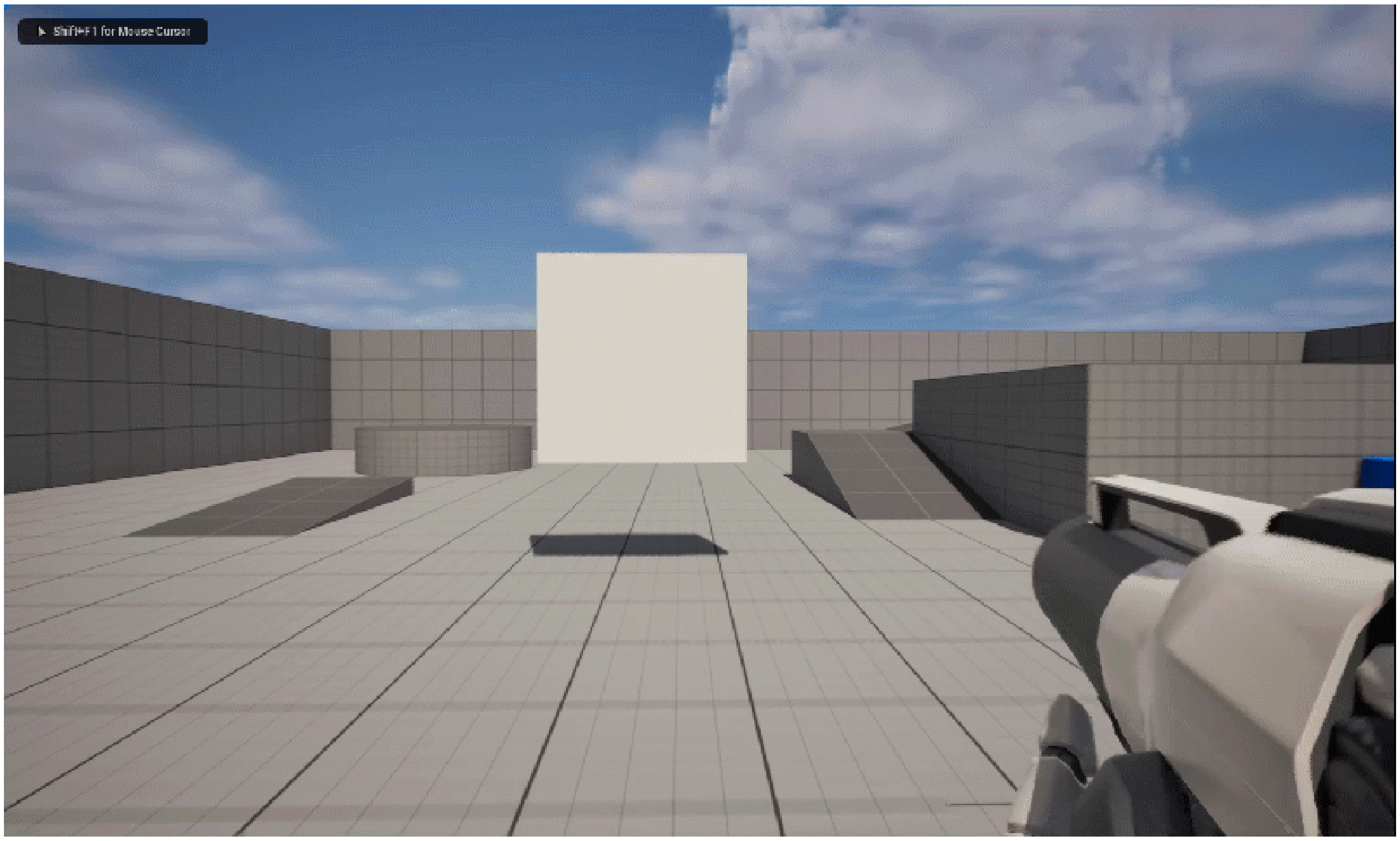
The StateTree Debugger uses Unreal Engine's **TraceServices**, such as Unreal Insights, to produce and analyze traced events. The system is built on top of the Trace Analyzer and Trace Providers, and it gathers the relevant events of one or multiple instances associated with a given StateTree asset.

This approach enables the system to debug multiple Editor, Client, and Dedicated Server processes at the same time from a single process.

Pre-requisites

This guide will use the StateTree created in the StateTree Quickstart Guide to demonstrate the StateTree Debugger. Please complete the Quickstart to follow along with the examples in this document.

