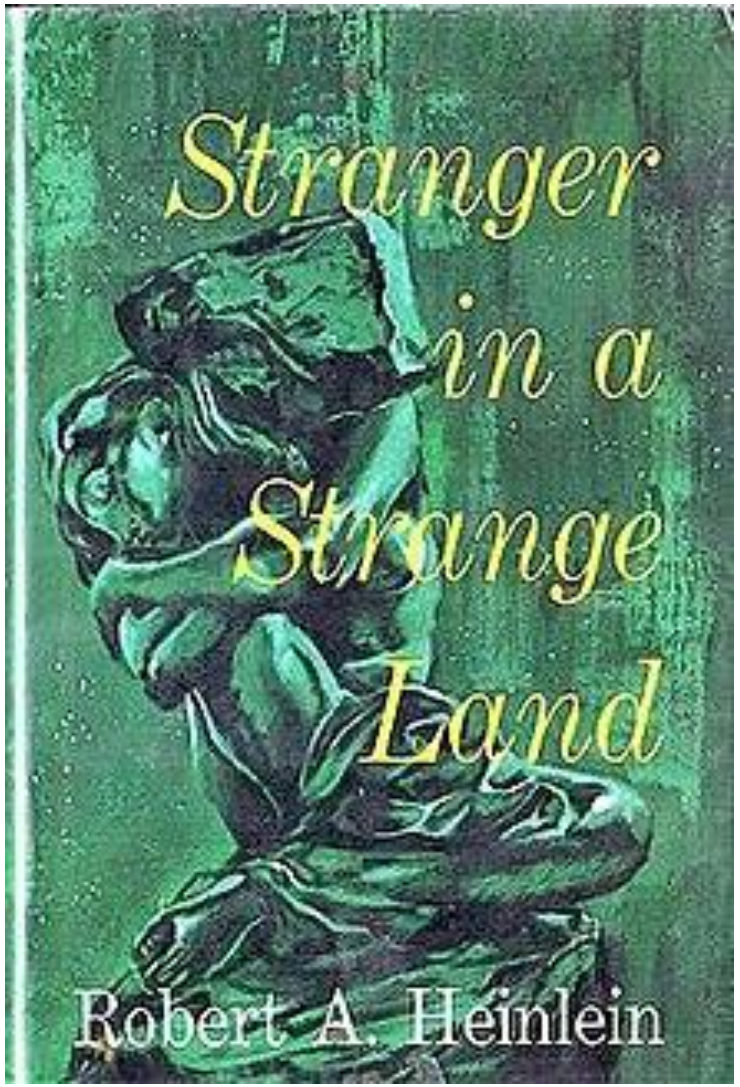
Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.

Grok: The word "grok", coined in the novel, made its way into the English language. In Heinlein's invented Martian language, "grok" literally means "to drink" and figuratively means "to comprehend", "to love", and "to be one with". This word rapidly became common parlance among science fiction fans, hippies, and later computer programmers[16] and hackers[17], and has since entered the Oxford English Dictionary.[18]



https://en.wikipedia.org/wiki/Stranger_in_a_Strange_Land

N-2&3: Decision Making, FSMs

1. How can we describe decision making?
2. What makes FSMs so attractive? What is difficult to do with them?
3. Two drawbacks of FSMs and how to fix?
4. What are the performance dimensions we tend to assess?
5. What are two methods we discussed to learn about changes in the world state?
6. FSMs/Btrees: R___ :: Planning : D____
7. When is the R__ good? When is D__?
8. H_____ have helped in most approaches.
9. What are two methods we discussed to learn about changes in the world state?

N-1: Decision Making, D-trees

1. How many outcomes does a D-tree produce?
2. What are advantages of D-Trees?
3. Discuss the effects of tree balance.
4. Must D-trees be a tree?
5. Can D-trees translate into rules? If so how?
6. How can we use d-trees for prediction?
7. What is the notion of overfitting?

Decision trees can represent any Boolean function of the input attributes

More on learning Dtrees:
https://courses.cs.washington.edu/courses/cse573/12sp/lectures/19-dtree.pdf
https://www.cc.gatech.edu/~bboots3/CS4641-Fall2016/Lectures/Lecture2.pdf

Learned D-tree: how well do they work?

- Many case studies have shown that decision trees are at least as accurate as human experts.
  - study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time; the decision tree classified 72% correct
  - British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms that replaced an earlier rule-based expert system
  - Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example

https://www.cc.gatech.edu/~bboots3/CS4641-Fall2016/Lectures/Lecture2.pdf

Reminder

- "What Makes Good AI – Game Maker's Toolkit"
  - https://www.youtube.com/watch?v=9bbhJi0NBkk&t=0s
  - https://www.patreon.com/GameMakersToolkit
  - React/adapt to the player – no learning required (authoring is)
  - Communicate what you're thinking
  - Illusion of intelligence; more health & aggression can be a proxy for smarts
  - Predictability is (usually) a good thing
    - Too much NPC stupidity can ruin an otherwise good game

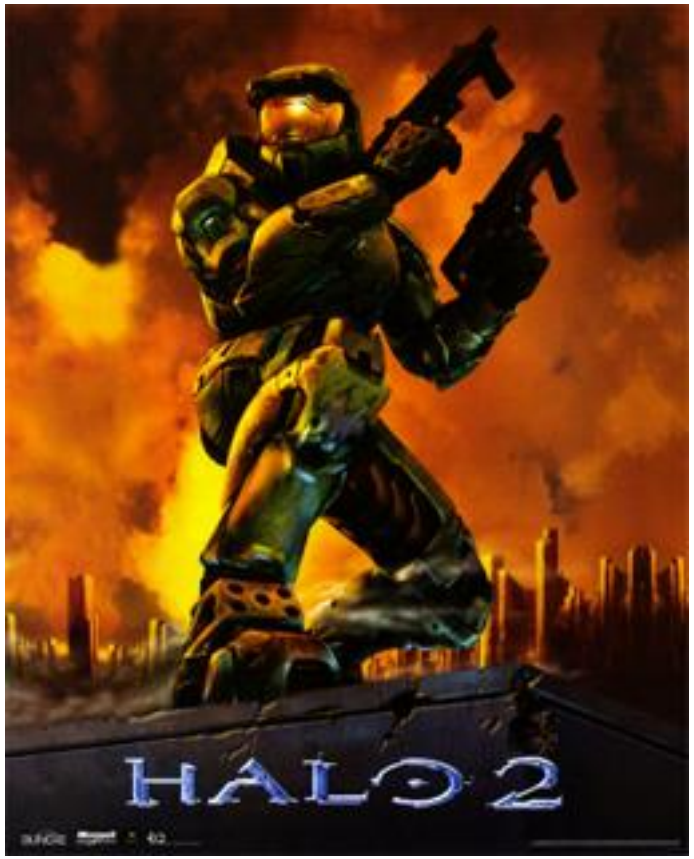
BEHAVIOR TREES (M CH. 5.4)

What if…

- We could fail gracefully?
  - If “confused” enter more and more general states
- Encode sequences of states?
  - Without having to bog each state down tracking more variables/conditions

Behavior Trees (B trees)

