Autonomous Robotics

Finite State Machines and Behavior Trees

Josiah Hanna
University of Wisconsin — Madison

Slide Credit: Many slides lightly modified from Mike Hagenow's "Intro to Robot Systems" course.

Announcements

Good job completing the midterm!

Homework 4 is due after spring break.

Final project will be released soon.

<2 days to complete mid-course evaluation survey.

Learning Outcomes

After today's lecture, you will be able to:

- Implement high-level robot behaviors.
- Compare and contrast finite state-machines and behavior trees.

What we've done so far

Basic control: directly get from current state to set point.

Inverse kinematics: given a desired point in task space, determine a configuration that achieves it.

Motion planning: determine a sequence of movements that reach a desired configuration.

We likely want robots to be able to do more than just move from point A to point B.

Overall behavior is a composition of multiple simipler sub-behaviors (and sub-behaviors might be compositions of even simpler sub-behaviors)

Goal: decompose behavior into sub-behaviors

One approach: finite state machines.

Powerful/simple tool to facilitate switching between different behaviors

- **State:** associated with a specific robot controller (e.g., move here)
- **State transitions:** conditions for when to switch states

Origin: automata theory

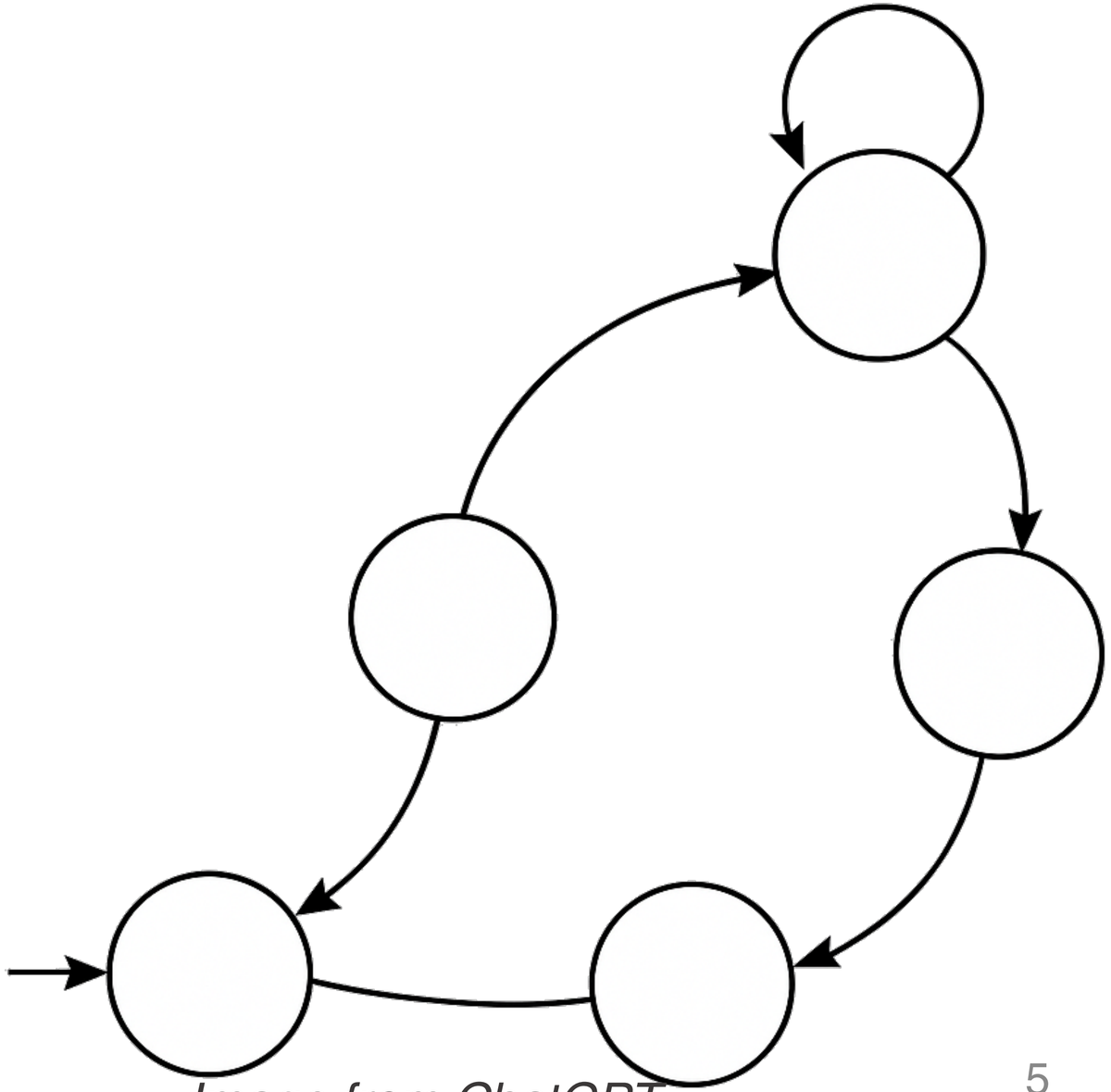


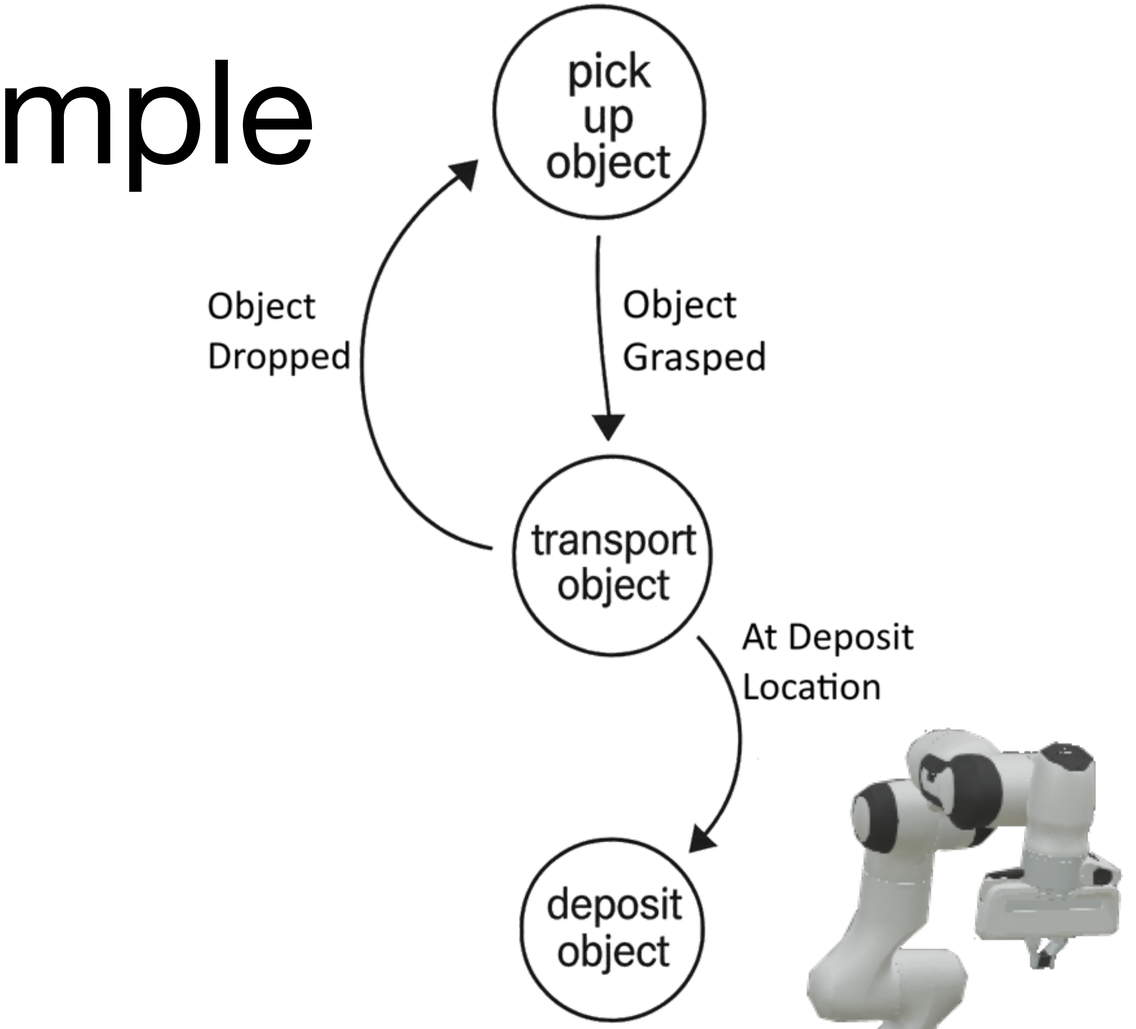
Image from ChatGPT

FSM Example