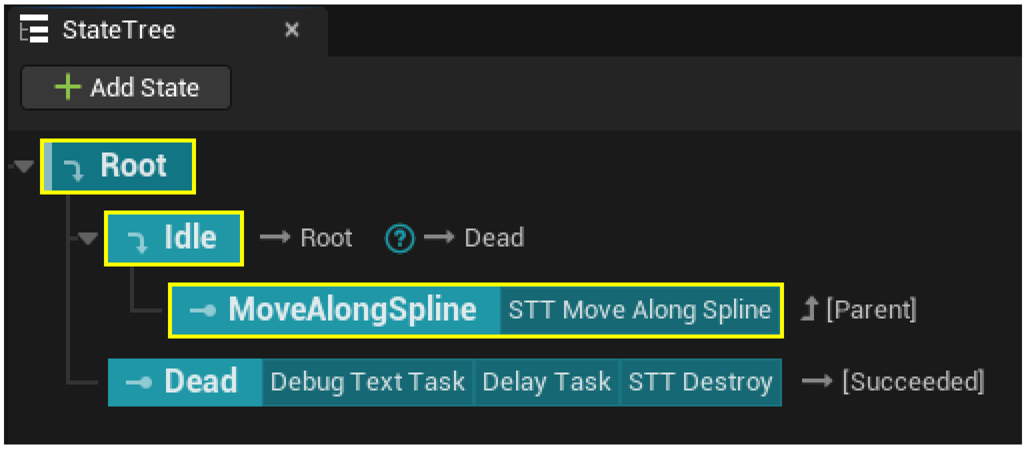
Once you complete the Quickstart guide, press **Play** to verify the behavior.



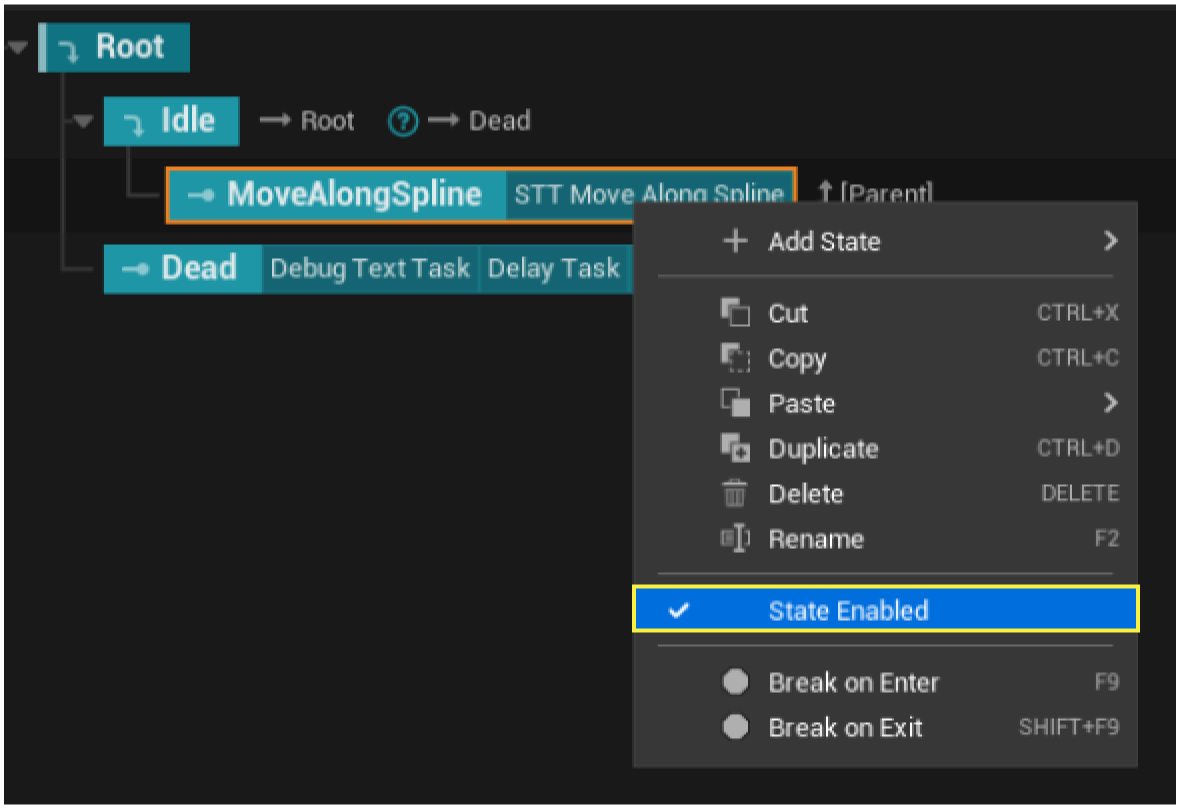
Enable and Disable States

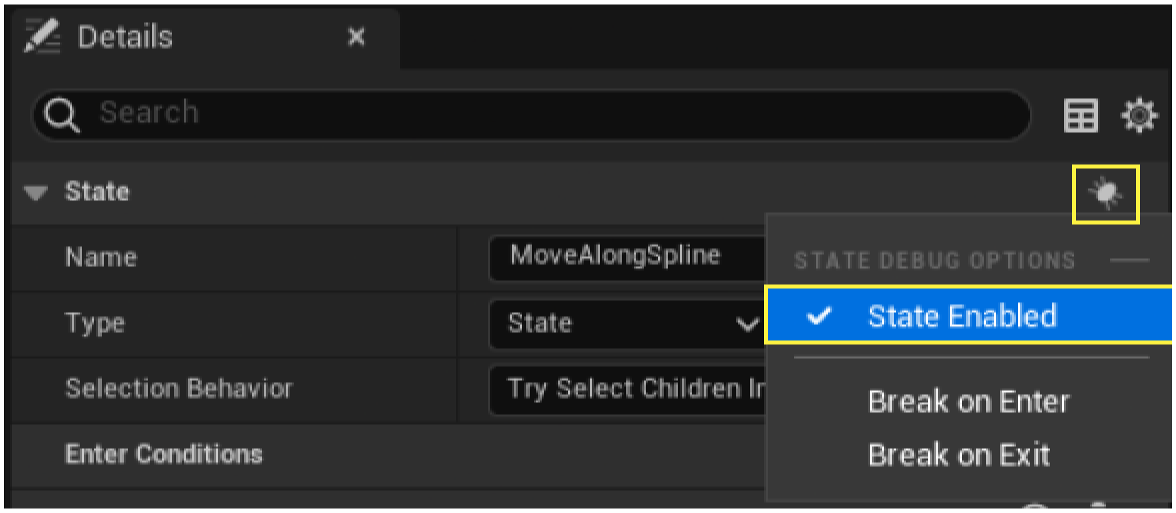
You can enable and disable individual specific States within a State Tree.

