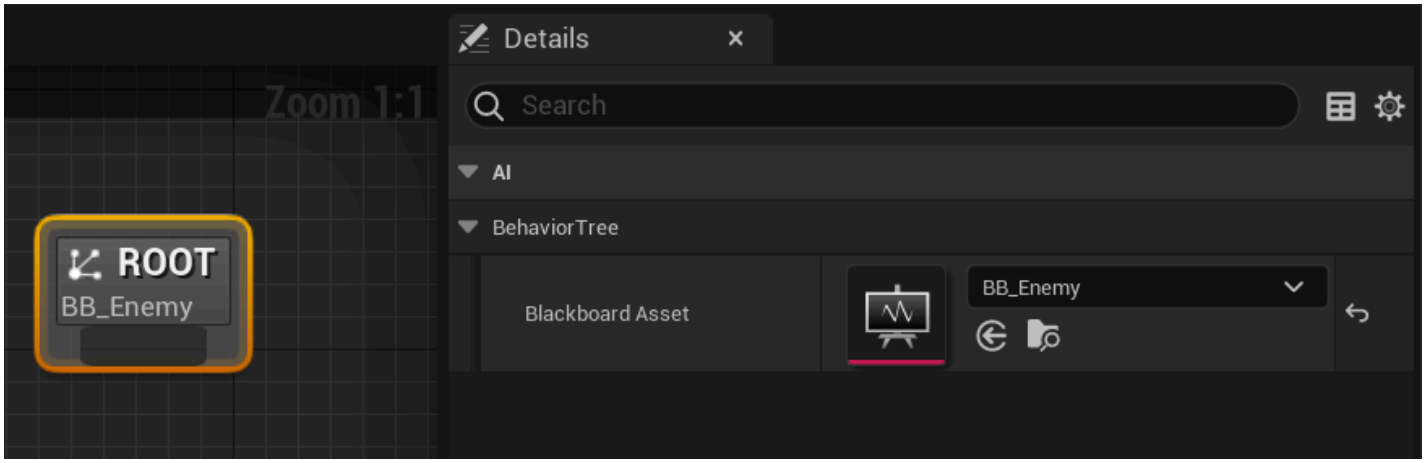
Behavior Tree Node Reference

Outlines the different types of nodes available when working in the Behavior Tree Editor.

Behavior Tree Nodes (base class) perform the main work of Behavior Trees, including tasks, logic flow control, and data updates.

Behavior Tree Node Types

The node that serves as the starting point for a Behavior Tree is a **Root node**. This is a unique node within the tree, and it has a few special rules. It can have only one connection, and it does not support attaching **Decorator Nodes** or **Service Nodes**. Although the Root node has no properties of its own, selecting it will show the properties of the **Behavior Tree** in the **Details** panel, where you can set the Behavior Tree's **Blackboard Asset**.



Aside from Root Nodes, there are four types of Behavior Tree nodes:

Node Type	Description
Composite Nodes	These are the nodes that define the root of a branch and the base rules for how that branch is executed.
Task Nodes	These are the leaves of the Behavior Tree, these nodes are the actionable things to do and don't have an output connection.
Decorator Nodes	Also known as conditionals. These attach to another node and make decisions on whether or not a branch in the tree, or even a single node, can be executed.
Service Nodes	These attach to Composite nodes and will execute at their defined frequency as long as their branch is being executed. These are often used to make checks and to update the Blackboard. These take the place of traditional Parallel nodes in other Behavior Tree systems.

Behavior Tree Node Instancing Policy

Behavior Tree Nodes (referred to here as "nodes") exist as shared objects, meaning that all agents using the same Behavior Tree will share a single set of node instances. This improves CPU performance while reducing memory usage, but also prevents nodes from storing agent-specific data. However, for cases where agents need to store and update information related to a node, Unreal Engine provides three solutions:

Instancing Nodes

The node's `bCreateNodeInstance` variable, when set to `true`, will grant each agent using the Behavior Tree a unique instance of the node, making it safe to store agent-specific data at the cost of some performance and memory usage. Certain Unreal Engine node classes, including `UBTTask_BlueprintBase`, `UBTTask_PlayAnimation`, and `UBTTask_RunBehaviorDynamic`, use this feature.

Storing on the Blackboard

A common solution is to store variables on the Blackboard. To do this, expose the name of the variable from your node, then fetch and store the Blackboard Key using that name during the node's initialization. You can then use the Blackboard Key to get and set the variable's value on your agent's Blackboard instance. This method supports variables of `bool`, `float`, `FVector`, `int32`, `enum` (stored as `uint8`) and `UObject*` types.

Storing on the Behavior Tree Node

You can create a custom struct or class to store variables inside the node's memory. The `UBTTask_MoteTo` class, for example, uses `FBTMoveToTaskMemory`. You can find the following code in `BTTask_MoteTo.h`:

```
struct FBTMoveToTaskMemory
{
/** Move request ID */
FAIRequestID MoveRequestID;

FDelegateHandle BBObserverDelegateHandle;
FVector PreviousGoalLocation;

TWeakObjectPtr<UAITask_MoveTo> Task;

uint8 bWaitingForPath : 1;
uint8 bObserverCanFinishTask : 1;
};
```

Copy full snippet

Many virtual functions in `UBTNode` take a `uint8*` parameter to the node's memory. This parameter indicates a block of memory allocated for an agent, the size of which will be returned by your overridden version of `GetInstanceMemorySize`. Nodes will allocate this amount of memory for each agent and will store this memory in a single contiguous block to optimize performance. However, this memory is not part of the UObject ecosystem, not part of Unreal Engine's reflection system, and is not visible to Garbage Collection. Because of this, `UPROPERTY` support is unavailable, and `TWeakObjectPtr` is recommended to store any `UObject` pointers that you may need.