BEHAVIOR TREES

ARTIFICIAL INTELLIGENCE | COMP 131

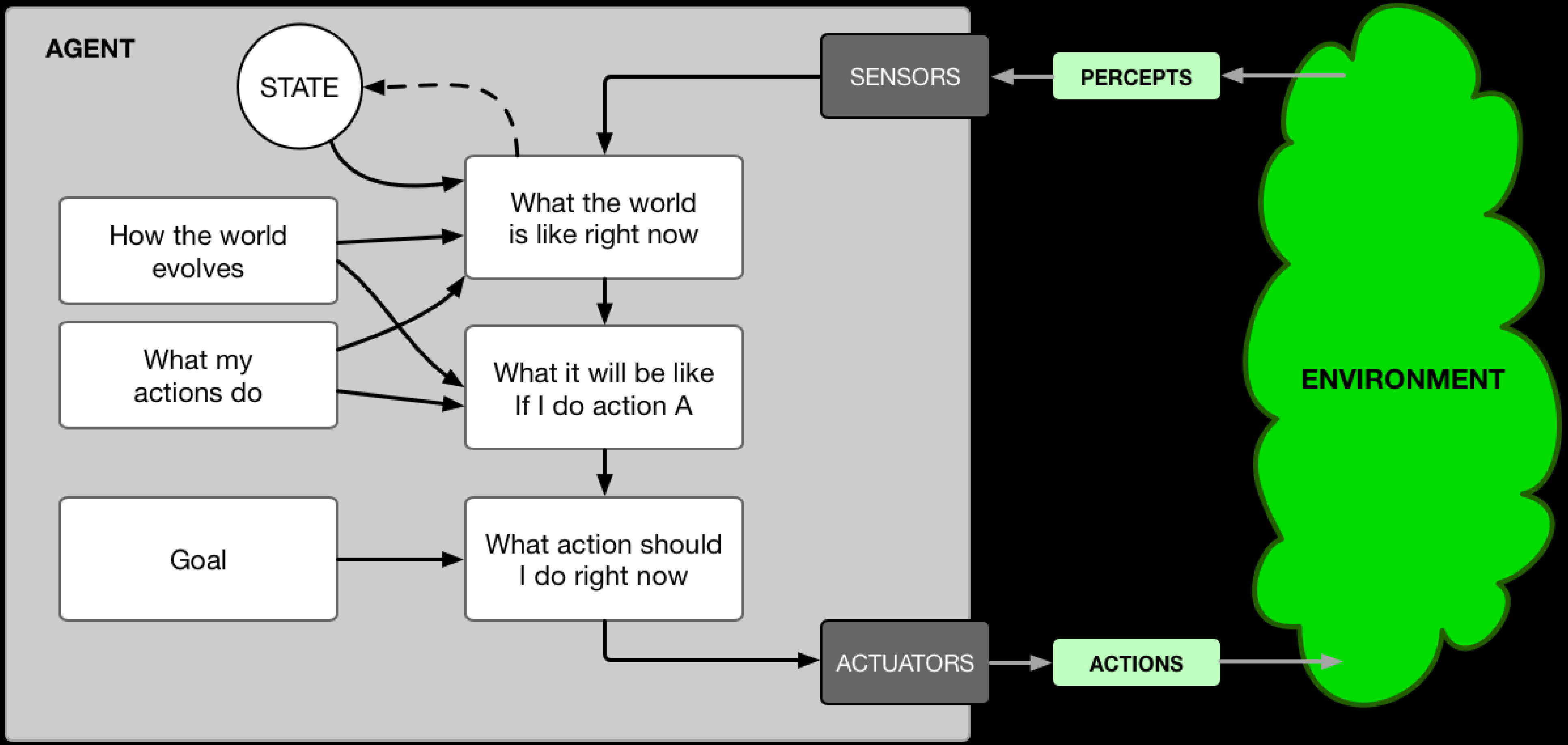
TODAY ON AI

- Behavior Trees
- Behavior Tree nodes
- Behavior Tree features
- Design patterns
- Readings
- Questions?