Pick and Place

- Many different possibilities
- One option: three controllers, one for each task phase
  pick, transport, place

Transitions help create a more **reactive** behavior



Finite State Machines

A finite state machine (FSM) is defined by tuple $(\Sigma, S, s_o, \delta, F)$.

- Σ is the input alphabet, a set of symbols representing events that trigger state transitions.
- S is a finite set of states.
- s_0 is the initial state.
- $\delta : S \times \Sigma \rightarrow S$ is the state-transition function that maps a state and an input symbol to a new state.
- $F \subseteq S$ is the set of final states.

FSM Example

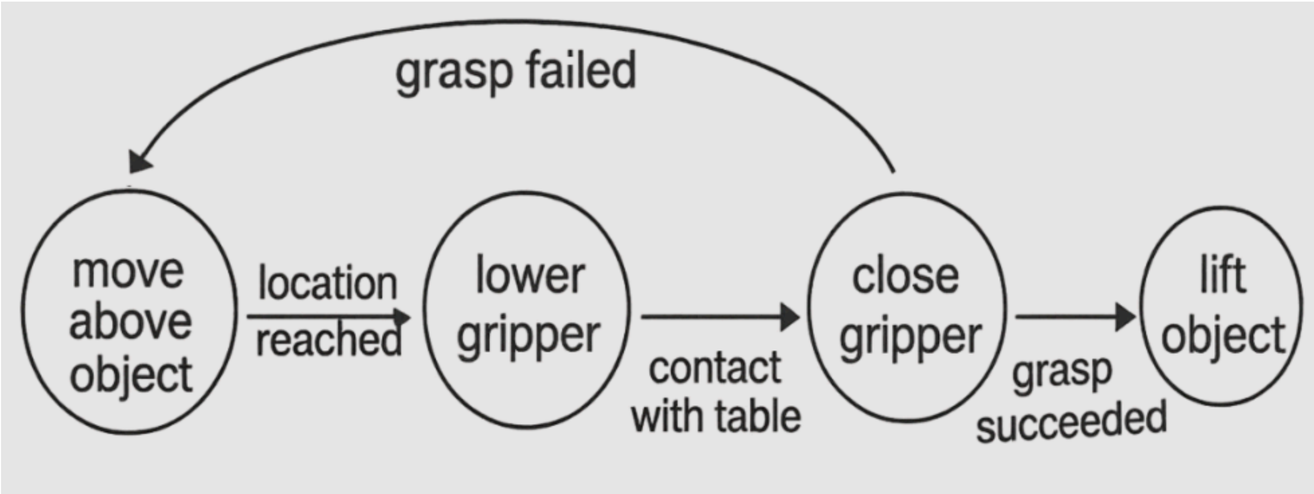
A finite state machine (FSM) is defined by tuple $(\Sigma, S, s_o, \delta, F)$.

