KTH ROYAL INSTITUTE OF TECHNOLOGY

Lecture 8: Behavior Trees and Task Switching

by Petter Ögren

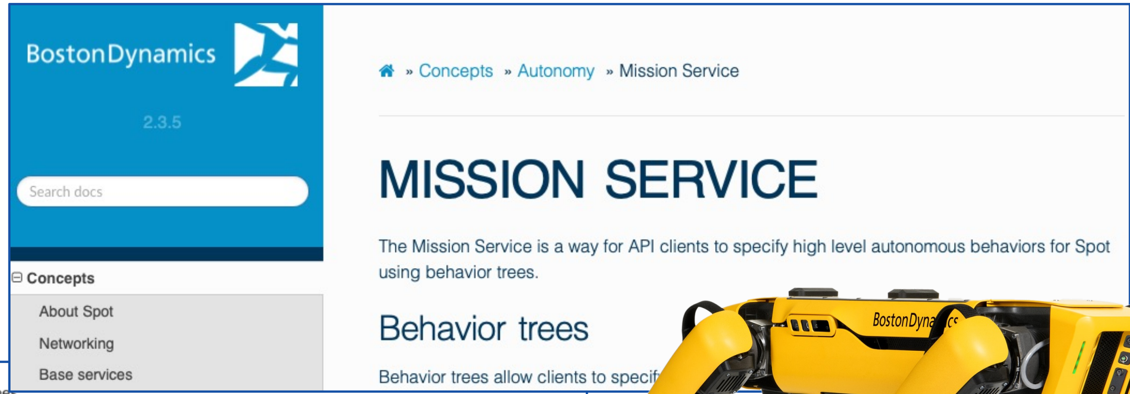
Content

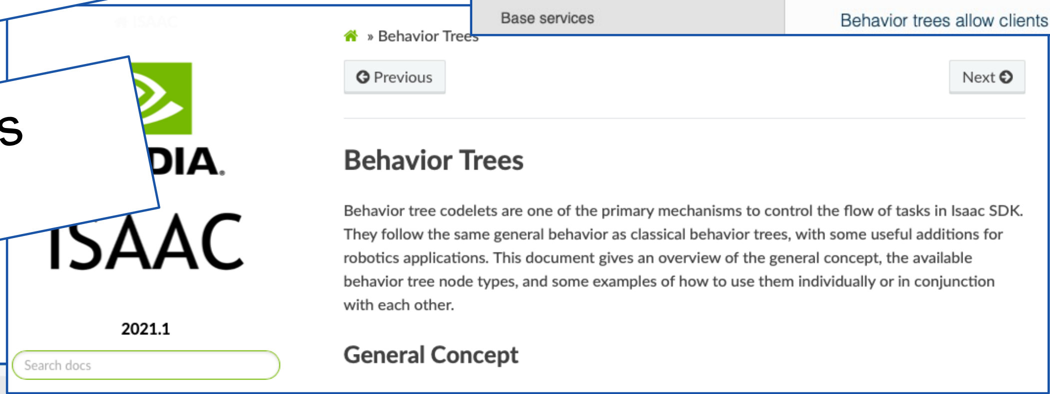
- When to use Behavior Trees (BTs)?
  - When deciding “what to do next”
  - Creating complex controllers/policies
- What are BTs?
  - Hierarchically modular policies
  - Optimally modular [1]
- How to create BTs?
  - Improvise
  - Use planning (backward chaining)
- The Big Picture
  - Genetic Algorithms
  - Control Theory (Performance Guarantees)
  - Reinforcement Learning