- Popular in subsections of industry since 2004; have reach ubiquity
  - Halo 2
  - Bioshock
  - Spore
- Easy to design
- Easy to alter
- Fail gracefully

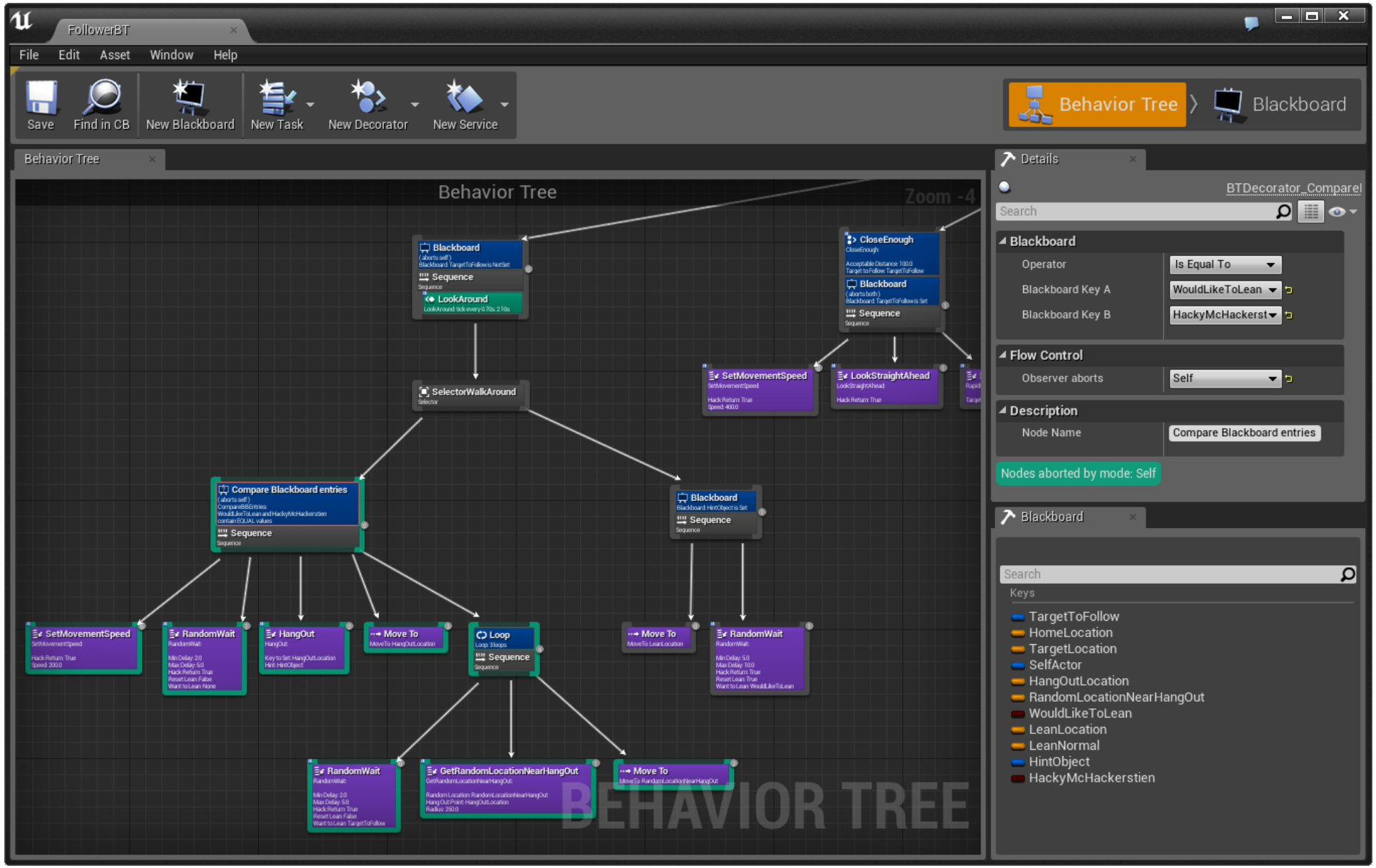


Behavior Trees

- Simple **reactive** planning that is synthesis of: HFSM, Scheduling, Planning, Action Execution
  - Mathematical Model of Plan Execution – describe switching between a finite set of tasks in a modular fashion
  - (Manually provided) tree of behaviors specifies what an agent should do under all circumstances
  - Path from root to leaf represents one course of action. All paths, all COAs
  - Search proceeds left-to-right (ie DFS)
- Decomposition allows flexibility & easy GUI integration
  - Easy to understand
  - Easy for non-programmers to create
- Aren't good in all instances… (stay tuned)
- Instead of *state*, employ *tasks*
- Composable, self contained, encourages reuse
  - Delegation of concerns – don't need to know how each sub-task implemented

Behavior Trees

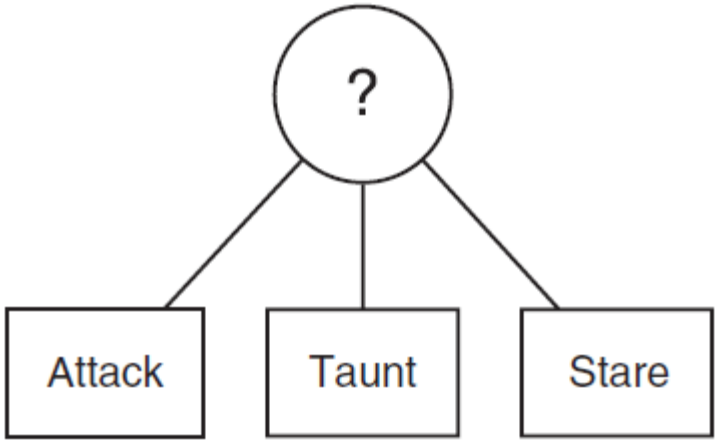
- All nodes (tasks/behaviors) return success, failure, none/running, or error
  - Behavior tree made of hierarchically connected tasks (not states!)
- Types of nodes:
  - Actions/Execution (leaf node): do something in the world
  - Conditions (leaf node): make a decision based on world condition
  - Composites/Control flow (one parent, one+ children): combine multiple tasks
    - Prioritized list: success if any child succeeds in order
    - Sequence: failure if any child fails in order
    - Sequential-looping: keep doing sequence until a failure
    - Probabilistic: choose probabilistically from set
    - One-off (random or prioritized): pick a single child randomly or with some priority
  - Decorators (one parent, one child): modify child task behavior
    - e.g UntilFail, RunLimit, Semaphore



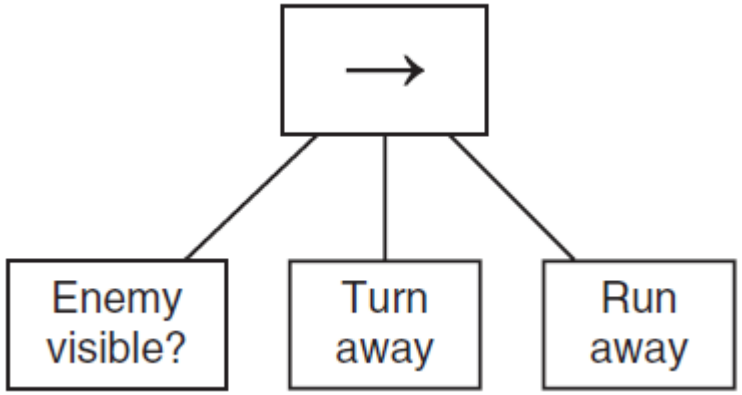
https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/index.html

https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/HowUE4BehaviorTreesDiffer/index.html