Round 1
1v1

SECTION 01

Behavior Trees



UNPLANNING AGENT: REFLEX AGENTS, MODEL-BASED REFLEX AGENTS, LEARNING AGENTS
PLANNING AGENT: MODEL-BASED, GOAL-BASED AGENTS

Wikipedia defines a Behavior Tree as: **Behavior trees are a formal, graphical modeling language used in Systems and Software Engineering**. Behavior trees employ a well-defined notation to unambiguously represent natural language requirements for large-scale software-integrated systems.

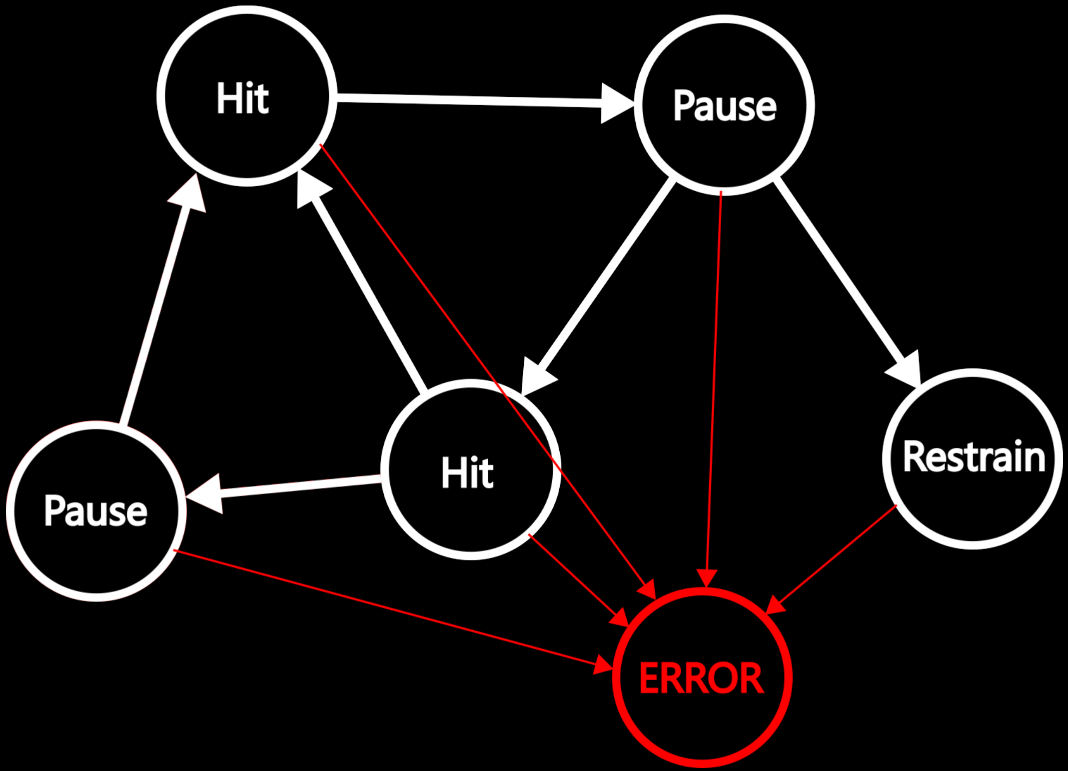
- Developed by **R. G. Dromey** with some key concepts published in **2001**
- Used to describe large-scale systems, embedded systems, role-based access control, biological systems, etc.