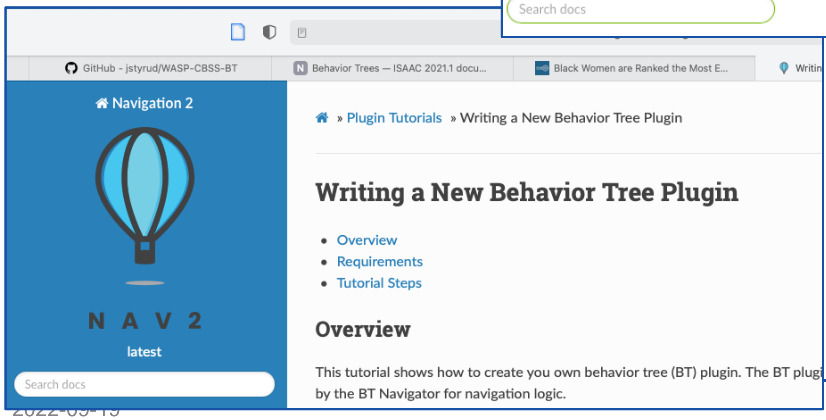
Behavior trees in use

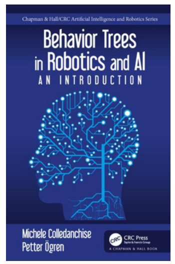
Invented by computer game programmers…

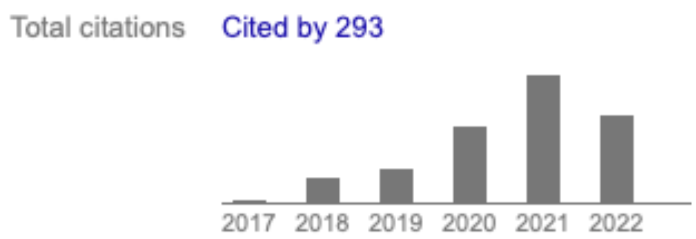
…refined by robotics researchers









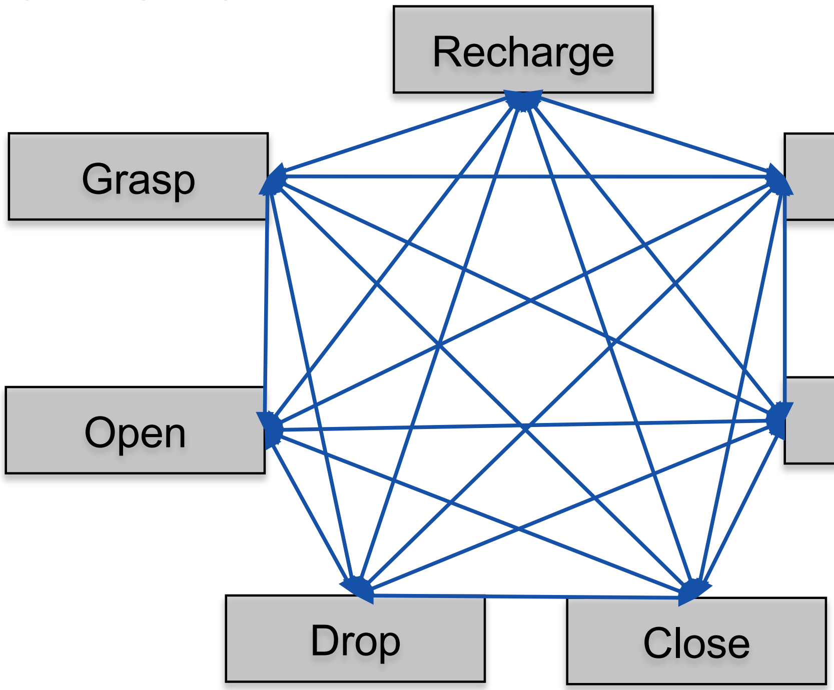


What to do next?

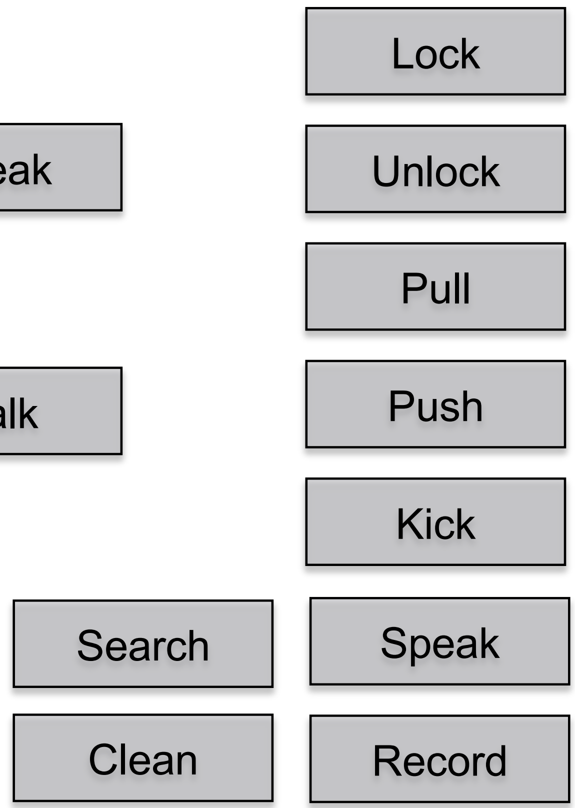
(any autonomous systems needs to answer this question)

Grasp	Recharge
Drop	Lock
Walk	Unlock
Open	Pull
Close	Push
Search	Kick
Clean	Speak
Listen	Idle
Run	Throw

What to do next?