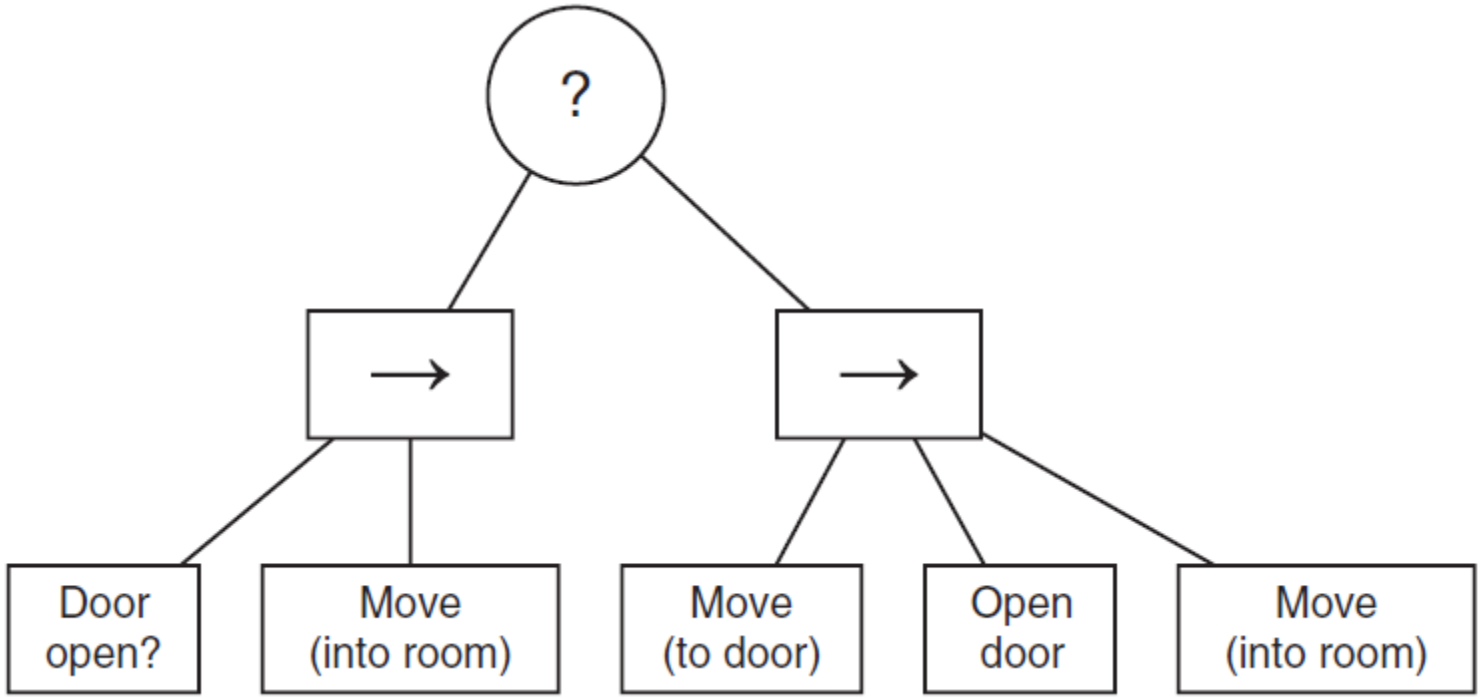
Behavior Tree Structure



M&F 5.22: Selector



M&F 5.23: Sequence w/ condition



M&F 5.25

One action per tick

- Execution of a BT starts from the root
  - Root sends ticks with a certain frequency to its child
    - Tick is an enabling signal that allows the execution of a child
  - Each node keeps track of current child to execute
  - When execution of a node is allowed, it returns execution status to the parent
    - running if execution not yet finished,
    - success if it has achieved its goal,
    - failure if it didn't,
    - error if an exception occurs (failure of code, not failure of attempted behavior)
- At start of 'tick', walk the tree to find our current node
  - If in it last frame continue, otherwise reset it
  - Alternative: keep track of executing node(s)
    - Store any currently processing nodes so they can be ticked directly within the behavior tree engine rather than per tick traversal of the entire tree

Node Types

- Leaves
  - Conditions
  - Actions
- Non-leaves
  - Composites
  - Decorators

Leaves

- Game logic
  - Library
  - custom
- Returns Success, Fail, Processing, or Error
- Init() – called first visit
- Run() –called until complete
- Parameters

Node Types

- Conditions
  - Test for some game property (e.g. proximity of player to NPC)
  - Each implemented as a task
- Actions
- Composites
- Decorators

```
Class BTCondition extends Node
{
        void run ()
        {
                if (condition met) {
                        return True
                }
                return False
        }
}
```

Node Types

- Conditions
- Actions
  - Alter game state
    - (e.g. play animation, change character internal state, run AI code, play audio sample, etc.)
  - Each is a task
- Composites
- Decorators

```
Class BTAction extends Node
{
        void run ()
        {
                if (execution conditions not met) {
                        return False
                }
                // Do whatever you need to do
                return True or False
        }
}
```

Node Types

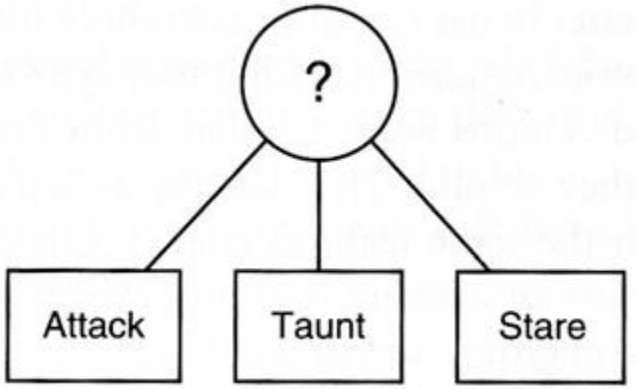
- Conditions
- Actions
- Composites
  - **Differentiates BTs from decision trees**
  - Allows for the combination of tasks without concern for what else is in the tree
- Decorators

Composite

- Composite Node:
  - One or more children
  - Sequence (AND), Selector (OR), or Random
  - Short circuiting of Boolean logic
  - Returns success or fail (based on children returns typically)