1. Open the StateTree asset **ST_ShootingTarget**.

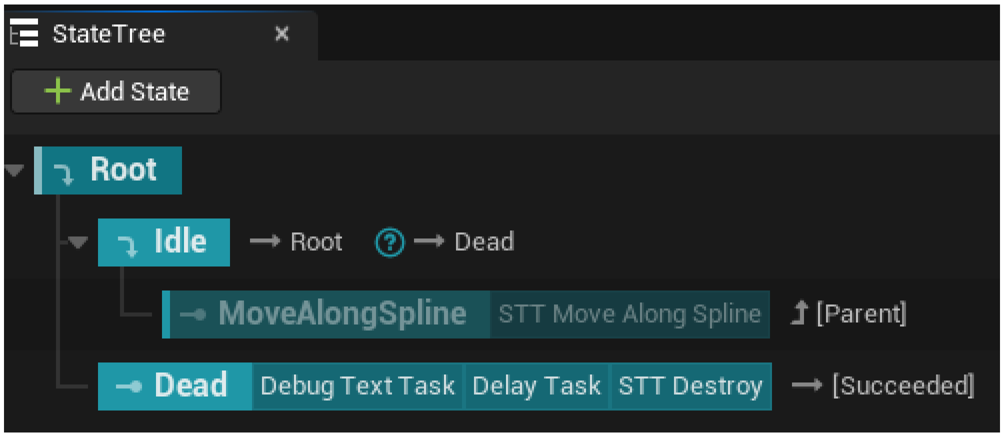


2. Right click the **MoveAlongSpline** State and deselect **State Enabled**. Alternatively, select the State and go to the **Details** panel. Click the **Debug Options** button and deselect **State Enabled**.