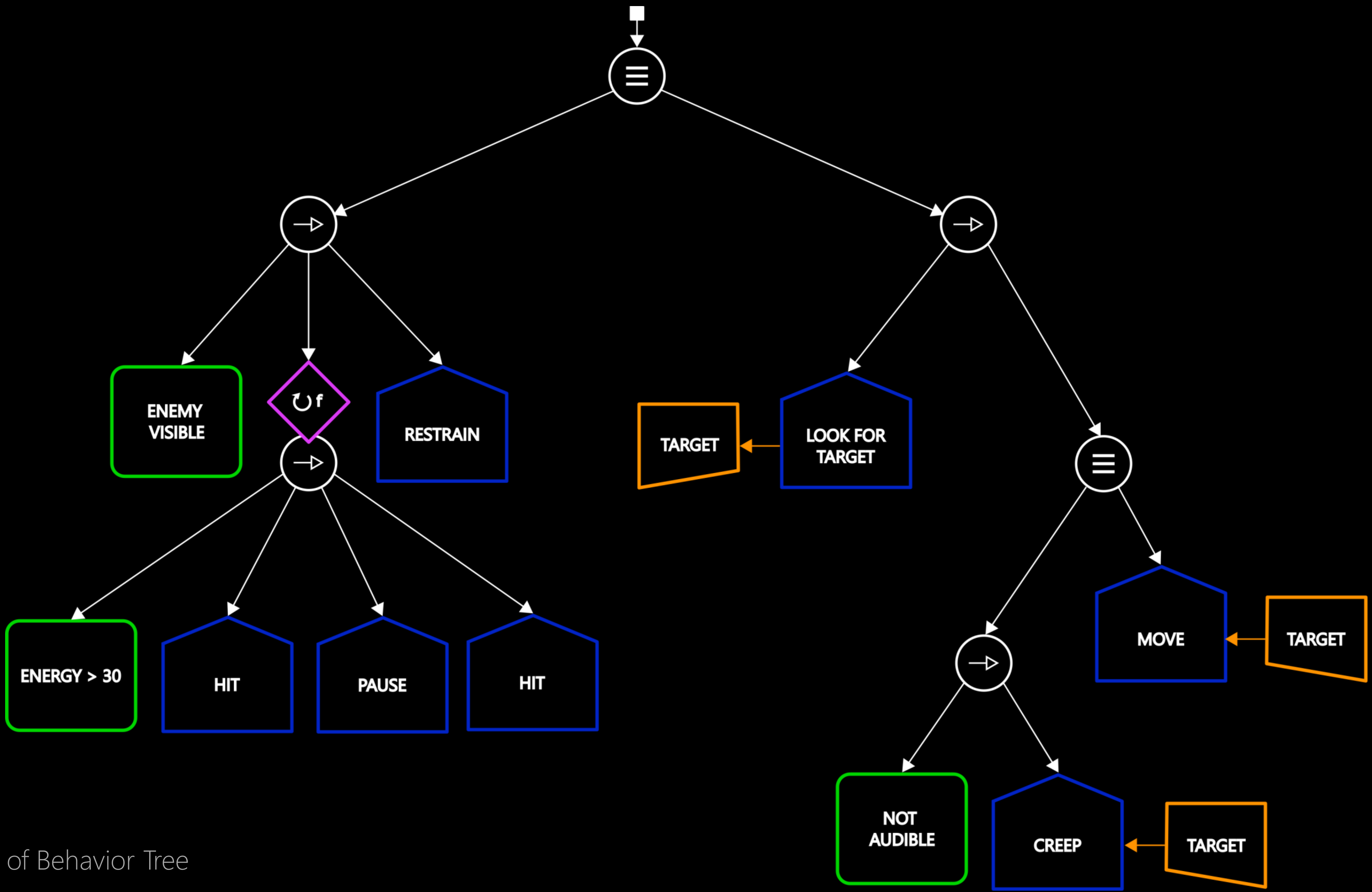
In 2004 and 2005, **Halo 2** and **Façade** AI designers adopted a similar graphical representation and name for a different formalism

They synthesized several techniques and algorithms in one manageable tool:

- Finite State Machines
- Hierarchical Finite State Machines
- Scheduling
- Search
- Resource Conflict Resolution

- While finite state machines are **reasonably intuitive** for simple cases, as they become more **complex**, they are hard to keep goal-oriented
- As the number of states increases, the transitions between states become exponentially complex
- Hierarchical state machines can help, but many of the same issues remain





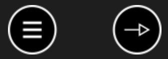
10

Example of Behavior Tree

SECTION 02

Behavior Tree nodes

Behavior trees organize their **nodes** into a **tree** or, more generally, **directed acyclic graph (DAG)**:

NODE	REPRESENTATION	RESULT
A **task** alters the state of the system		SUCCEEDED, FAILED, or RUNNING
A **condition** tests some property of the system		SUCCEEDED, or FAILED
A **composite** aggregates tasks and conditions		SUCCEEDED, FAILED, or RUNNING
A **decorator** alters the basic behavior of the tree-node it is associated with		SUCCEEDED, FAILED, or RUNNING
Sub-trees as a reference to complex behaviors		SUCCEEDED, FAILED, or RUNNING
Operations on the blackboard (read or write)		n/a