- Σ = {Object Grasped, At Deposit Location, Object Dropped}
- S = {pick up object, transport object, deposit object}
- s_0 = pick up object
- $\delta = \begin{cases} \text{(Pick up object, Object grasped) —> transport object} \\ \text{(Transport object, At deposit location) —> deposit object} \\ \text{(Transport object, object dropped) —> pick up object} \end{cases}$
- F = {deposit object}.

Adding layers

Each behavior node might be too complex for a single controller to handle.

Solution: each behavior node can contain its own FSM over simpler behaviors.



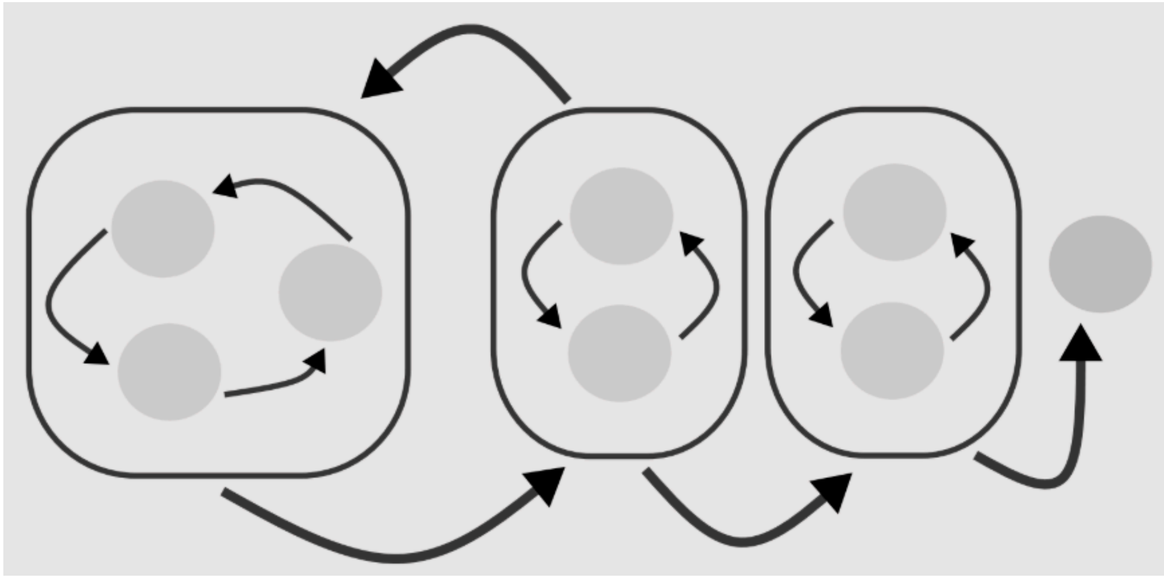
Hierarchical FSMs

Each state is implemented with its own internal FSM.

Higher-level FSM monitors for its own transitions.

Can add an arbitrary number of layers.

Problem: FSMs can get messy (to debug and maintain) with a large number of states.