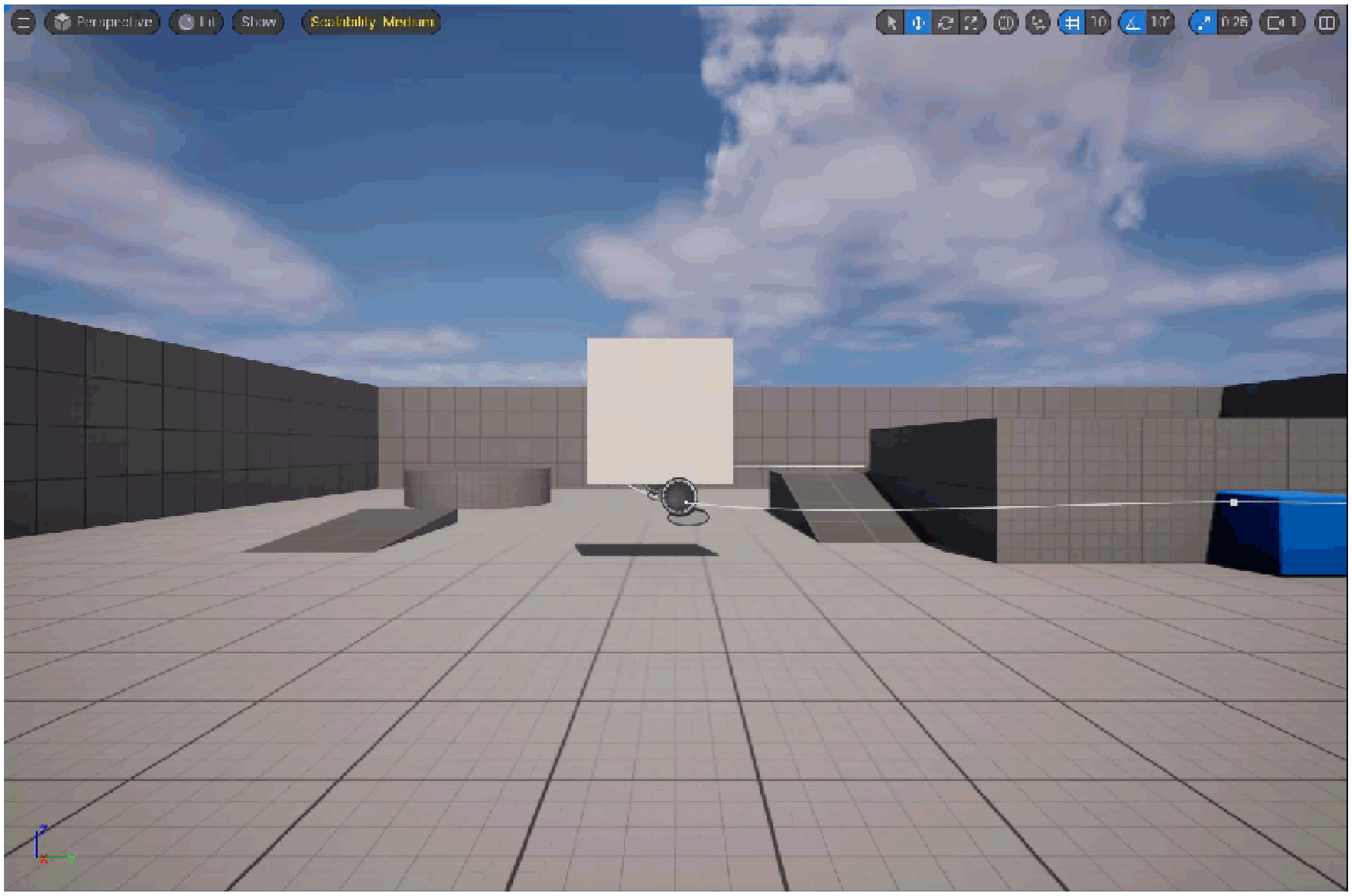




3. **Compile** and **Save** the State Tree. Notice how the disabled State is visualized with a darker color in the window.


StateTree
Add State
Root
Idle → Root ? → Dead
MoveAlongSpline STT Move Along Spline [Parent]
Dead Debug Text Task Delay Task STT Destroy → [Succeeded]

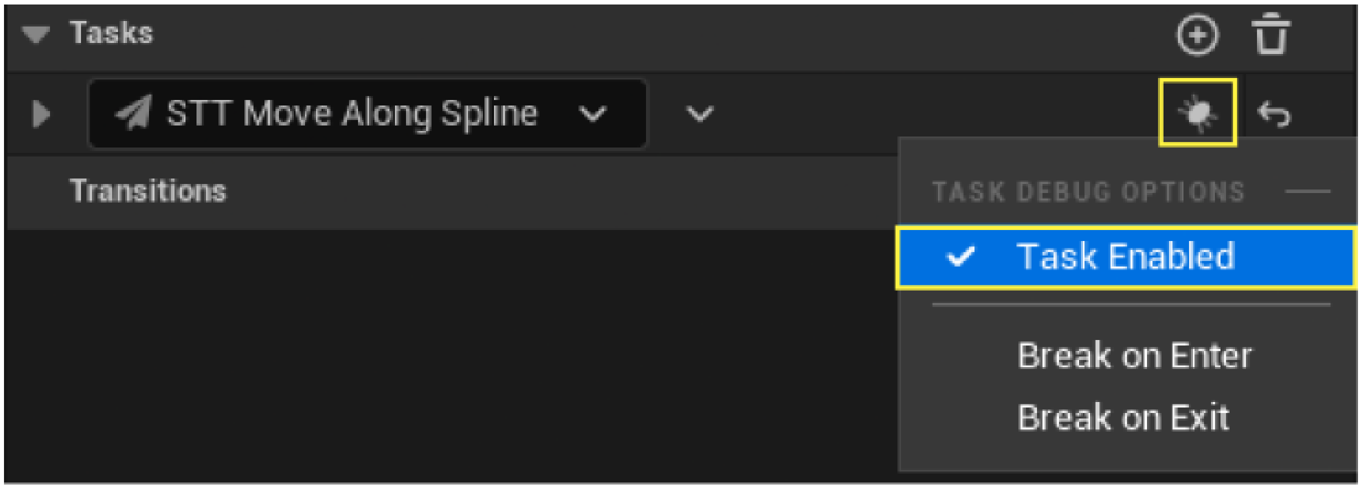
4. Press **Play** to see the results. The **MoveAlongSpline** State is disabled so the tree goes from the **Idle** to the **Dead** State immediately.



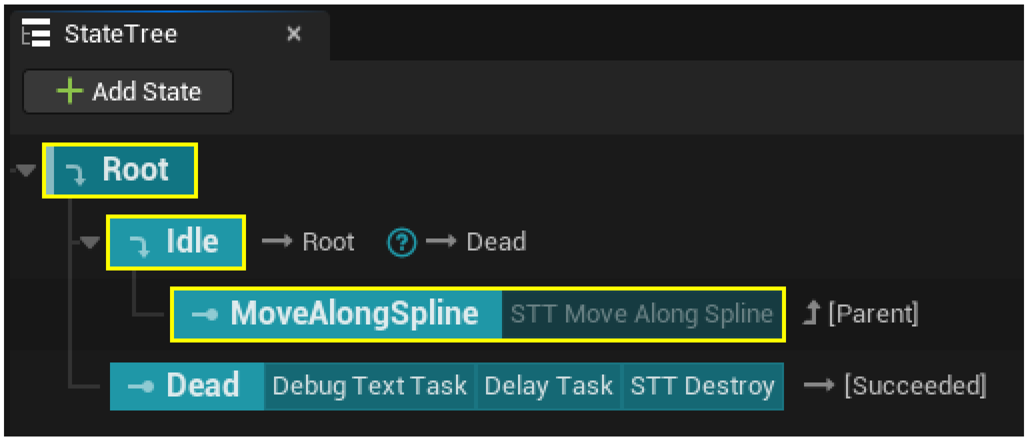
Enable and Disable Tasks

You can enable and disable individual Tasks within a State.