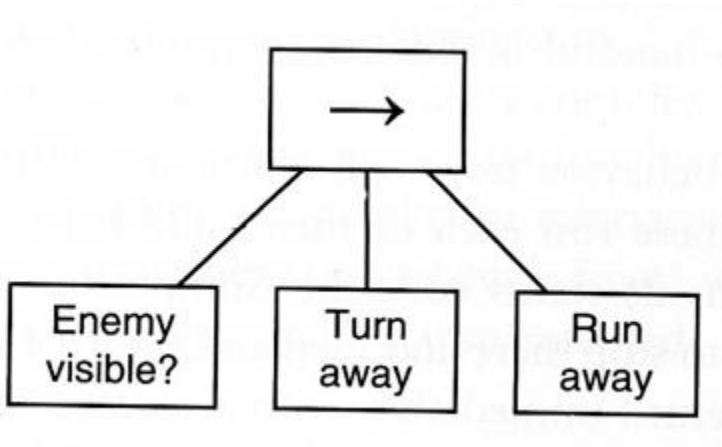
Composite Nodes: Selector

- Selector
  - Run child tasks until one of them succeeds
  - Return failure if all tasks fails



Composite Nodes: Sequence

- Selector
- Sequence
  - Series of tasks that all must succeed

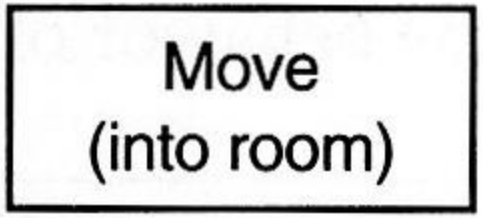


```
Class BTPriorityList extends Node
{
        children = []

        void run ()
        {
                if (execution conditions not met) do {
                        return False
                }
                for child in children do {
                        if child.run() == True do {
                                return True
                        }
                }
                return False
        }
}
```

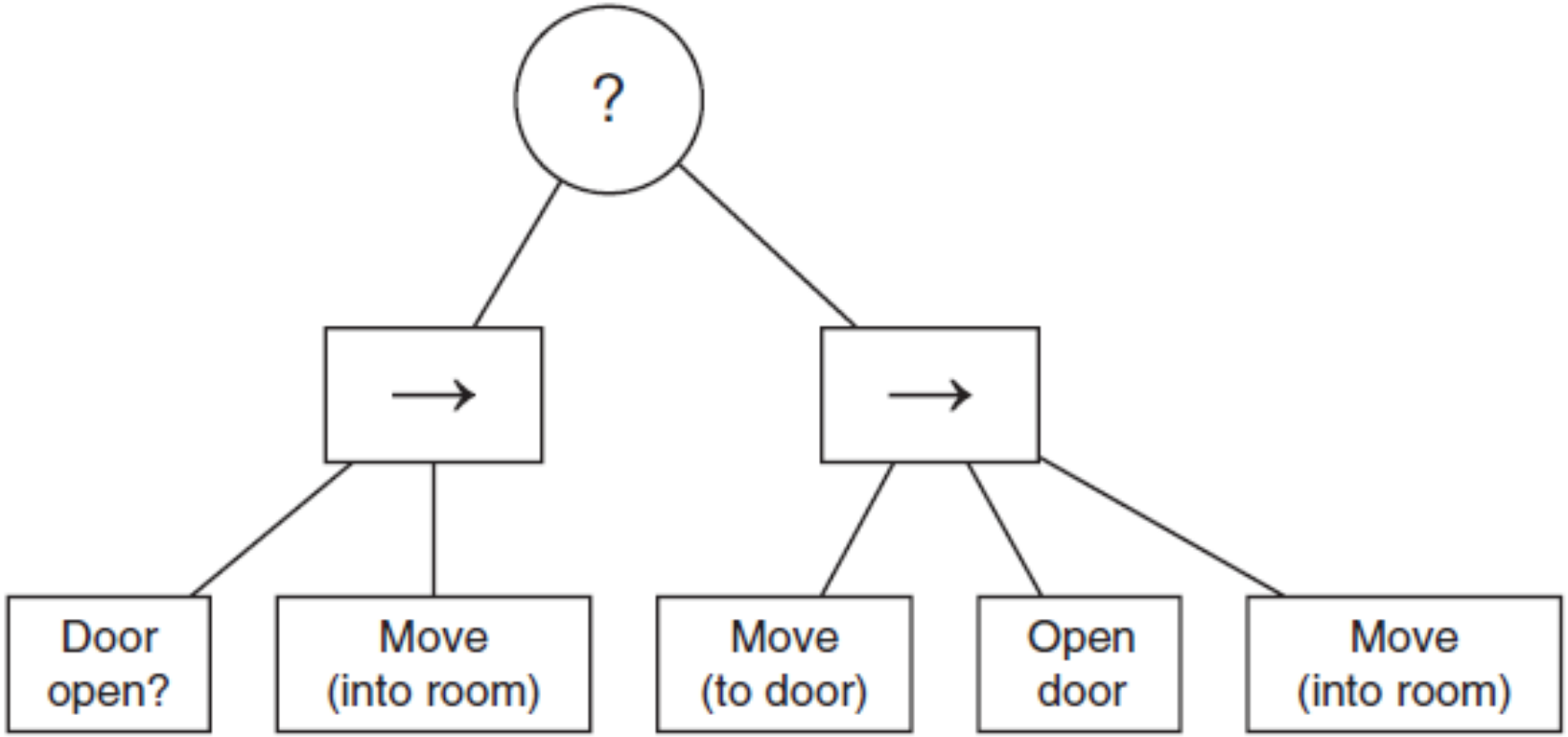
```
Class BTSequence extends Node
{
        children = []

        void run ()
        {
                if (execution conditions not met) do {
                        return False
                }
                for child in children do {
                        if child.run() == False do {
                                return False
                        }
                }
                return True
        }
}
```

Example

- Enter room where player is standing. Player may close the door.



