Behaviour Tree Reference Guide

1 An Overview of Behaviour Trees

Behavior Trees in Robotics and AI defines Behaviour Trees as "a way to structure the switching between different tasks in an autonomous agent, such as a robot or a virtual entity in a computer game". BTs occupy a similar place to FSMs (Finite State Machines) as a control structure in that both are used to represent the decision-making process and the sequence of actions or states an entity can take based on certain conditions or inputs.

Behaviour Trees or HFSMs (Hierarchical State Machines) were initially created for controlling game AI and NPCs (non playable characters) as a way of better modelling complex behaviour and creating more interesting scenarios and interactions for the player. More generally, BTs can be used to express both modelling and implementation characteristics for any autonomous agent system, including collaborative and reactive behaviours.

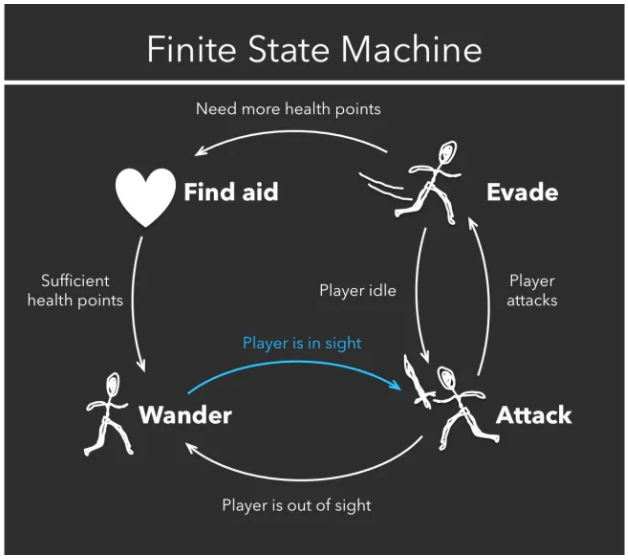


Figure 1: Finite State Machine for a simple video game

BTs fundamentally differ from FSMs in that they have an inherent hierarchi-

cal structure which implies a certain order of operation. They are represented as a tree connected graph where nodes can be either roots, i.e. they have no parent nodes above them, or leafs or non-leafs, a leaf node has no child nodes beneath them. Henceforth we will also be using the BT notation for the BehaviorTree.CPP framework to be discussed further in the next section.

Leaf nodes usually imply a certain behaviour which is an action taken by the agent, such as picking up an object or interacting with the environment. When executing said action they will return a Tick to its parent indicating its current state. Ticks are what causes state transitions inside a BT and they can either be Running, as in the child node is currently executing its action, or Success or Failure implying the result of the action being completed.

Leaf nodes are a type of a execution node which are classified as either Action or Condition nodes, whereas Condition nodes return Success if a certain predicate holds, Action nodes return Success if their underlying behaviour finishes successfuly.

Non-leaf nodes are subdivided into 3 different types and mostly imply making a certain decision about its child nodes. Specifically those are:

- Sequence Nodes will execute its children nodes from left to right until all of them return a Success Tick. If any child returns a Failure or Running tick the node will propagate the same tick upwards. Therefore it returns Success iff all its children return Success.

- Fallback Nodes will execute its children nodes from left to right until any child returns Success or Running then it will return Success or Running. Therefore it only returns Failure iff all its children return Failure.

- Decorator Nodes have a single child and alter the Tick of its child according to a user specified rule. For instance one could have an inverting decorator that only returns Success if its child returned Failure.

In the context of robotics BTs have also been widely used precisely because what makes BTs useful for autonomous agents is also extremely useful when incorporated into robotics systems. That is, robotic systems have classically been built on top of modular feedback control structures which naturally lend themselves quite well to a BT behaviour. It's also important to note that a hierarchy of tasks is fairly common in robotics as complex tasks can often be decomposed into a sequence of simpler tasks which also aligns quite well with the philosophy behind BT development.

2 BT.CPP Framework

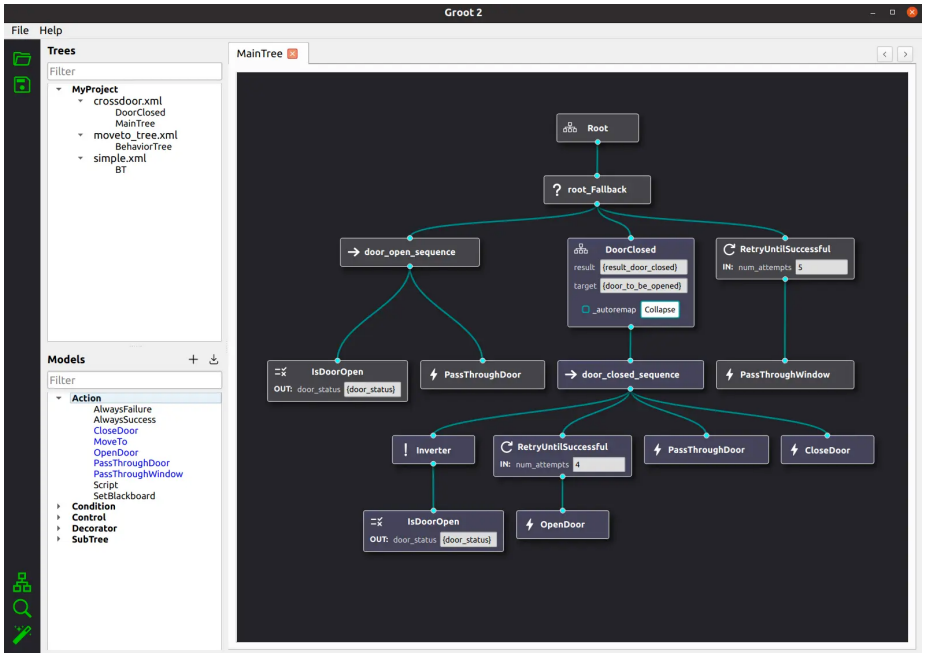


Figure 2: Picture of a behavior tree for a robot to open a door.