The most common **composites** are:

- **Sequence**: Children are evaluated in order (left to right). It fails as soon as one of the children fails, otherwise it succeeds
- **Selection**: Children are evaluated in order (left to right). It fails if all children have failed, otherwise it succeeds
- **Priority**: Like selection, but the children are evaluated in order of priority
- **Random sequence**: Like sequence, but the children are evaluated in random order
- **Random selection**: Like selection but the children are evaluated in random order

The most common decorators are:

- **Logical negation**: It executes the attached node and then it negates its result

- **Until Succeeds**: It executes the attached node while it fails returning **RUNNING**. It returns **SUCCEEDED** at the first success.

- **Until Fails**: It executes the attached node while it succeeds returning **RUNNING**. It returns **SUCCEEDED** at the first failure.

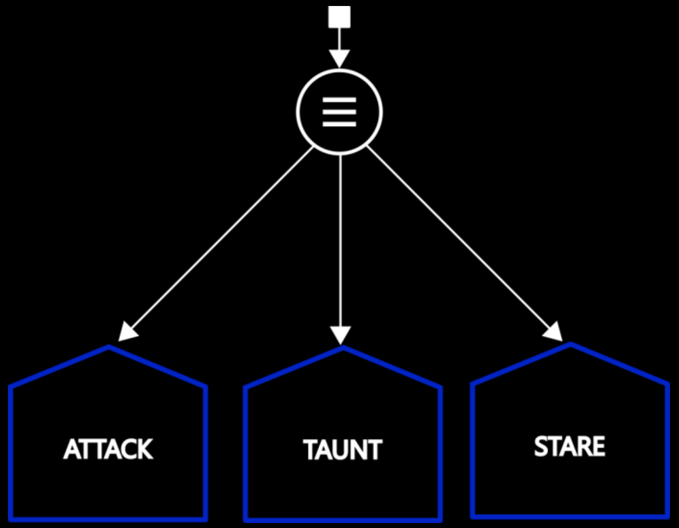
- **Resource semaphore**: It resolves conflicts between nodes associated with the same resource. It returns **RUNNING** while the resource is not available.

- **Timer**: It executes the attached node for a specific amount of time. It returns **RUNNING** while the timer is running. It returns **SUCCEEDED** after the expiration.

Real Behavior Tree implementations normally have an inter-node communication mechanism called **blackboards**.

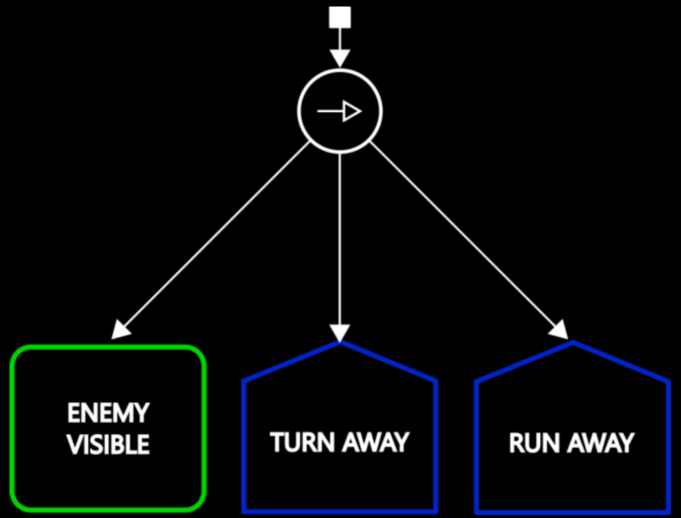
- The **blackboard** implementation is one big design choice:
  - **One** blackboard for the whole tree
  - **Sub-tree** private blackboards
  - All the above

- The simplest implementation of a blackboard is a **hash-table** or **dictionary**:
  - The **key** is the variable name
  - The **value** is the variable value



Try all the children **until one** succeeds.

```
class Selector Node
  function run
    for child in children
      if child.run
        return TRUE
    return FALSE
```

