Behavior Trees: Robots & Games

An Introduction
to
Behavior Tree
Basics
for DPRG

Rudyard (Rud) Merriam
9 November 2021

Basics, Basics, & Basics

Basics because there are many changes, differences, and improvements (?) across libraries

Goal is to provide context for further exploration

Behavior Tree Origins

- Developed in gaming for Non-Player Characters (NPC)
- Adopted for controlling robots
- Don’t confuse with data structure Binary Tree: B-tree
- Unfortunately terminology is inconsistent in literature and implementations

Behavior Tree Overview

- They are a “directed rooted tree”
- Searched depth first, i.e. left most nodes processed to leafs
  - Left nodes are higher “priority”
- Leaf nodes are *action* or *condition* nodes
  - No child nodes
- Internal nodes are *control nodes*
  - Must have at least one child node
- Nodes return: *success, failure, running*

Single Nodes

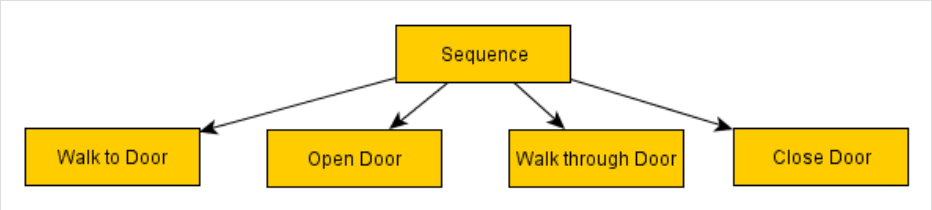
- Action – leaf node and no child
  - “Does something”, e.g set driving speed
- Conditional – leaf node and no child
  - A test of information
  - Never returns *running*
- Decorator – has a single child
  - Changes result of child
  - Inverter, Succeed, Fail, etc

Composite Nodes

- Composite nodes have one or more child nodes
- Results from child nodes determines their result
- Process each child generally left to right
  - There are some that do randomly or in parallel

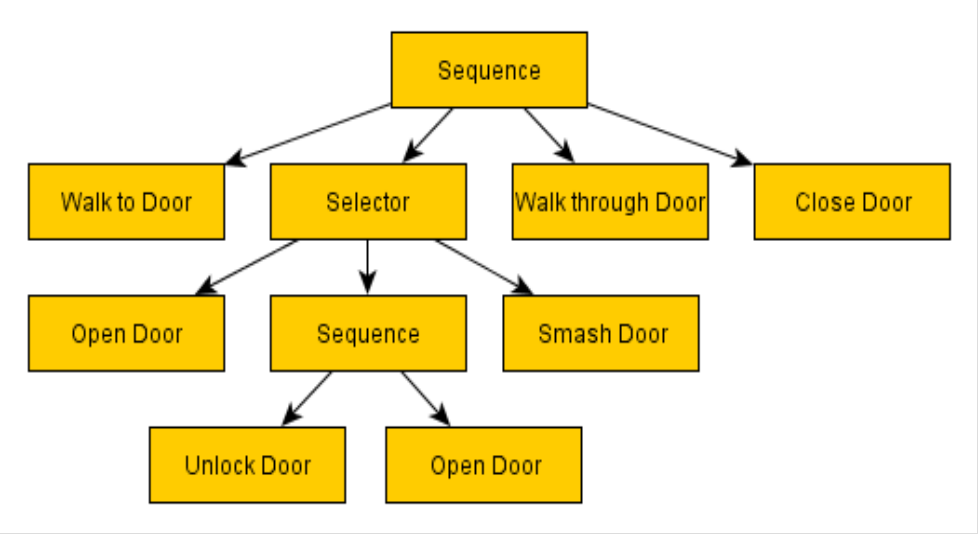
Sequence Node

- Executes each child in order
- Is an AND operation
  - All children must succeed
- Fails if a child fails
- Succeeds when all succeed
- Returns *running* if a child does