1. Enable the **MoveAlongSpline** State and go to the **Details** panel. Click the **Debug Options** button next to the **STT_MoveAlongSpline** Task and deselect **Task Enabled**.



2. **Compile** and **Save** the State Tree. Notice how the disabled Task is visualized by the darker color.

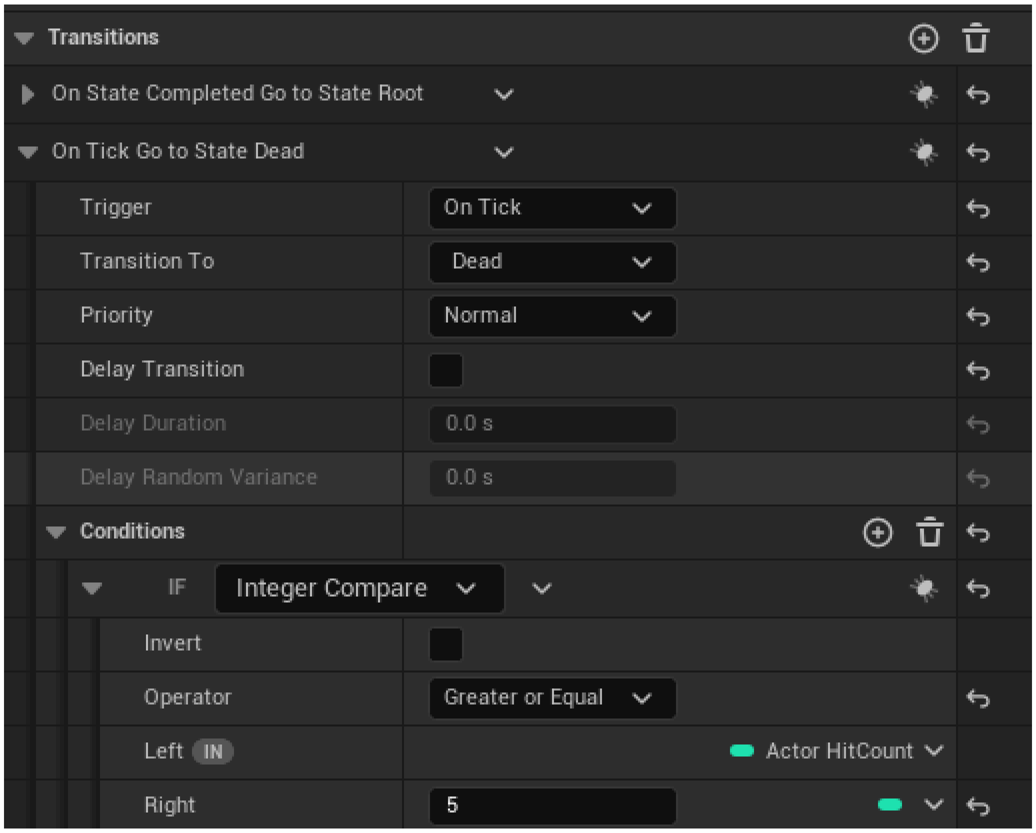


3. Press **Play** to see the results. The **MoveAlongSpline** State is enabled, but the **STT_MoveAlongSpline** Task is disabled. This results in the MoveAlongSpline State being evaluated, but because no Tasks are active, the tree returns to the Idle State.

Condition Debug Options

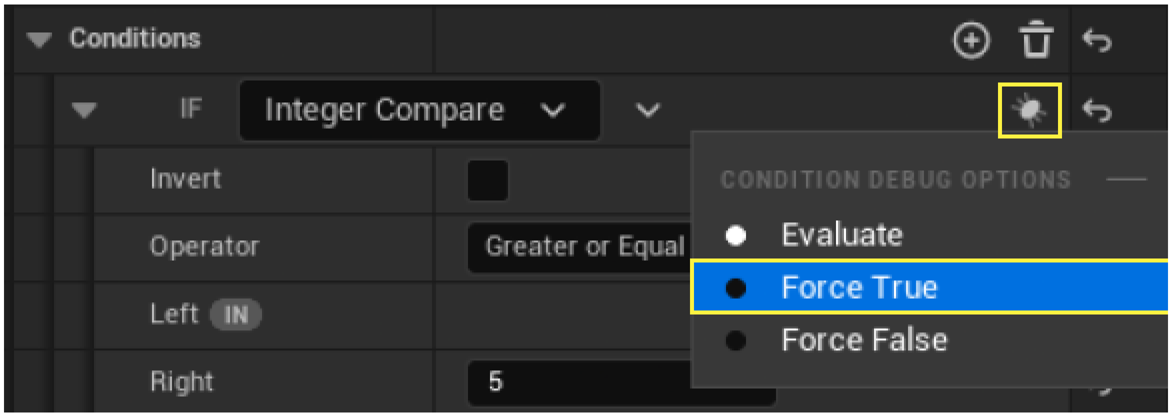
You can force the results of a condition check in a State Tree for testing purposes.