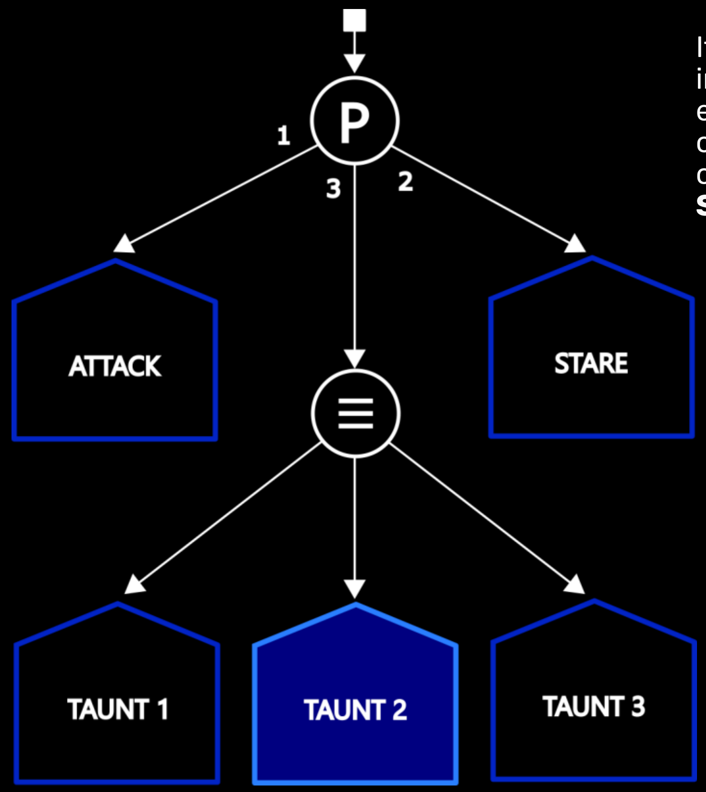


Execute all the children sequentially,
succeeding if all succeed.

```
class Sequence Node
  function run
    for child in children
      if not child.run
        return FALSE
    return TRUE
```


Typical sequence behavior

SEQUENCE NODES

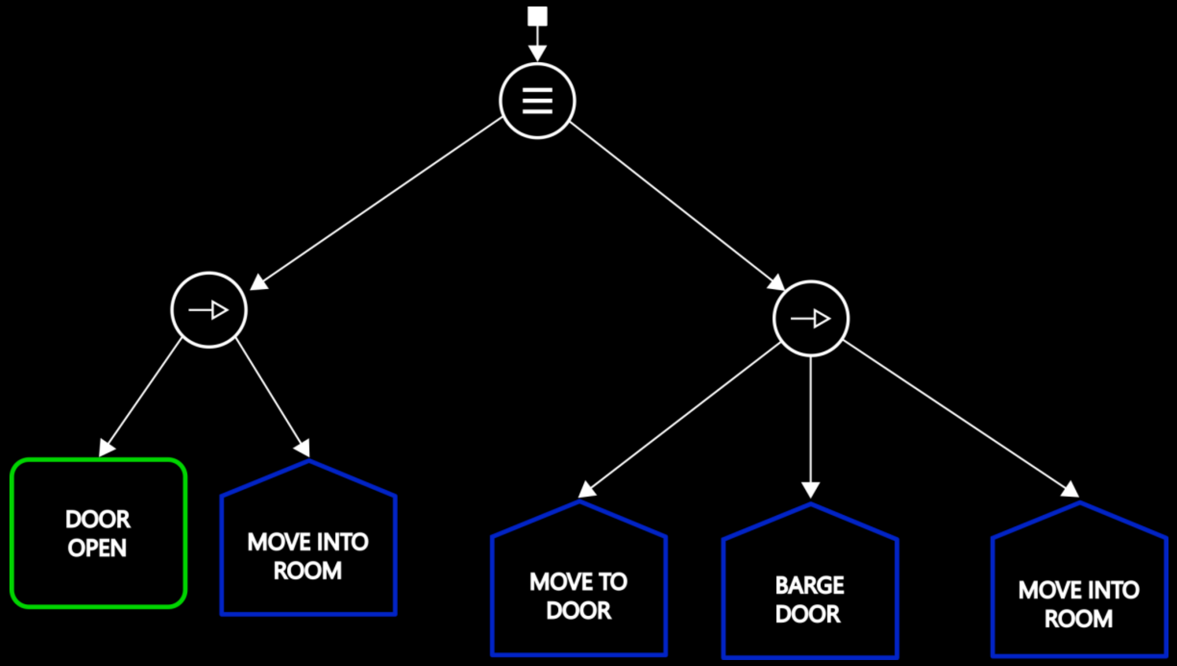


If **TAUNT 2** is marked as in a running state, its execution will continue only after the execution of both **ATTACK** and **STARE**.