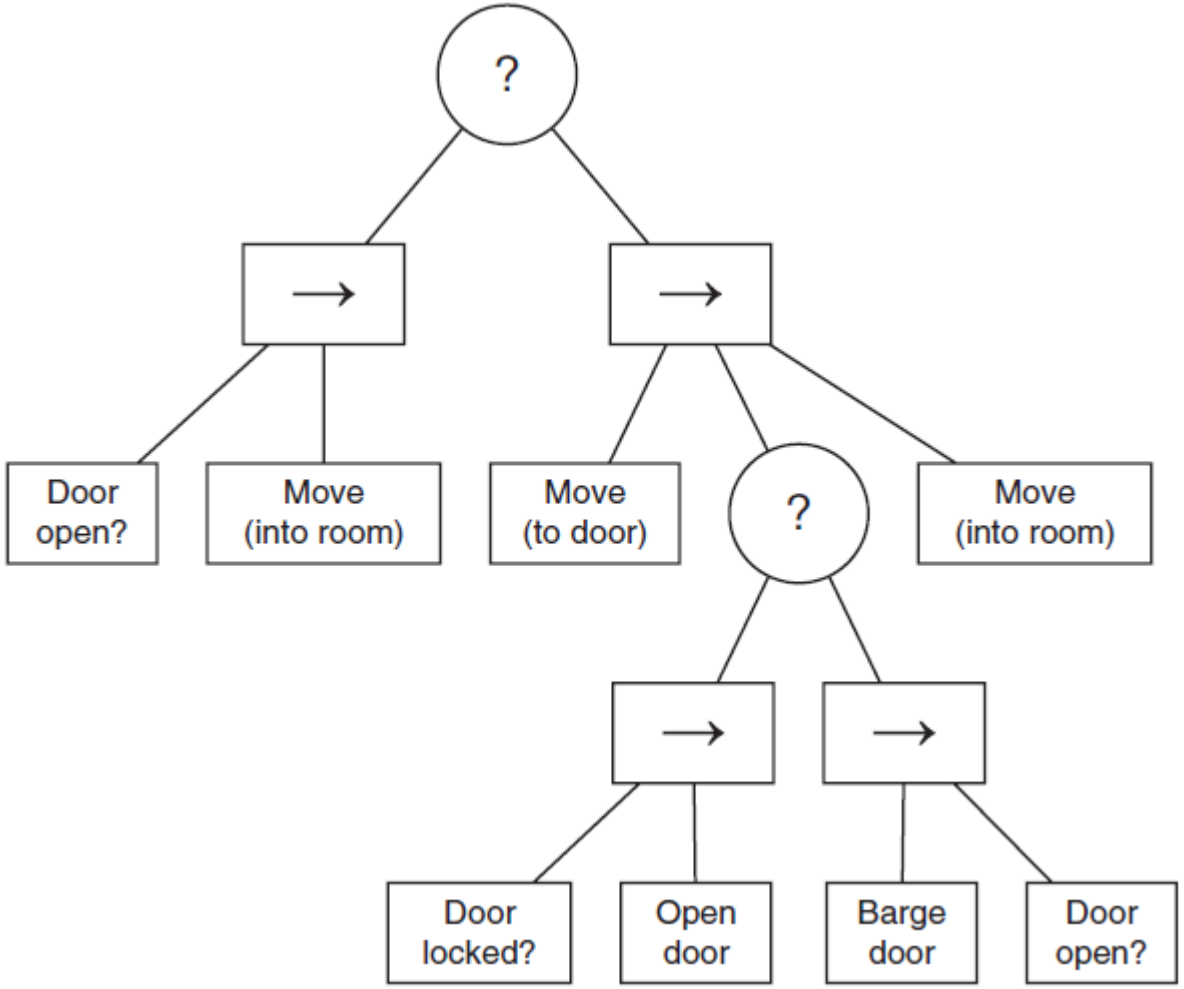
Example



M Fig 5.25

What if the door is locked?

Example



M Fig 5.27

Non-deterministic Composites

- Strict order == predictable
- We saw partial-orders help this
- Fake partial-order with random shuffle
- 2 new (sub)types of composites
  - ND Selector
  - ND Sequence
  - The original selector/sequence are deterministic (that is, totally ordered)

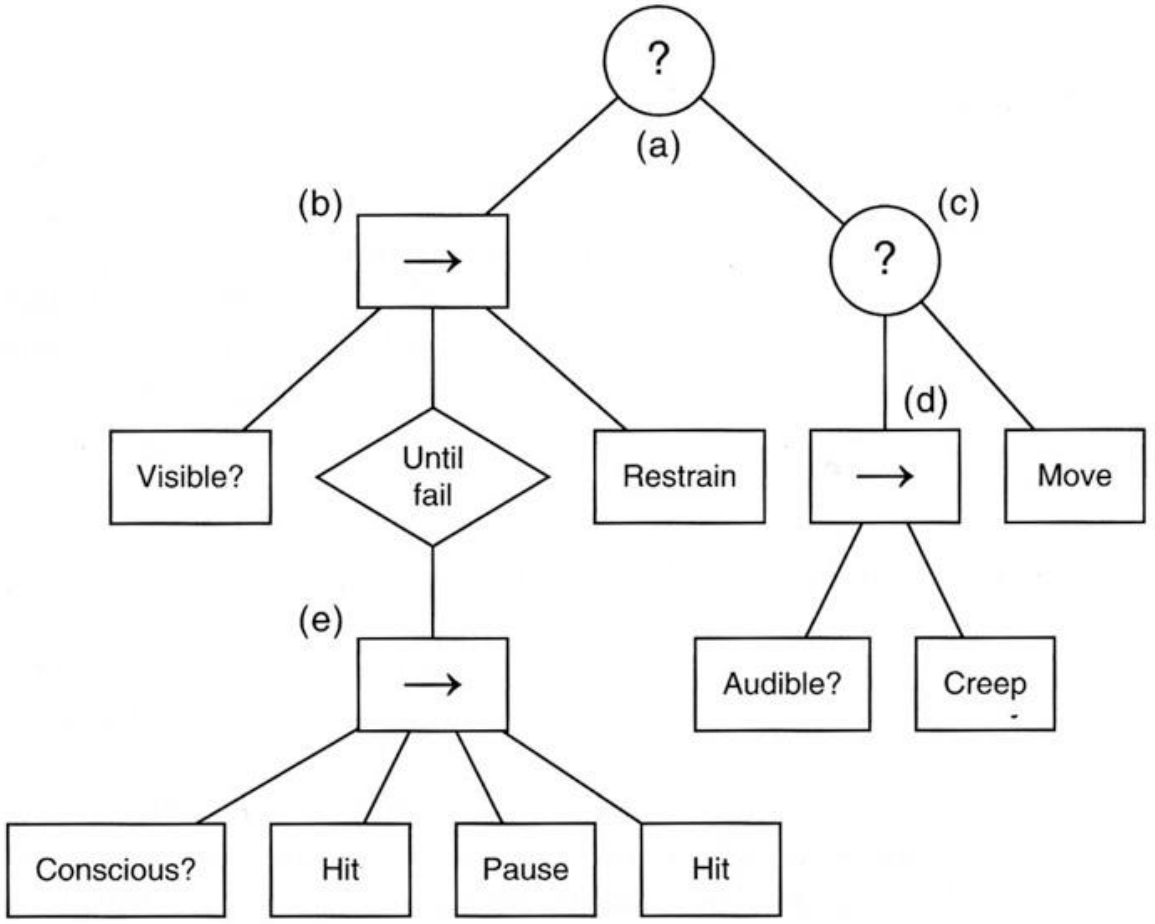
Node summary (so far)

- Conditions
- Action: leaf, alter state of game, move, play animation, etc.
- Composites:
  - Prioritized list: choose subtask, with priority given to certain “questions”
  - Sequence: do all subtasks in order
  - Sequential-looping: sequence, start over when done
  - Probabilistic: randomly choose a subtask
  - One-off: pick one subtask (prioritized or random), but never repeat the choice
- Decorators

4th node type: Decorators

- “Wraps” other nodes
- Has a single child task and modifies it in some way
  - Inverter (ie NOT)
  - Filters (allows child to run (or not))
  - Run Until Fail
  - Repeater
  - Succeeder (always true, runs child but doesn’t care about success/failure)
  - Guard Resource (semaphores)