1. For this example, select the **Idle** State and go to the **Details** panel. a. Expand the **Transitions** section, then expand the **On Tick Go to State Dead** struct. b. Finally, expand the **Conditions** to see the **Integer Compare** condition between the **HitCount** and the value of **5**.



Transitions
On State Completed Go to State Root
On Tick Go to State Dead
Trigger: On Tick
Transition To: Dead
Priority: Normal
Delay Transition: ☐
Delay Duration: 0.0 s
Delay Random Variance: 0.0 s
Conditions
IF Integer Compare
Invert: ☐
Operator: Greater or Equal
Left IN: Actor HitCount
Right: 5

2. Click the **Conditions Debug** button and select **Force True**. This will result in this condition always returning true when evaluated.



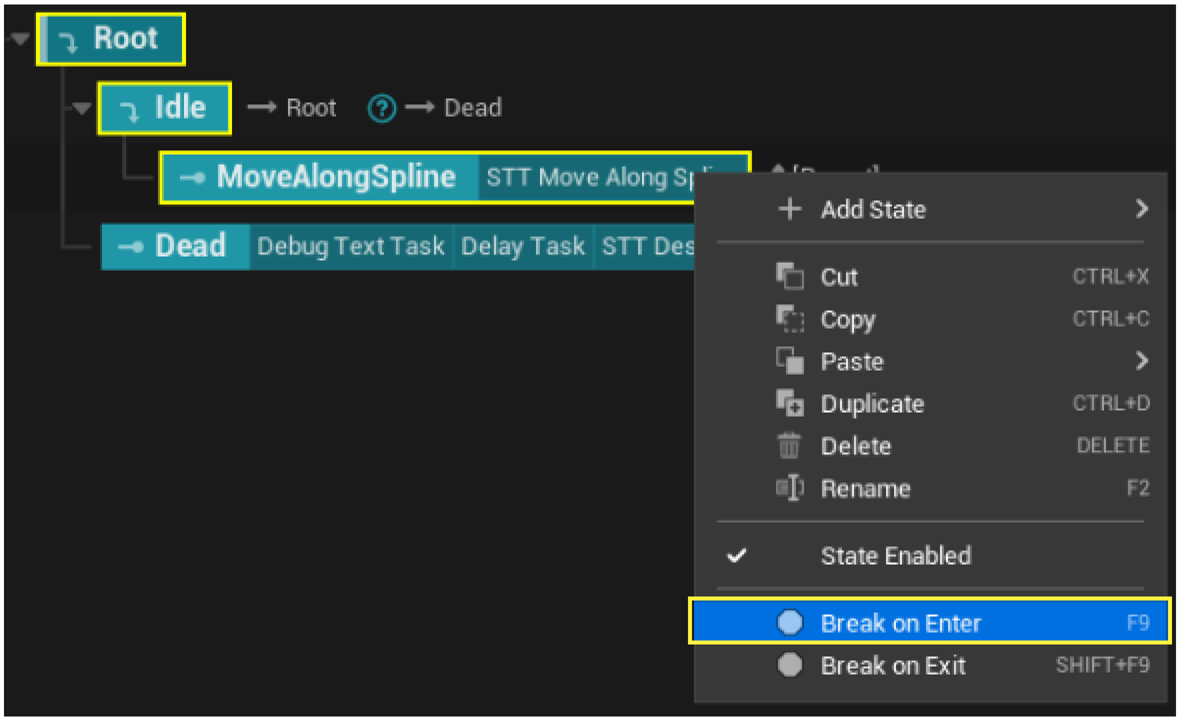
3. Press **Play** to see the results. When the Idle State is executed, the **Transitions** are evaluated. In this case, the **On Tick Go to State Dead** returns **True** because the Integer Compare between the Actor **HitCount** and **5** always returns True. In other words, we are simulating the Actor being hit more than 5 times.

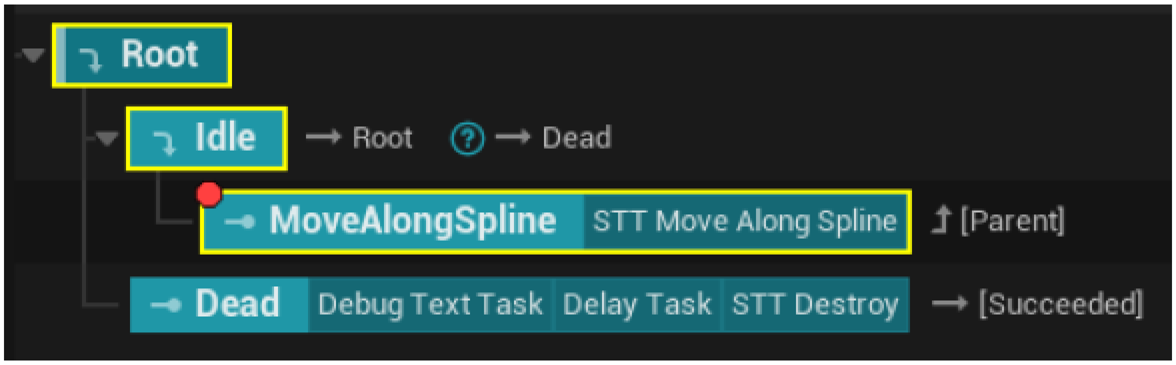
Breakpoints

Breakpoints can be added entering or exiting any State and Task, and while executing a Transition.