BT.CPP is one of the most used frameworks for the development of behaviour trees and was also created by the same authors as the Behavior Trees in Robotics and AI book

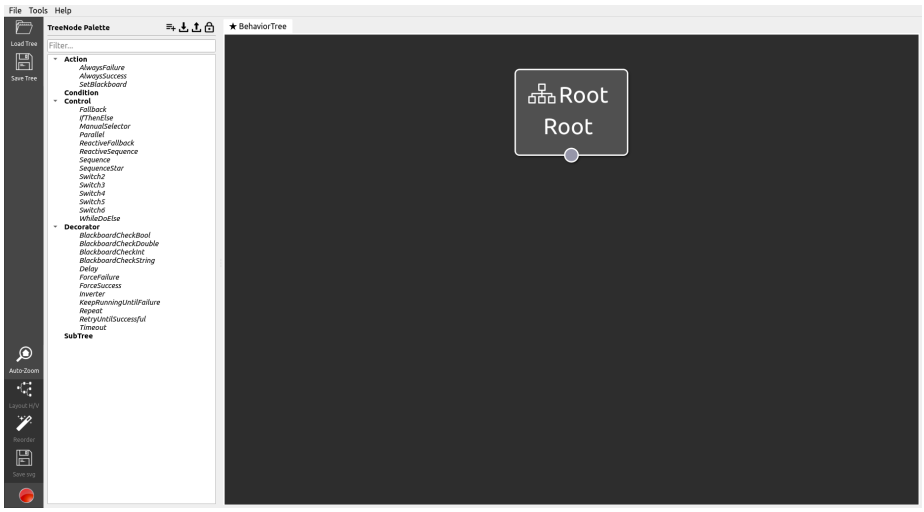
2.1 Nodes and Trees

Nodes are a way to express behaviours upon receiving an activation condition (a tick). Designers often use behaviour for both dynamic and static behaviours in mission engineering, that is BTs can be composed prior to the mission as well as created and assigned at runtime.

Remember from the previous section nodes can be either control nodes (Sequence, Fallback or Decorator) or Action Nodes (Condition or Action).

2.2 Groot

Groot is a behaviour tree visualising tool that lets us create and edit BTs as well as monitor them in real time. After installing run it with the *Groot* command. You should then see the following screen:



Drag and drop nodes from the left sidebar to place them in the tree. On the side you can also Load and Save trees. Often we need to develop behaviours distinct from the default behaviours provided by Groot, in order to do that we can create our own action/control nodes.

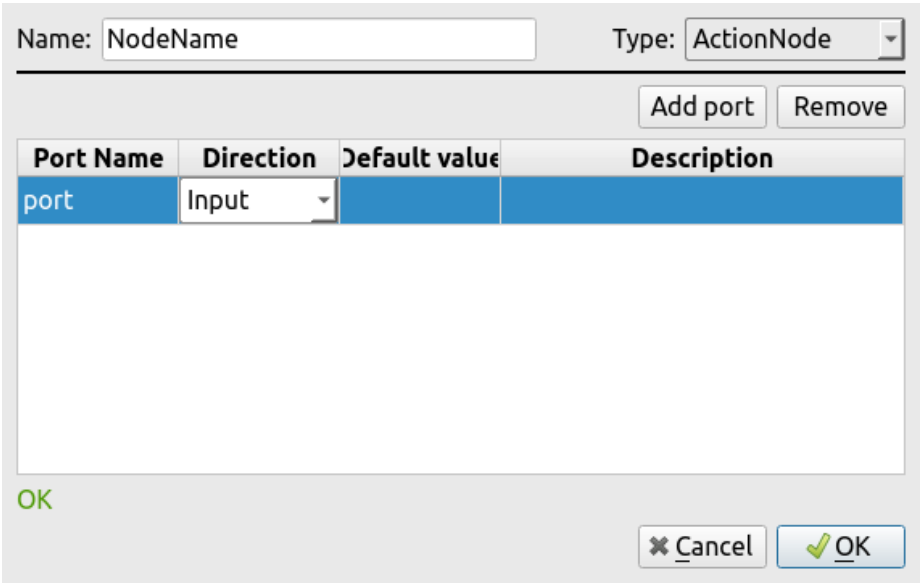
In order to create a custom node you can click the 3 bars icon next to the TreeNode Palette. You should then see the image below:

Make sure to name your node a descriptive name so others can understand its function.

3 Final Considerations

With this basic reference guide you should have all the necessary knowledge to perform simple modelling tasks using behavior trees.

If you want to delve deeper into the development of BTs we recommend the Behavior Trees in Robotics and AI Book by Michele Colledanchise, as well as the official documentation for the BT.CPP_v3 Framework.


Name:
NodeName
Type:
ActionNode
Add port
Remove
Port Name
Direction
Default value
Description
port
Input
OK
Cancel
OK