Example

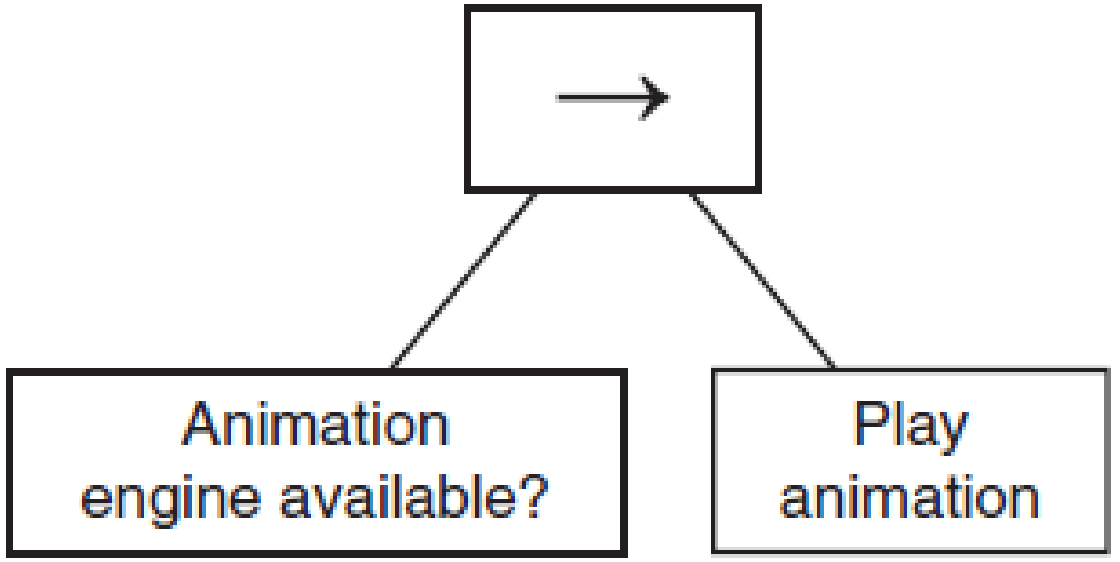


M 5.29

Semaphores

- Check for restricted resources
  - Keeps a tally of available resources and number of users
  - e.g. animation engine, pathfinding pool, etc.
- Typically provided in a language library

Guarding Resources



M 5.30

Advanced hacks

- Interrupt daemons: jump from a node to an entirely different section of the tree based on external conditions changing

- Shortcuts: jump from within one child node to another directly

More Complex Approaches

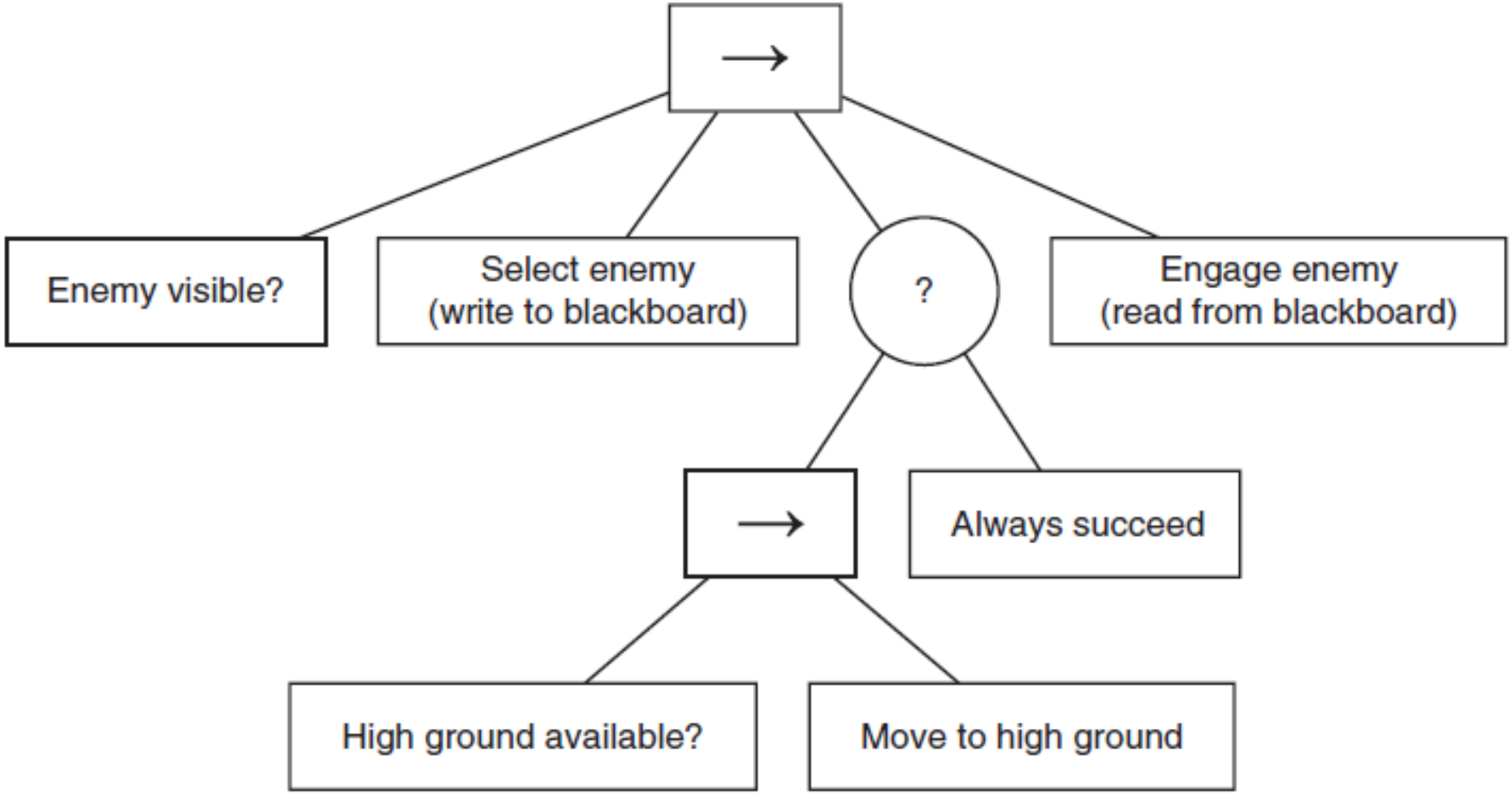
- Concurrency (tasks run on threads or via multitasking & scheduling algorithms)
  - Essential to make BTs useful (* UE4 disagrees)
  - Most common practical implementation
  - Millington codebase has example w/ cooperative multitasking
- Blackboard communication for sharing data

UE4 & Concurrent Behaviors

- "Standard" behavior trees often use a Parallel composite node to handle concurrent behaviors.
  - The Parallel node begins execution on all of its children simultaneously.
  - Special rules determine how to act if one or more of those child trees finish
- **Why not use Parallel nodes? Clarity, Debugging & Optimizations**
  - Parallel nodes can be very confusing, even for relatively simple behaviors.
  - Effectively Parallel nodes are simultaneously running a bunch of separate sub-trees, but any or all of those sub-trees may need to abort if one of them fails, or they may succeed when the others finish (whether successful or failing).
  - Parallel behaviors can be confusing even in simple cases, and with the number of options potentially available, it can become highly confusing.
  - Parallel nodes make it harder to optimize performance, especially in terms of making event-driven trees.

