inMind Academy
Robotics Track

Behavior Tree Research Paper

Written by
Alexa Fahel

Submitted on
20 - 3 – 2025

Submitted to
Mr. Ziad Saoud

Table of Contents

Table of figures

Introduction

Behavior Trees (BTs) are a flexible framework used to model decision-making in AI systems. They organize actions and conditions into a hierarchical structure, offering scalability, modularity, and ease of maintenance [1]. BTs are widely used in robotics, video games, and autonomous systems, providing an efficient alternative to traditional models like Finite State Machines. This report explores the structure, benefits, and applications of BTs.

Part 1: Research Component

a. Definition and Structure

1) What are behavior trees

Behavior trees are a hierarchical structure used mostly in AI, especially in game development, autonomous systems, and robotics model decision-making processes. They organize the behaviors into a tree made up of "nodes" which is traversed in a specific order to determine the system's behavior.

2) Elements of a Behavior Tree

a) Behavior trees: Designed to be traversed in a particular sequence until a terminal state is achieved
b) Leaf Nodes: Each leaf node performs an action, whether a simple check or a more complex one. Leaf nodes are where the BT interfaces with lower-level code specific to the application being worked on.
c) Internal Nodes: They are non-leaf nodes that evaluate the state of their child nodes and apply their own rules to determine which node shall be expanded next.

3) Behavior Tree Terminology

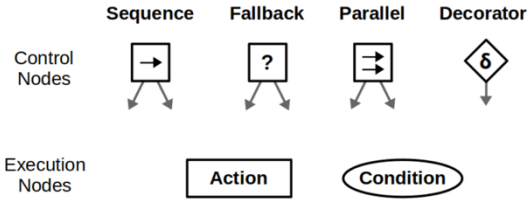

Figure 1- Behavior Tree Nodes

Ticks: Discrete update steps in which behavior trees are executed. After a node ticks, it returns a status (Success, Failure, or Running) to the parent node.

Control nodes are internal nodes that determine the way the BT is traversed based on the status of their children. These latter can either be execution nodes or other control nodes.

Referring to figure 1, sequence, fallback, and parallel may have any number of children, yet they differ in how they process them.

Decorator nodes always have a single child and modify its behavior according to a custom-defined policy.

Execution nodes also known as the leaves of the BT, can either be *Action or Condition nodes*.

Condition Nodes can only return success or failure within a single tick.

Action Nodes can span multiple ticks and may return Running until they reach a terminal state. For example:

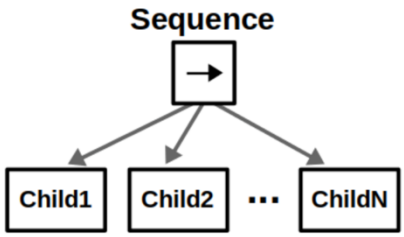

Figure 2- Sequence Node

Sequence nodes execute children in order until one child returns Failure or all return Success.

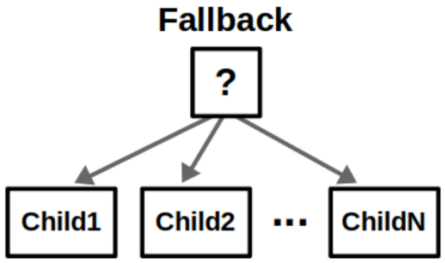

Figure 3- Fallback Node

Fallback nodes execute their children sequentially until one returns Success or all children return Failure. These nodes are essential for creating recovery behaviors in autonomous agents.

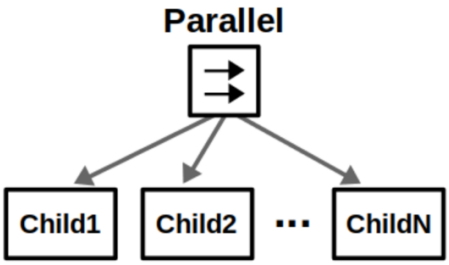

Figure 4- Parallel Node

Parallel nodes execute all their children in "parallel", each child node is ticked individually in order during each tick. Parallel nodes return Success when at least M child nodes succeed (M is between 1 and N), and failure when all children nodes fail.

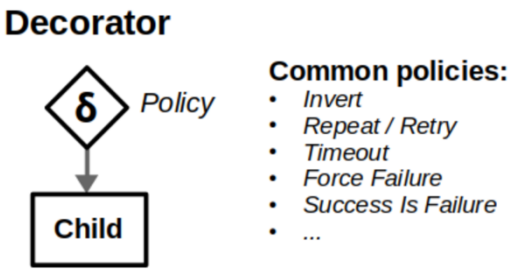

Figure 5- Decorator Node

Decorator nodes modify a single child node according to a custom policy. Every decorator has its own set of rules for altering the status of the "decorated" node.

For example, an "invert" decorator will swap success with failure and vice versa.

4) Comparison with other decision-making architectures

a. Finite State Machines

Finite State Machines (FSM) are "the most basic mathematical models of computation" [2]. They consists of a set of states, transitions and events to perform a grab-and-throw task, as seen in figure 6. This discussion applies to all control architectures based on FSMs including Mealy and Moore machines.

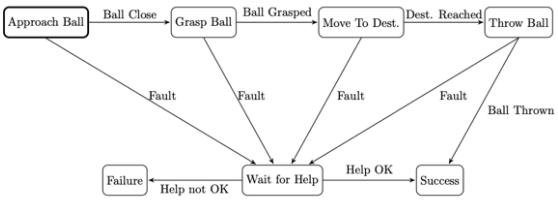

Figure 6- FSM graphical representation

FSM are used due to their three main advantages:

1. *Common structure*: they are utilized in various areas of computer science.