Breakpoints are stored temporarily throughout an Editor session. However, they will be lost if the asset gets reloaded. They do not require compilation of the StateTree, unlike Disabling States or Tasks.

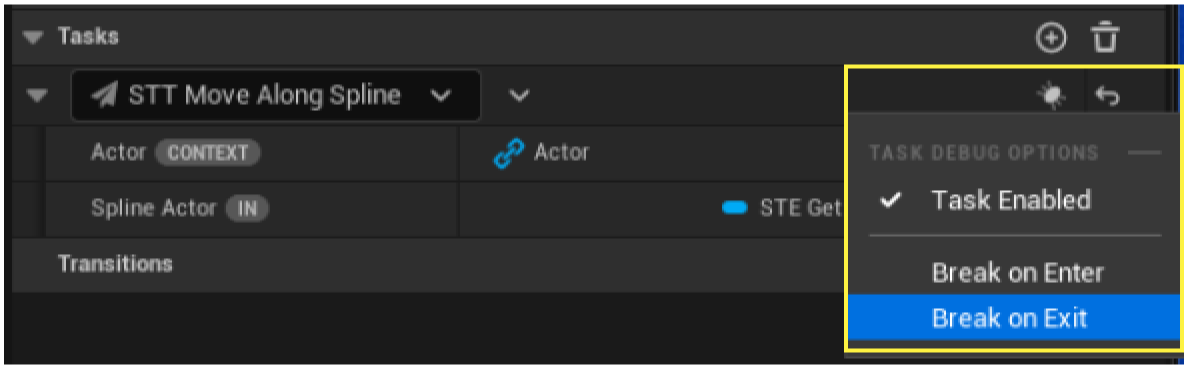
1. For this example, right click the **MoveAlongSpline** State and select Break on Enter. Notice how the State now has a red icon signifying that execution will break once the tree enters this State.





2. **Compile** and **Save**. Click **Play** to test the results. As you can see, execution stops as soon as the State is entered.

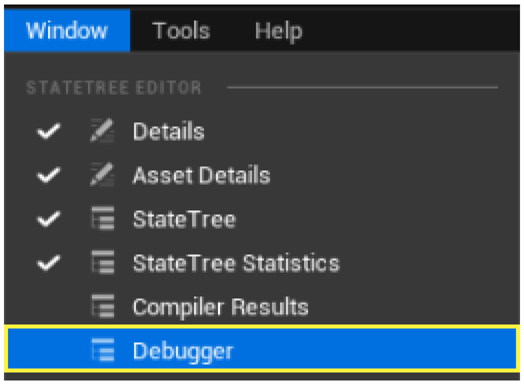
3. You can also add a Breakpoint to a Task by clicking the **Task Debug Options** button next to the Task name and selecting **Break on Enter** or **Break on Exit**.



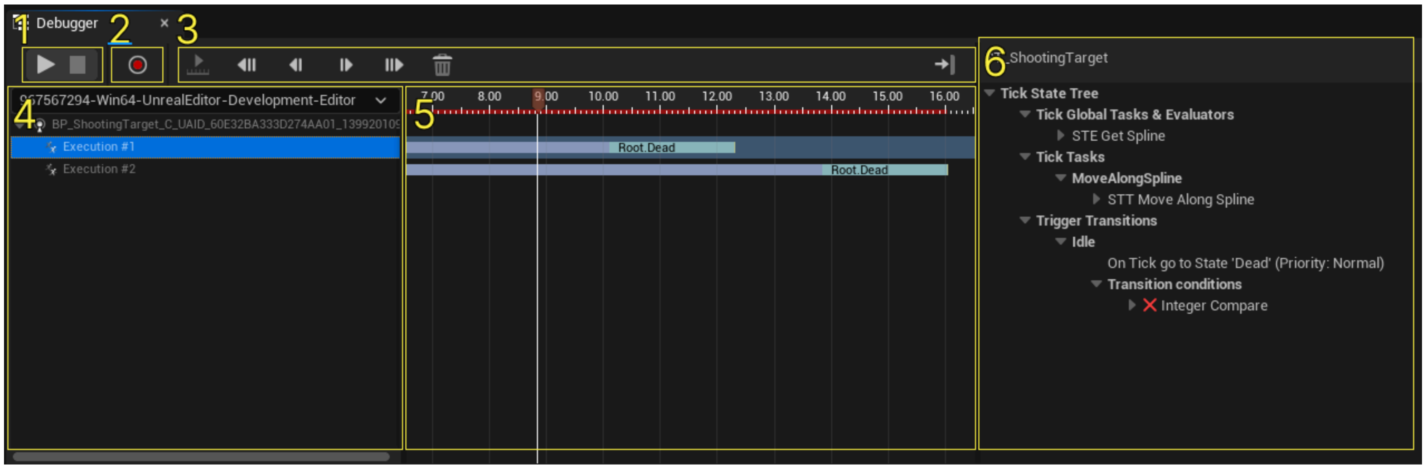
The Debugger Tab

The **Debugger** tab provides detailed runtime information about the State Tree. It can be used to track its execution and get variable data during a pause.