Source: https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/HowUE4BehaviorTreesDiffer/index.html

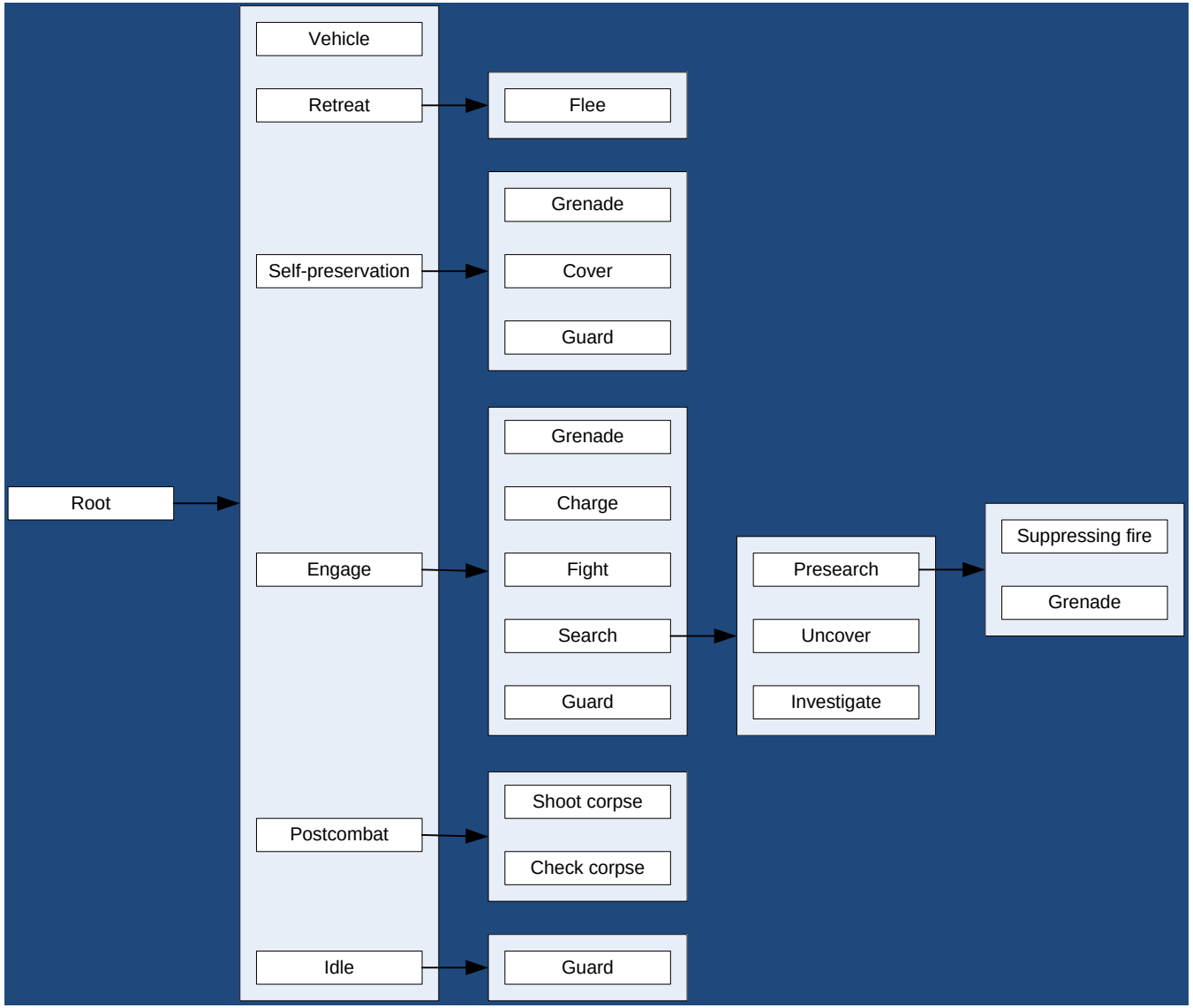
Blackboard Agents



M 5.36

https://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php

BTs in *Halo 2*



BTs in *Halo 2*

- Determining which behaviors are relevant can be costly (in terms of time)
  - Why? We're constantly checking relevancy of behaviors that are not actually running
- How can we overcome that?
  - Behavior tagging – Move commonly used checks to decision-time
  - This process effectively locks or unlocks portions of the behavior tree, which you could view as changing the structure of the tree.
  - Example: a vehicle passenger cannot access subtrees devoted to fleeing, searching, or self-preservation

BT Pros and Cons

- Cons
  - Clunky for state-based behavior
    - That is, changing behavior based on external changes
  - Isn't really thinking ahead about unique situations
  - Only as good as the designer makes it (just follows the recipes)
- Pros
  - Better when pass/fail of tasks is central
    - Sound familiar? (harder to think about state…)
  - Appearance of goal-driven behavior
  - Multi-step behavior
  - Fast (generally slower than FSMs)
  - Recover from errors
- Hybrid system may be answer
  - Adds authorial + toolchain burden

- On Maintainability:
  - Transitions in BT are defined by the structure, not by conditions inside the states.
  - Because of this, nodes can be designed independent from each other
  - When adding or removing new nodes/subtrees, it is not necessary to change other parts of the model
- On Scalability:
  - A BT with many nodes can be decomposed into smaller sub-BTrees
  - Saves the readability of the graphical model
- On Reusability:
  - Thanks to independence of nodes in BT, subtrees are also independent.
  - This allows the reuse of nodes or subtrees among other trees or projects.

Strengths as compared to…?

- FSMs vs Behavior Trees:
  - This is the strength of the behavior tree: It separates the states from the decision logic.
- Decision Trees vs Behavior Trees
  - Single common interface for all tasks/nodes means arbitrary conditions, actions, and groups can be combined together without any of them needing to know what else is in the behavior tree

IMPORTANT NOTE

- Any desired set of behaviors that can be represented in a Behavior Tree can be represented in an FSM and vice versa.

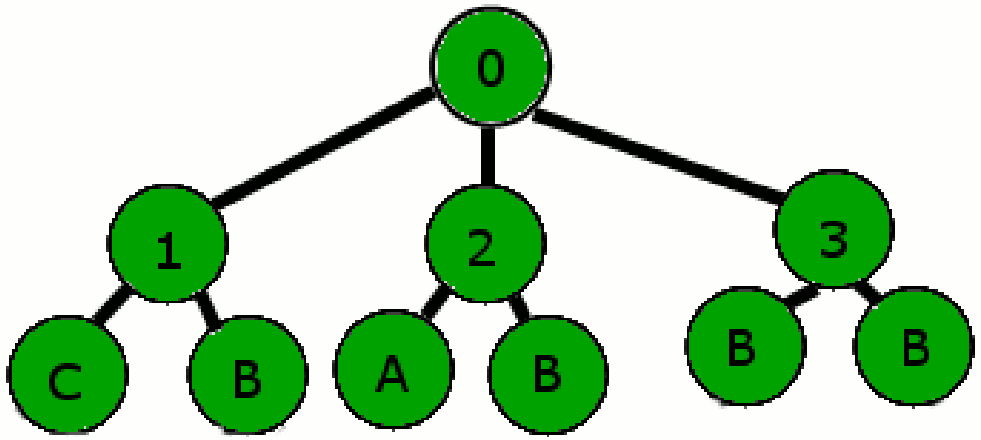
- Differences:
  - **Sequences of Actions**: FSMs require extra variable tracking to handle sequences of states
  - **Error Recovery**: FSMs require many more linkages to do what Btrees do naturally