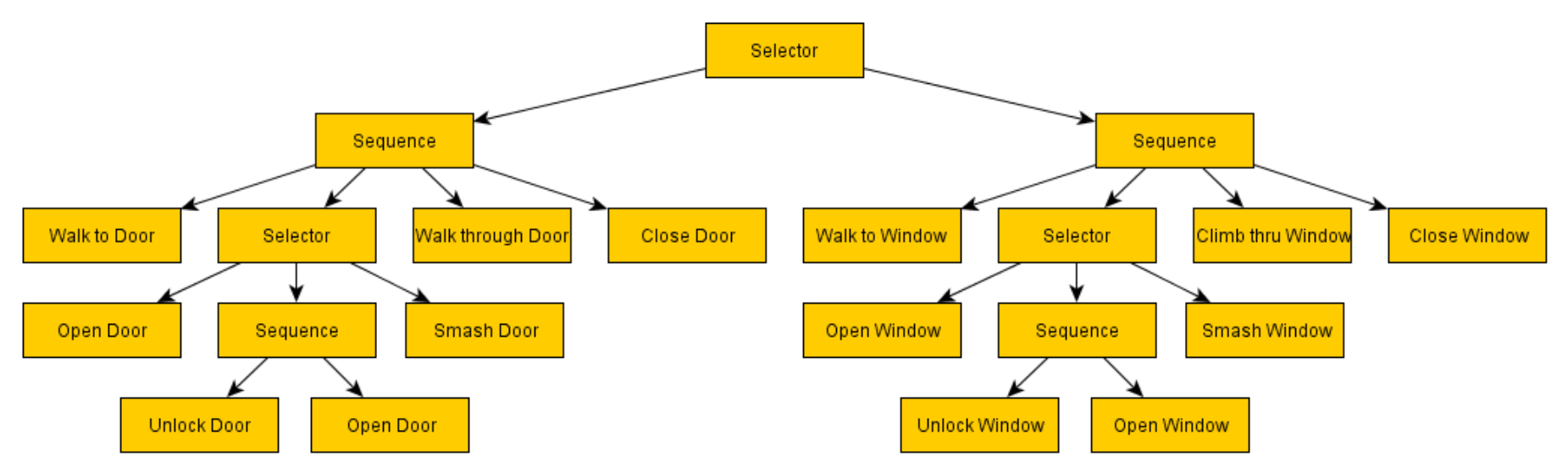


Selector Node

- Executes each child in order
- Is an OR operation
- Succeeds if any child succeeds
- Fails when all fail
- Returns running if a child does
- Equivalent to *subsumption*
  - Except stops when a child succeeds



Tree Expansion (Relatively) Easy



Memory Selector and Sequence

- Different names depending on implementation
  - Original seems to be Selector* / Sequence*
  - I use MemSeqNode / MemSelNode
- Node remembers left nodes that succeed
  - These nodes are not visited again
- There is usually a *reset* capability to clear memory
  - Required to restart a behavior pattern once completed