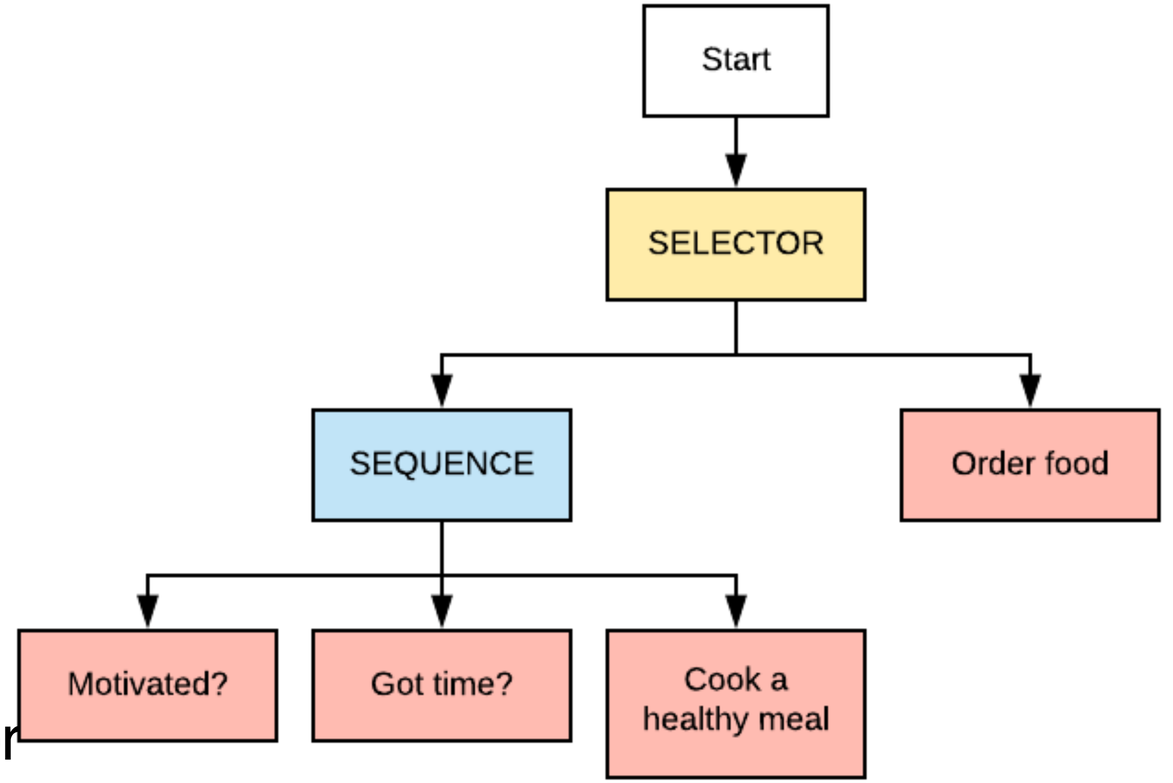


Behavior Trees

Tree of nodes defining the robot **system behavior**

Types of nodes:

- **Composite**: one or more children
- **Decorator**: one child and performs transformations of the child's output
- **Action**: zero children, execution of behavior