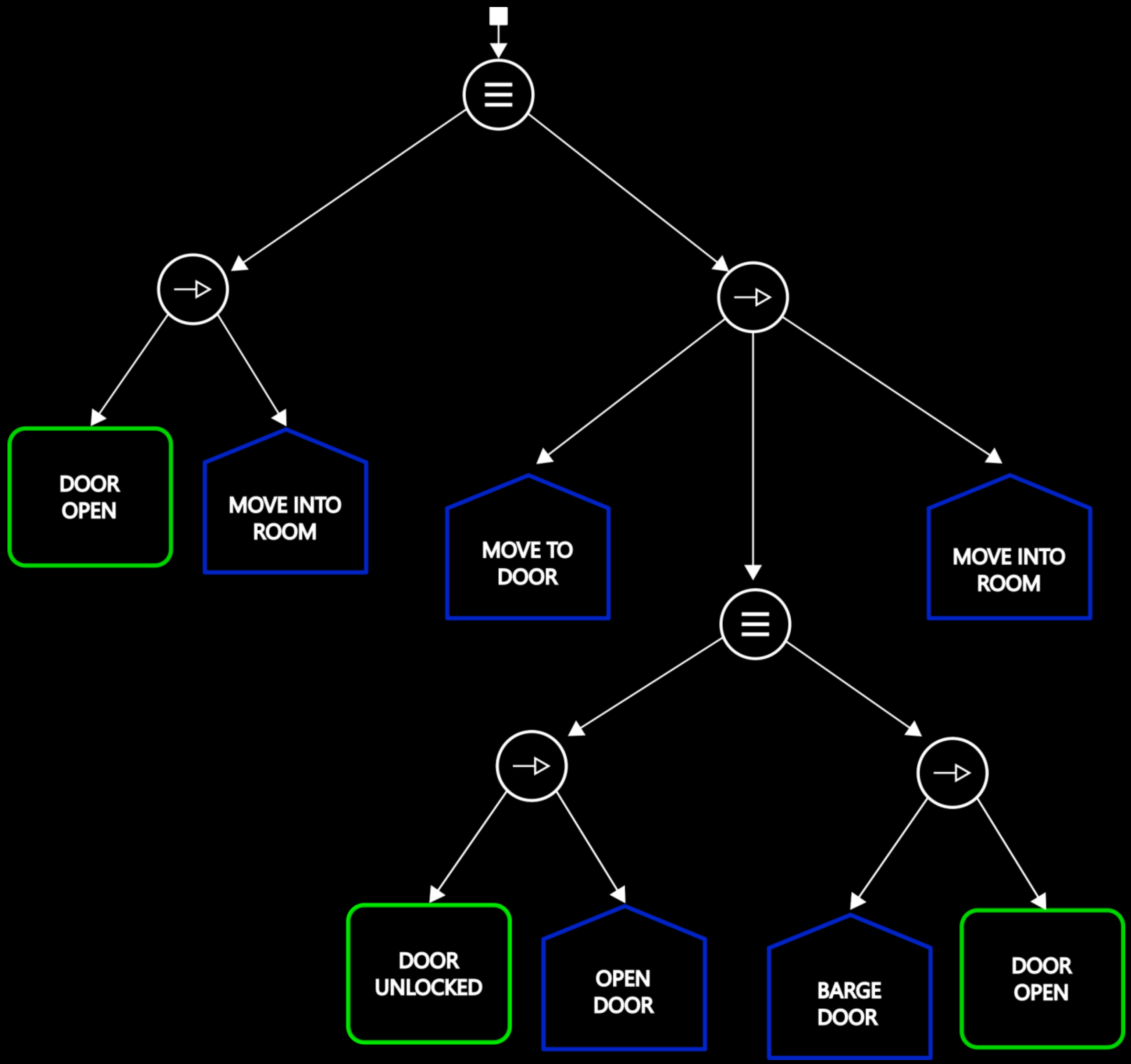
Try all the children **until one** succeeds in order of priority.