Running Nodes

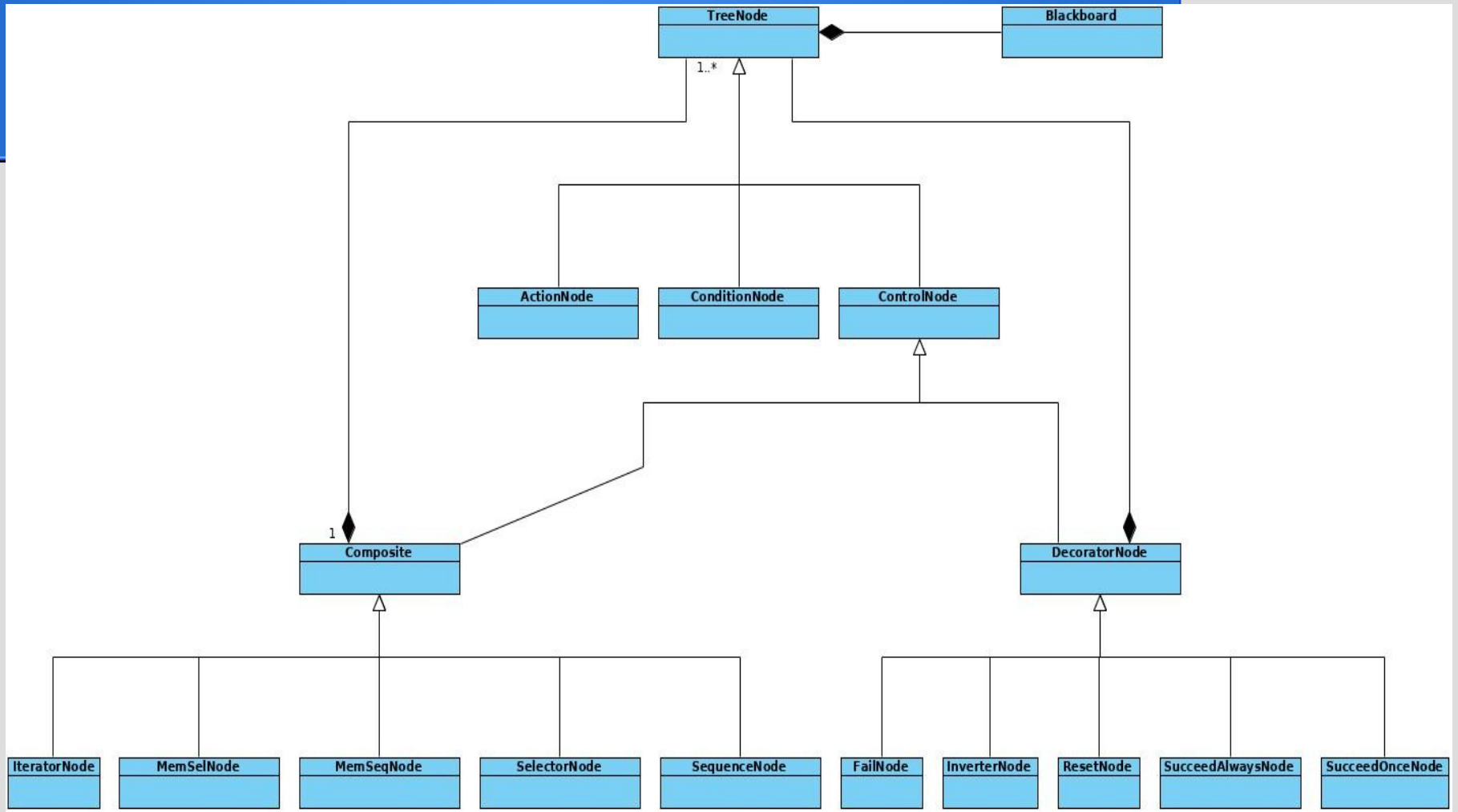
- Variations in handling running
- Original usage processed tree as usual
  - If running node reached simply called again
  - It should check if it should still be *running*
- Later changes attempt to improve performance
  - One possibility is going directly back to running node
  - Might lose reactive nature

Loop, Iterator, etc

- As names imply loops over the child nodes
- May be N times or continuous
  - Sometimes returns on M of N *succeeding*
- Need to check how it handles *running*

Parallel Node

- All children are executed “simultaneously”
  - May be implemented as randomly executing rather than in sequence
  - Sometimes implemented as separate threads
- When one child succeeds must halt other children



Code at https://gitlab.com/robot-libraries/behaviortrees

Blackboard Common Data Store

- Common approach is global (?) data in *Blackboard*
  - Action nodes read or write data
  - Condition nodes read data for decision
- General processing loop:
  1) Update blackboard from sensors
  2) Run tree reading and updating blackboard
  3) Use blackboard to update controls
- Other possibilities? (Discussion!!)

References

- Material to either elaborate or confuse
- Behavior Trees in Robotics and AI: An Introduction
  - https://arxiv.org/abs/1709.00084
- Behavior trees for AI: How they work
  - https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work
  - Series of books on game AI ; https://www.gameaipro.com/
- The Behavior Tree Starter Kit
  - https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter06_The_Behavior_Tree_Starter_Kit.pdf
- BehaviorTree.cpp: https://www.behaviortree.dev/
  - A heavy library with atypical terminology
- Java Behavior Tree and much more:
  - https://github.com/libgdx/gdx-ai/wiki/Behavior-Trees