Fun fact: original use case was modeling characters in the video game industry

Image from [Awaiting Bits](#) Blog

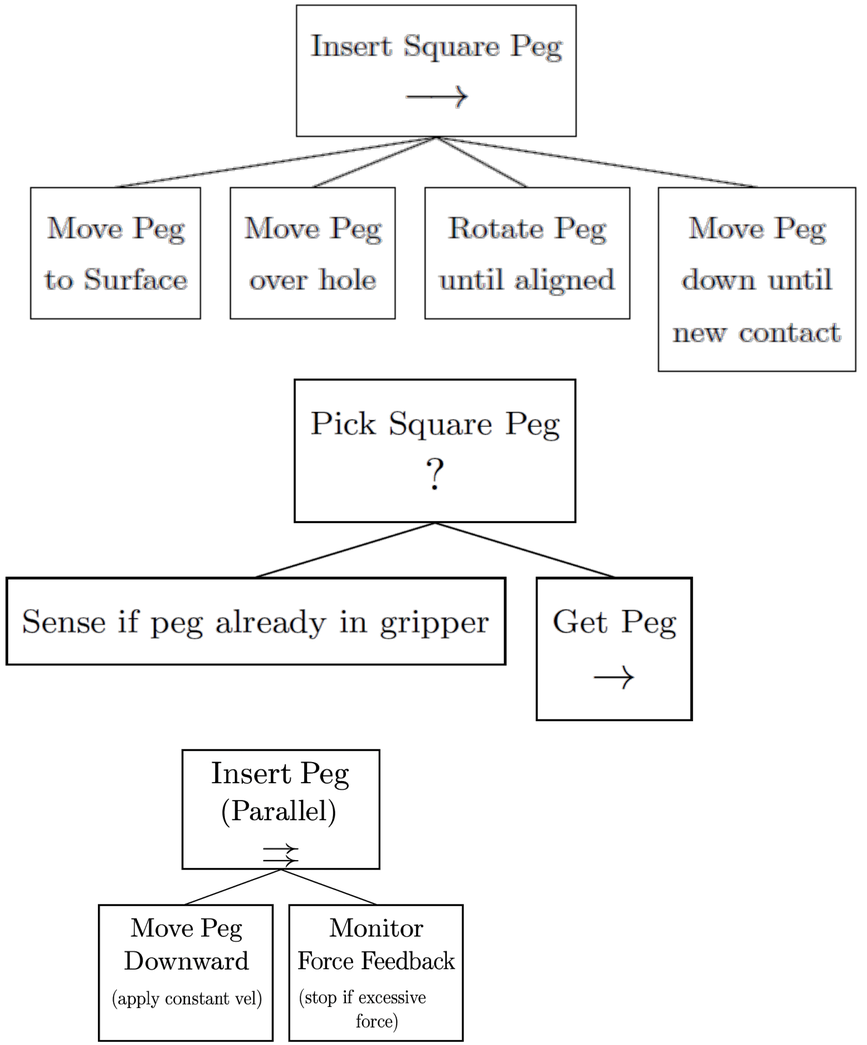
Behavior Trees

Composite Nodes

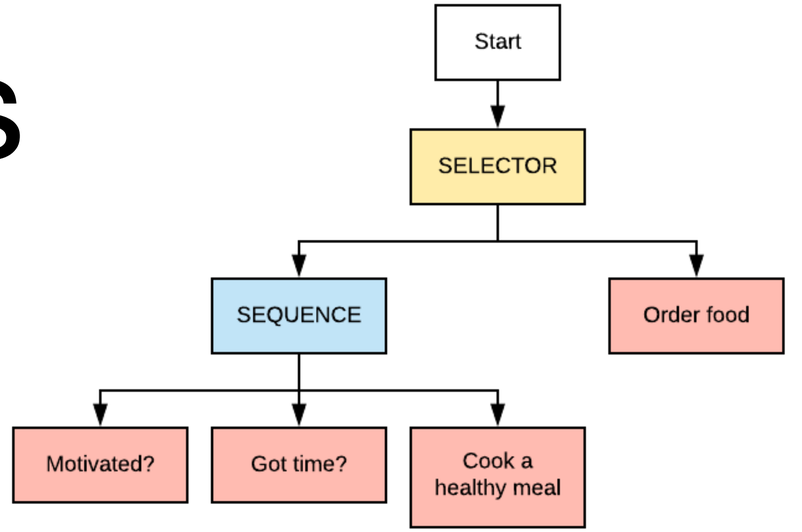
- one or more children
- regulate the control flow

Common examples:

- **Sequence node:** execute children in order, success if all succeed, fail if any fails (AND operator)
- **Selector**: execute children in order, success if any succeeds, failure if any fails (OR operator)
- **Parallel**: executes all N children at the same time, success if M or more succeed, fail if (N-M) fail (for a choice of M≤N)



