The Behavior Tree Editor provides an intuitive user interface for developing complex behavior trees in high-level language.

This documentation is intended to provide users with a brief technical overview of Behavior Tree memory management, followed by information on creating and implementing custom nodes in C++.

Markup Language (XML)

On CRYENGINE, Behavior Trees are described using XML and are hot-loaded every time the user jumps into the game via the Editor.

Memory Model

Behavior Trees maintain a relatively small memory footprint by sharing immutable (read-only) data between instances of the same tree, and by only allocating memory for objects/activities that are necessary for a current situation.

This memory is split into two categories namely, Configuration Data and Runtime Data.

Configuration Data

Configuration Data refers to that data which stays immutable, i.e., never changes during the execution of a Behavior Tree. It contains the following information related to the configuration of the Tree:

- Events
- Variables
- Timestamps
- Event Handles
- A "skeleton" of the Behavior Tree, with specific configuration data for each node.

In code, this Configuration Data lives in a structure called the Behavior Tree Template that can only be modified when the Tree isn't running (i.e., in "Edit" mode). While Events, Variables, Timestamps and Event Handles can also be changed via the Data Definitions block of the Behavior Tree Editor, the "skeleton" can be modified by changing the structure of the tree and node parameters via the BTE.

Memory for Configuration Data is allocated from the level heap.

It is freed on level unload when running the game through the Launcher, or when the player exits Game Mode and goes back to "Edit" mode, i.e., the Behavior Tree Editor.

Runtime Data

When an AI agent is spawned, a Behavior Tree Instance is created and associated with the agent.

Its responsibility is to execute the Behavior Tree, based on the latest version of the Behavior Tree Template provided by the code of Behavior Tree Editor. It could be said that the Behavior Tree Instance runs the Behavior Tree with the provided Configuration Data.

The Tree is run in the following manner:

- The active nodes (nodes that are visited by the agent during execution of the Tree) are spawned.
- When a node is spawned and ticked for the first time, a Runtime Data object is allocated to the node and the AI agent.
- The Runtime Data object persists for as long as the agent keeps visiting the node (which can be for several frames), and is freed as soon as the agent leaves the node.

This Runtime Data lives in each node of the tree that is visited and, as opposed to Configuration Data, is mutable and very likely to change during the execution of a Tree.

Memory for Runtime Data is allocated from a bucket allocator.

This is to minimize fragmentation; since Runtime Data is normally of a few bytes, and is frequently allocated and freed, it would be hard to find contiguous blocks of memory for large chunks of data. The bucket allocator is cleaned up on level unload.