You can open the tab by clicking **Window > Debugger**.

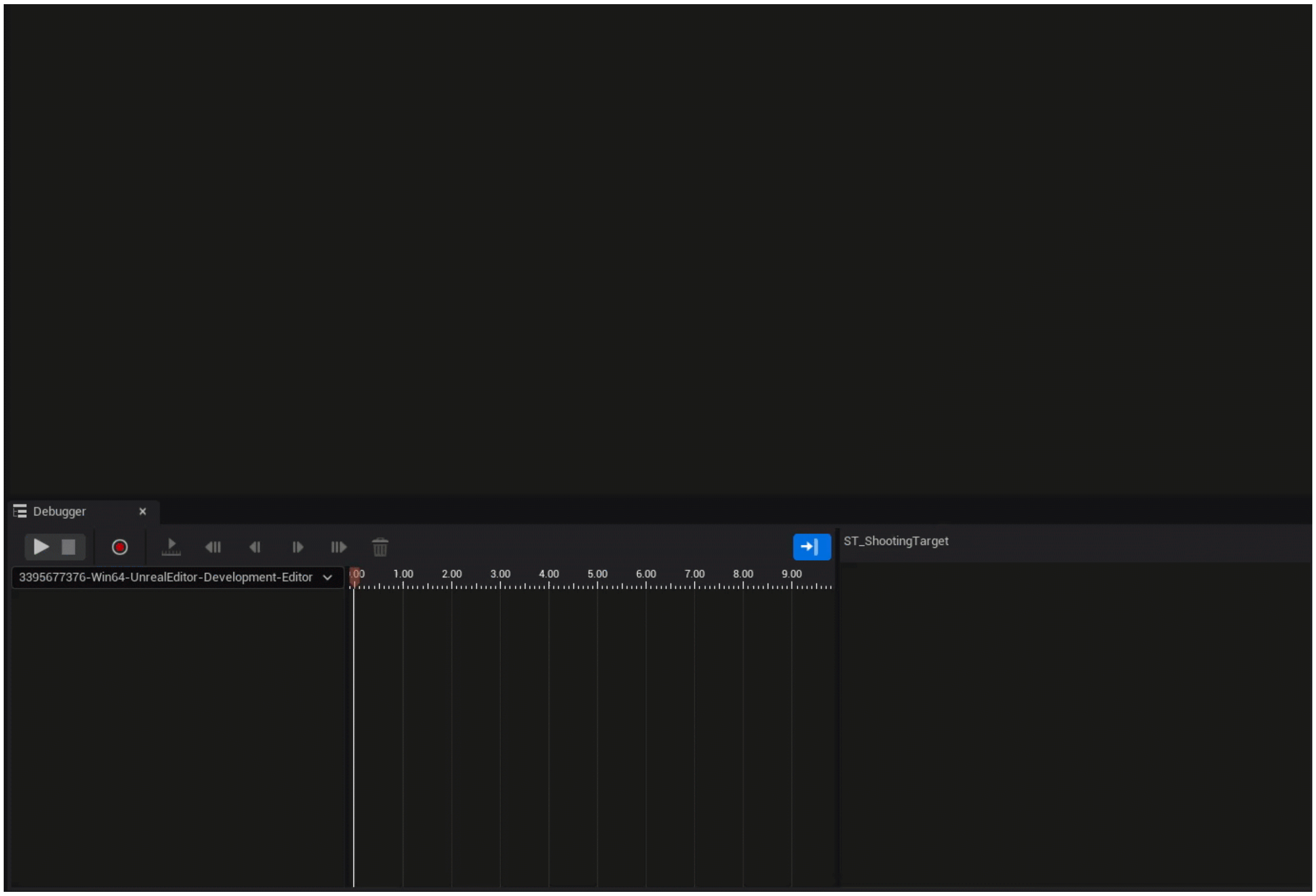


The Debugger tab interface has the following areas:



- **(1) Editor simulation controls**: These buttons control simulation in the viewport. You can start, pause, and stop the simulation.
- **(2) Trace Session recorder**: This button records the live session in the Visual Logger for later review. Pressing it will start the tracing with the Frame and StateTreeDebug channels enabled.
- **(3) Analysis controls**: These buttons control the recorded session playback. You can start and stop the trace analysis, step through the session one frame at a time, or jump to the previous or next changed frame.
- **(4) Trace and Execution area**: You can select a specific trace from the dropdown. You can also select a specific recorded execution.
- **(5) Timeline**: The timeline shows the available executions along with their active States. You can scrub the timeline manually to get information about a specific state.
- **(6) Details panel**: this panel shows execution details about the active State of the selected Execution. It displays information such as global Tasks and Evaluators, Tasks, and Transitions. It also shows the data and logic that was executed in the State.

In the example below, we clicked the **Play** button to start the play session and record the first **execution**. Once we stopped play mode, we scrubbed the timeline to view the recorded details of the executed States.


Debugger
3395677376-Win64-UnrealEditor-Development-Editor
0.00
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
ST_ShootingTarget