2. *Intuitive*: the design is straightforward.

3. *Simple to implement*: easy to code and integrate into systems.

FSMs compromise a list of disadvantages, the reactivity. Modularity tradeoff sheds the light on the challenges that arise in reactive systems:

1. *Maintainability:* adding or removing states requires re-evaluating potentially a large number of transitions

2. *Scalability*: with many states it becomes difficult to modify both for humans and computers

3. *Reusability:* transitions depend on internal variables making it challenging to reuse the same sub-FSM across multiple projects or systems.

Theoretically, anything can be expressed as a BT, FSM or any other abstraction, yet each model compromises its own disadvantages and advantages,

Comparing BTs and FSMs, there is a tradeoff between modularity and reactivity.

BTs are easier to modify and compose while FSMs are better at designing reactive behavior.

b. BT libraries in Python and C++

1. py_trees: https://github.com/splintered-reality/py_trees

2. BehaviorTree.CPP: https://github.com/BehaviorTree/BehaviorTree.CPP

b. Applications

1) Robotics

Use case: In autonomous devices like self-driving cars, BTs are used to manage behaviors like obstacle avoidance, navigation, emergency responses... [2].

For example, a BT might have a Sequence node for the car's primary mission and if it detects an obstacle along the way it could trigger a Fallback node to switch to another behavior like avoiding it or stop and wait [3].

2) Video Games

Use case: BTs control the actions and the decision-making of NPCs in video games [4].

For example: In a strategy game, an enemy AI could have a BT with nodes like "Patrol Area," "Chase Player," and "Attack Player." These nodes might be controlled by a Sequence node where the enemy first checks if the player is within range; if not, it continues patrolling, and if the player is detected, it switches to chasing and attacking.

Part 2: Theoretical Application Design

a. Grasp & Place Behavior Tree

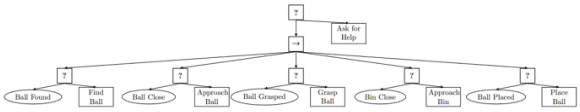

Figure 7- Grasp & Place BT

The aim of the following Behavior tree is to have a robot look for a ball, approach it, pick it up, approach the bin, and place it inside. The tree is made of many fallback nodes each having a condition and action noe as children as well as a sequence node to go through these fallback nodes in order and another fallback node on top to ask for help if our behavior tree didn't execute as expected.

To elaborate, the tree starts with a fallback node that begins by checking if a ball is found, if not, then the robot searches for it. If the robot finds the ball then he proceeds to check if the ball,l is far away or not and approaches it if it is. Then if the ball is not grasped the robot picks it up, and if the bin is far away the robot approaches it. Finally the robot places the ball in the bin if he hasn't done so already. If at any point any of the low level fallback nodes return failure, then the tree goes to the "ask for help" action which waits 5 seconds for a human to fix the robot then returns success. In reality this action should wait for more than 5 seconds but I set it as such so as not to have idle time in my code.

b. Cross Door Behavior Tree

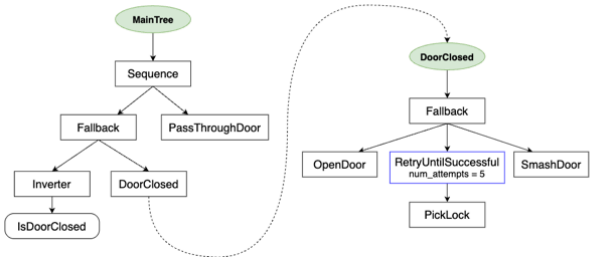

Figure 8- CrossDoor BT

The CrossDoor behavior tree example models the decision-making process for an agent trying to pass through a door. If the door is open, the agent simply passes through. If the door is closed, the agent first attempts to open it, then tries picking the lock up to five times, and finally resorts to smashing the door open if necessary.

Looking more closely, we first begin by checking if the door is open, if it is we pass through, otherwise we move on to the DoorClosed subtree. This subtree consists of a fallback node and several actions. First we try opening the door, then if that didn't work we keep trying to pick the lock 5 times untill we get a successful attempt, and if not then the door is smashed open.

Conclusion

In conclusion, Behavior Trees (BTs) provide a flexible and scalable solution for managing complex decision-making in AI systems. Their modular structure enhances maintainability and reusability, making them ideal for applications like robotics and video game AI. Compared to other architectures like Finite State Machines, BTs offer better scalability and adaptability, allowing for more dynamic and efficient behavior management.

References

[1] Castro, S. (n.d.). *Introduction to behavior trees - Robohub*. https://robohub.org/introduction-to-behavior-trees/

[2] Colledanchise, M., & Ogren, P. (2022). *Behavior Trees in Robotics and AI*. https://arxiv.org/pdf/1709.00084

[3] Purplesquirrelusa. (2024, September 15). *Unlocking the Power of Behavior trees in AI and Robotics: A guide by Curate Consulting Services*. Curate Partners Merch. https://curatepartners.com/blogs/skills-tools-platforms/unlocking-the-power-of-behavior-trees-in-ai-and-robotics-a-guide-by-curate-consulting-services/#:~:text=Autonomous%20systems%2C%20including%20autonomous%20vehicles,best%20route%20to%20its%20destination.

[4] Sekhavat, Yoones. (2017). Behavior Trees for Computer Games. International Journal on Artificial Intelligence Tools. 26. 10.1142/S0218213017300010.