A priority node **forces** the tree to evaluate higher-priority nodes before continuing the evaluation of the one marked as **running**.

```
class Priority Node
  function run
  for child in sort(children)
      if child.run
         return TRUE
    return FALSE
```

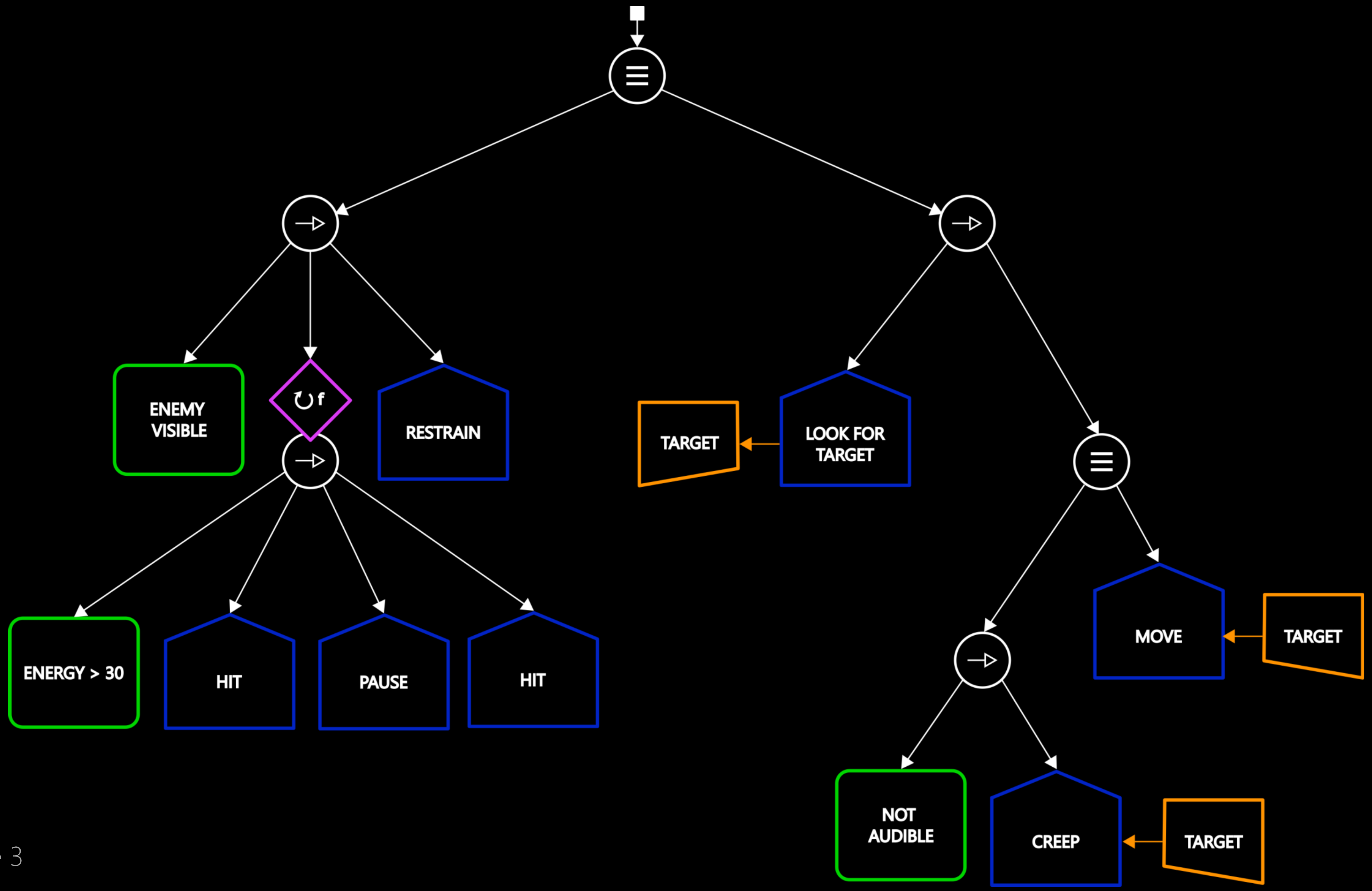

DOOR OPEN
MOVE INTO ROOM
MOVE TO DOOR
BARGE DOOR
MOVE INTO ROOM

Example 1


DOOR OPEN
MOVE INTO ROOM
MOVE TO DOOR
MOVE INTO ROOM
DOOR UNLOCKED
OPEN DOOR
BARGE DOOR
DOOR OPEN

19

Example 2


ENEMY VISIBLE
f
RESTRAIN
TARGET
LOOK FOR TARGET
ENERGY > 30
HIT
PAUSE
HIT
MOVE
TARGET
NOT AUDIBLE
CREEP
TARGET

Example 3

SECTION 03

Behavior Tree features

Behavior trees have several highly desirable **features**:

- The basic components are reusable
- AI can be goal directed and autonomous
- AI can respond to events
- The knowledge base is easy to read and debug
- The knowledge base is easy to maintain

Behavior trees also have several **improvements** over hierarchical finite state machines:

- The history of the state transitions is clear
- Easy to build sequences
- Easy to add new behaviors without rewiring

- It is hard to **produce** a minimal set of tasks, conditions, and decorators
- Less important criticism:
  - Slow to react to changes in strategy
  - It is hard do verify that the set of tasks, conditions, and decorators is powerful enough to describe the AI requirements

Another more valid criticism about behavior trees is that they do not implement a full search in the search space, rather a so-called **reactive search** or **planning**.

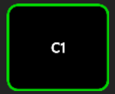