Behavior Trees



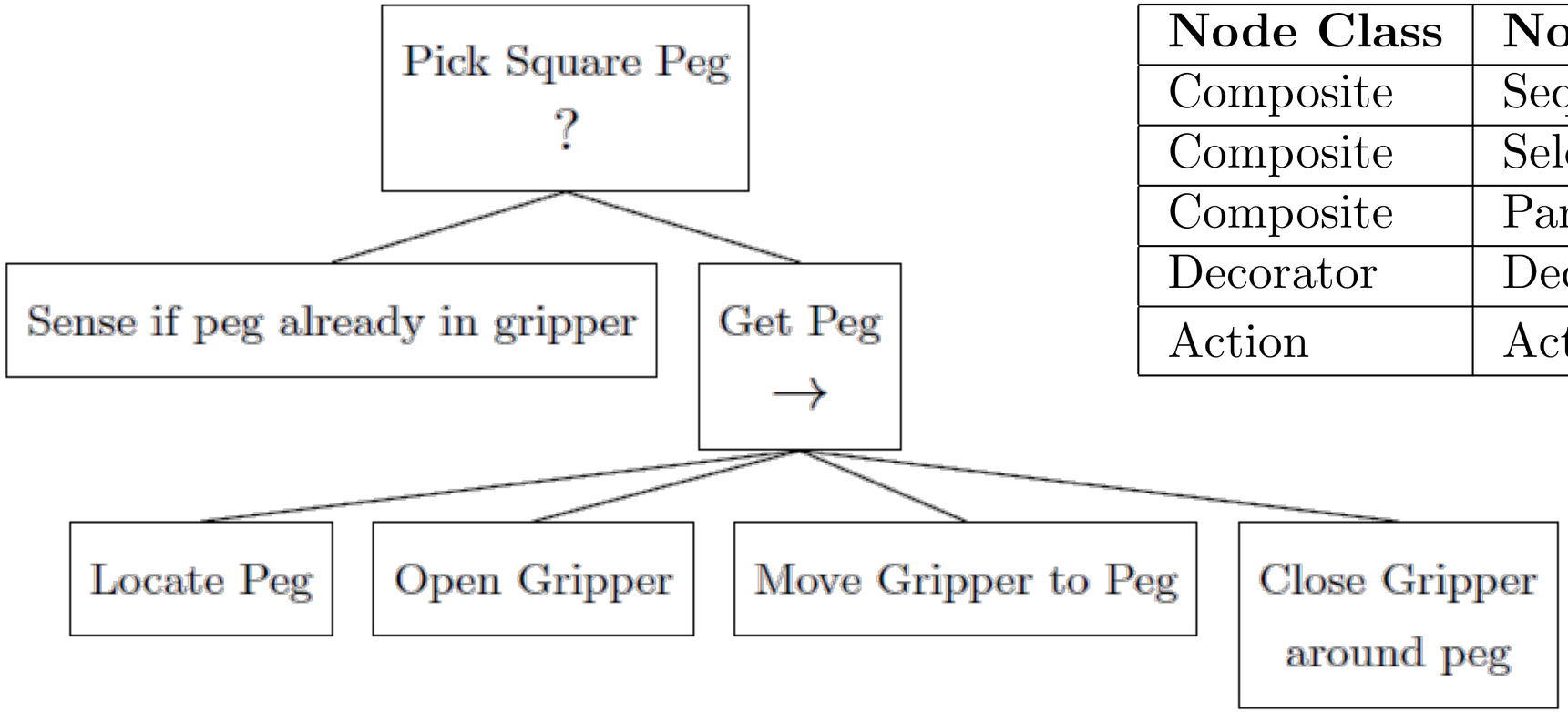
Decorator

- one child and performs transformations of the child's output
- Examples
  - **Inverter:** invert status of child, Success -> Failure, Failure -> Success (NOT operator)
  - **Override success:** return success no matter what (non-critical operations)
  - **Repeating:** repeat child execution until success, until failure, or endlessly

Action

- very flexible, any amount of complexity/behavior that you want to program
- can have input/outputs