Btree, Dtree, FSMs: Caveat Emptor

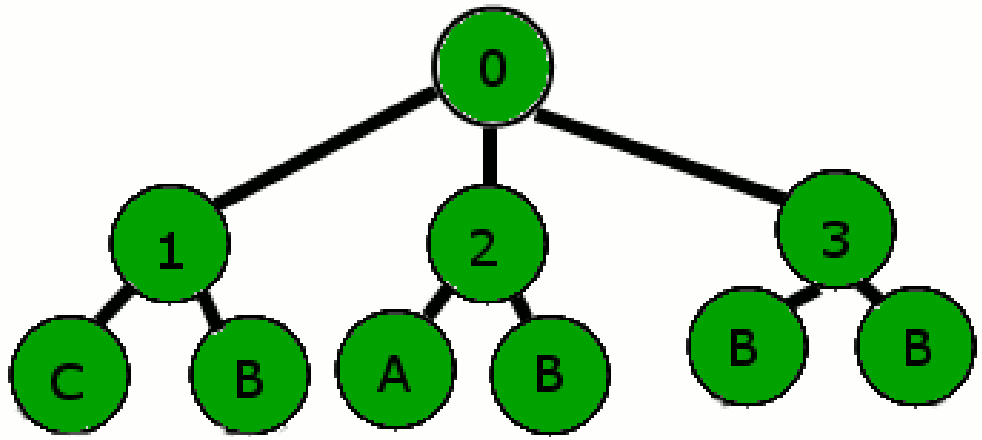
- In principle/theory… **they're equivalent**
  - There is nothing a behavior tree can do that a FSM cannot do
  - There is nothing a FSM can do that a decision tree cannot do
- While FSMs CAN represent sequences, it's awkward. Authoring/Design differences
  - BTrees make it easier to **design plan-like sequences** of actions
  - BTree **hierarchical decomposition** makes it easier to design behavior **modularly**
- Hybrid techniques possible: eg Dtree to evaluate FSM transitions
- Dtrees, Btrees and FSMs (and rules): reactive decision making
  - rely on **the ability of the designer** to create a good structure
  - can NOT respond to novel situations (**designer must anticipate all**)
  - have about same computation time (**quick responses**)
  - create enemies with **reliable patterns** of behavior (**less true for rules**)

However, in practice

- Complexity generally follows: Dtree < FSM < Btree
- Decision trees are evaluated from root to leaf, every time.
  - For a decision tree to work properly, the child nodes of each parent must represent all possible decisions for that node.
- Behavior trees handle false conditions slightly differently
  - If all of a child node's conditions are met, its behavior is started.
  - When a node starts a behavior, that node is set to 'running', and it returns the behavior.
  - A 'running' node knows to pick up where it left off
  - If any running node fails, the traversal returns to the parent. The parent selector then moves on to the next priority child



Ready
Visiting



Ready
Visiting
Failed
Running
Complete

https://gamedev.stackexchange.com/questions/51693/decision-tree-vs-behavior-tree

Image:
https://nailcube.wordpress.com/2011/07/26/interview-with-the-creators-of-dig-n-rig/

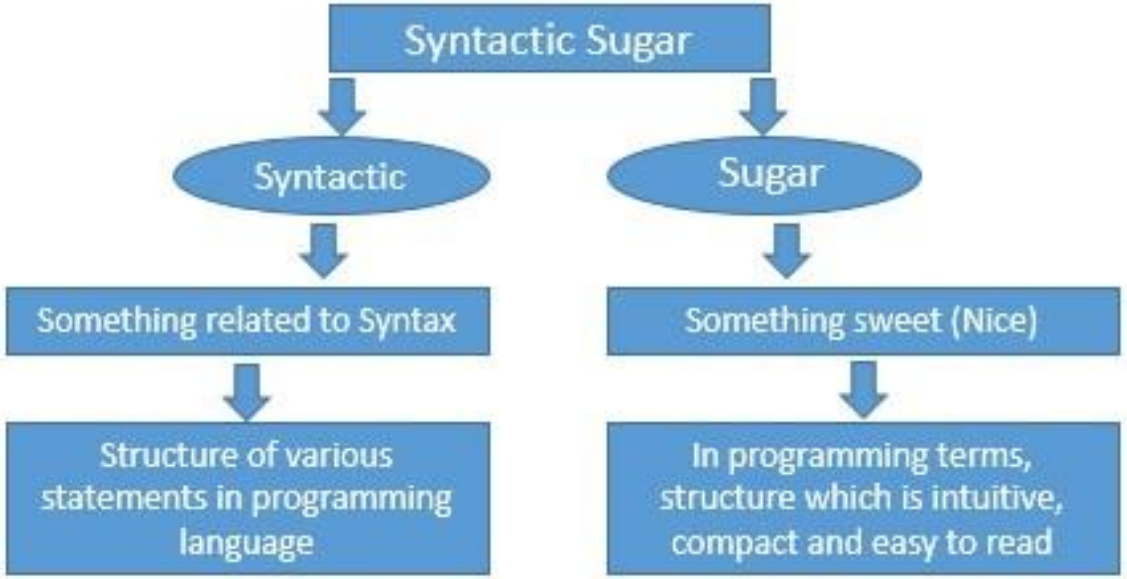
Syntactic Sugar

What is referred to by the term “Syntactic Sugar”?

- Syntactic Sugar is used in many programming languages
- Syntactic Sugar is a simplified & compact syntax structure. It is intuitive, expressive, very easy to understand

Wikipedia:

- A construct in a language is called "syntactic sugar" if it can be removed from the language without any effect on what the language can do: functionality and expressive power will remain the same.



Source: http://backtobazics.com/concept/what-do-you-mean-by-term-syntactic-sugar/

REACTIVE DECISION MAKING ALTERNATIVES

Reactive Decision Making

- Real-time decision making by performing one action every instant
- Examples
  - State-action table
  - Universal plan
  - Behavior trees
  - Rule systems

Reactive Planning

- Behavior trees implement a simple form of reactive planning
  - Real-time decision making by performing one action every instant

Reactive Planning

- Where a state-action table gives us:

  s_1 --> a_1

  s_2 --> a_2

  …

  we get this from reactive plans:

  s_1 --> a_{11} a_{12} a_{13}…

  s_2 --> a_{21} a_{22}…

  …

Reactive Planning

- Advantages
  - Try things, fail, and fall back
  - Appearance of goal-driven behavior without a formal definition of goals
  - Fast

Reactive Planning

- Advantages
  - Try things, fail, and fall back
  - Appearance of goal-driven behavior without a formal definition of goals
  - Fast
- Disadvantages
  - Can’t really think ahead
  - Only as forward-thinking as the designer makes it

Next Class

- More decision making!
  - Production / Rule Based systems
  - Fuzzy logic + probability
  - Planning