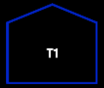
Planning defines a branch of Artificial Intelligence devoted to find a sequence of actions that will lead to a goal.

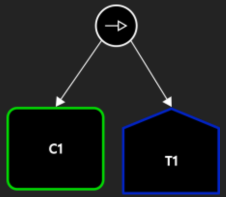
Reactive planning denotes a class of algorithms for **action selection** by autonomous agents that differs from **classical planning**:

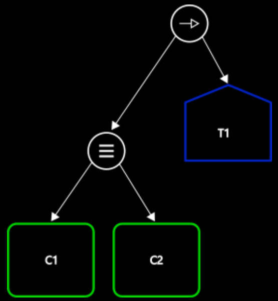
- They are **time-bound** so they can quickly deal dynamic and unpredictable environments
- They compute **just one (or few more) next action** in every instant, based on the current context

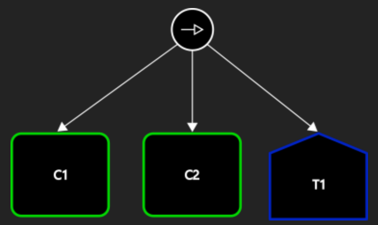
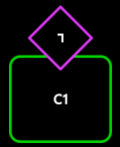
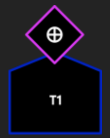
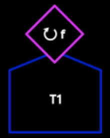
SECTION 04

Design patterns

TASKS	IF-THEN-ELSE	RULE NAME
	**return** C1	`Empty`
	**return** T1	`Always`

	**if** C1:
	    return T1
	else
	    return failed
		`Conditional execution`

	**if** C1 | C2:
	    return T1
	else
	    return failed
		`Conditional execution OR`

TASKS	IF-THEN-ELSE	RULE NAME
	**if** C1 & C2: **return** T1 **else** **return** failed	Conditional Execution AND
	**return not** C1	Negation
	**if not** Ti1.has_expired(): **if** T1 == succeeded **return** running **else** **return** failed **else** **return** succeeded	Timer
	**if** T1 == succeeded **return** running **else** **return** succeeded	Until fail

READINGS

Chapters 10 and 11

https://arxiv.org/abs/1709.00084

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

https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

QUESTIONS ?

Tufts
UNIVERSITY

ARTIFICIAL INTELLIGENCE

COMP 131

FABRIZIO SANTINI

VERSION 4.2