Behavior Tree Example

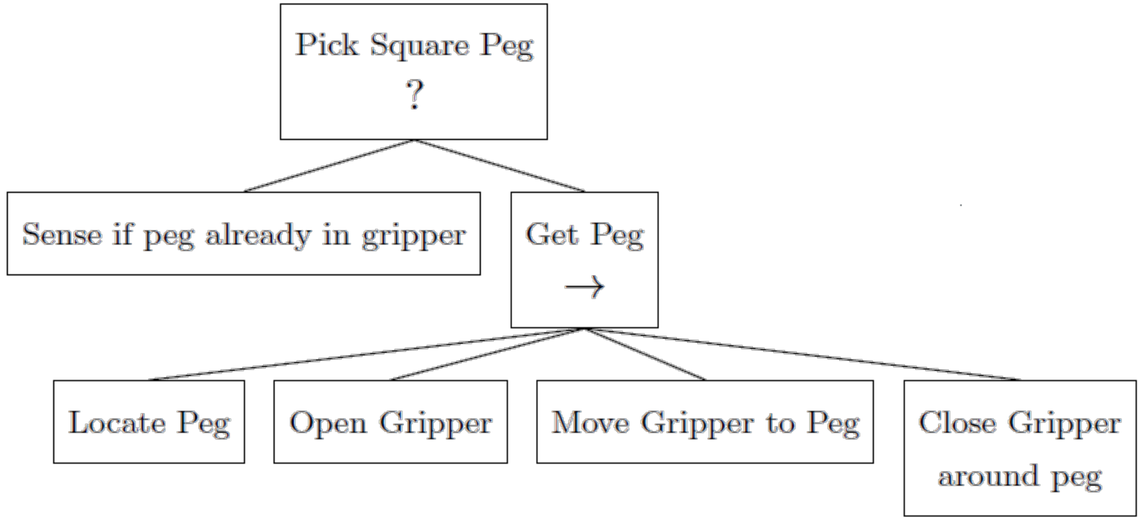


Node Class	Node Type	Symbol
Composite	Sequence	→
Composite	Selector/Fallback	?
Composite	Parallel	⇒
Decorator	Decorator	◊
Action	Action	Text

Behavior Trees Summary

Can use a **behavior tree** to **script** *sophisticated robot behavior*

- Sequences
- Parallel activity
- Looping (through decorator)



Node Class	Node Type	Symbol
Composite	Sequence	→
Composite	Selector/Fallback	?
Composite	Parallel	⇒
Decorator	Decorator	◊
Action	Action	Text

Behavior trees *rely on* and *ultimately execute action code* that defines the physical robot operations

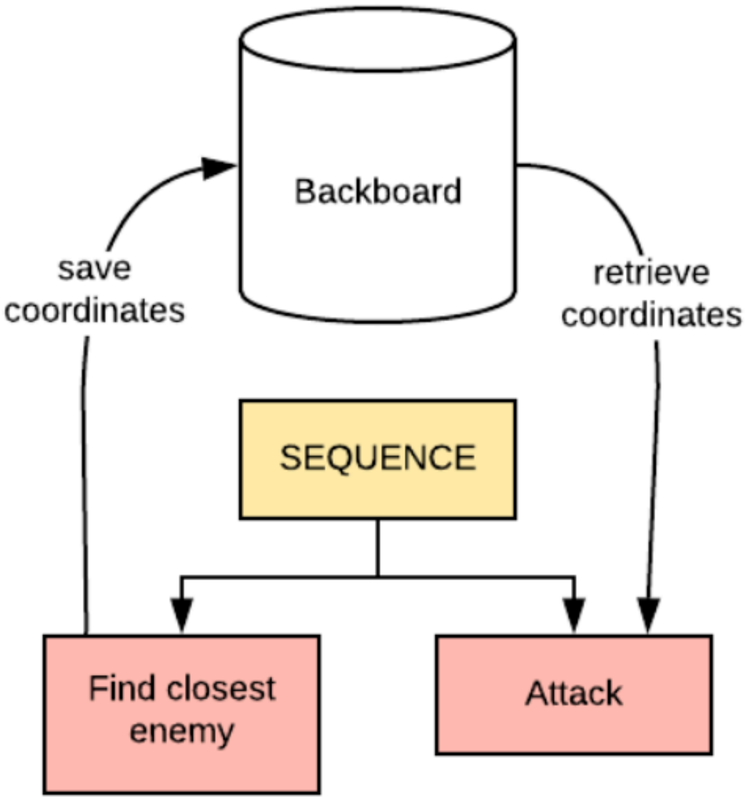
Communication Between Nodes

Often need some way for nodes to access same memory.

Common solution: blackboard architecture.

Nodes can read/write to a shared memory.

- Can also be used by perception / localization / motion control.



Credit: Awaiting Bits

Summary

- Discussed how robot behaviors may need to be decomposed into simpler sub-behaviors.

- Introduced finite state machines and behavior trees as two methods to implement the decomposition.

Action Items

- Have a good spring break!
- Reading on reinforcement learning due after spring break.