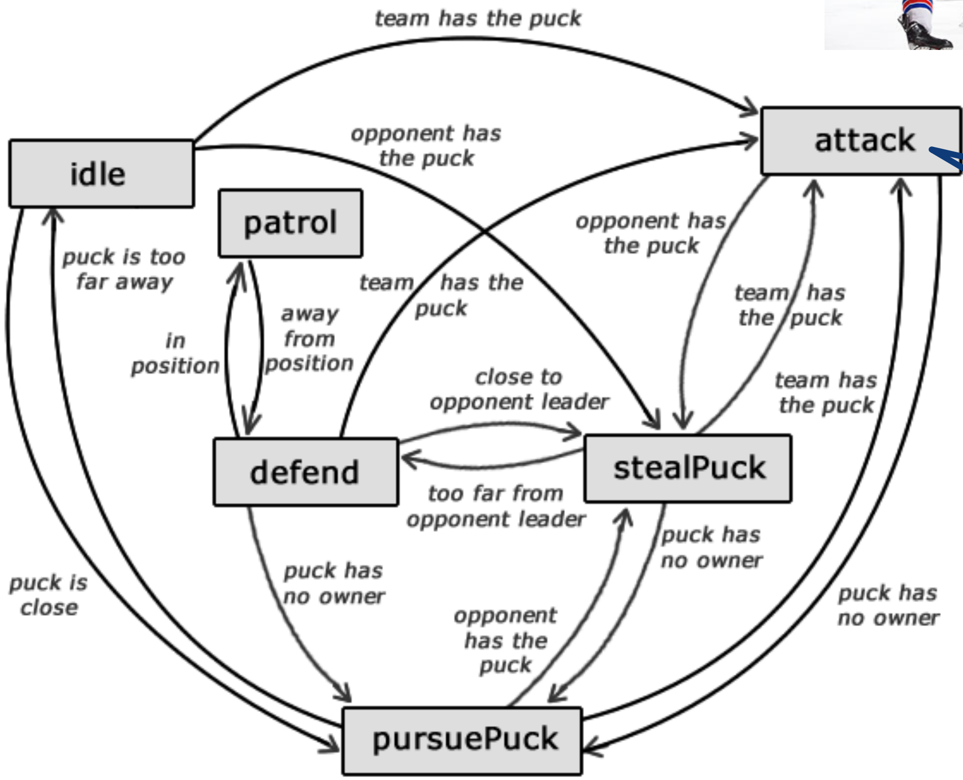


Finite State Machine



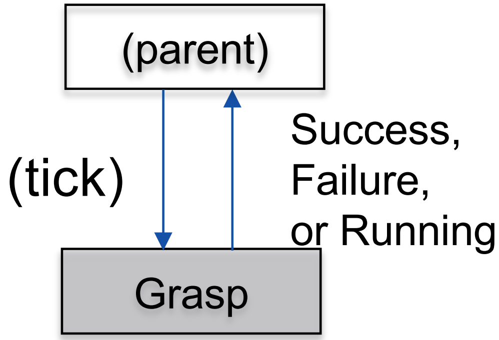
Each action needs to know “What to do next”…

Can you spot the Bug?



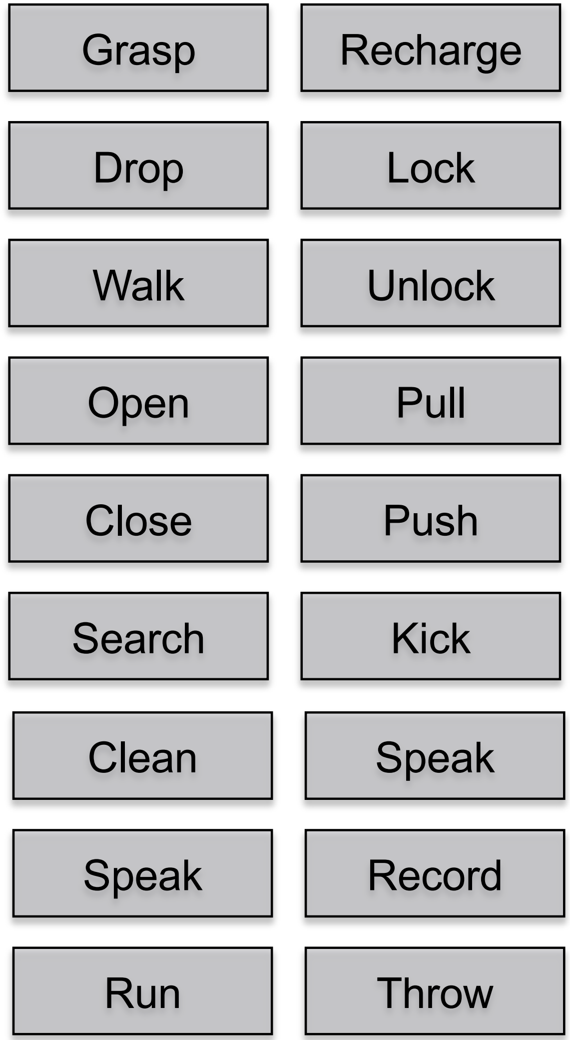
Can be expanded …

What to do next?



Behavior Tree

Each action needs to know
“Did I Succeed or Fail?”



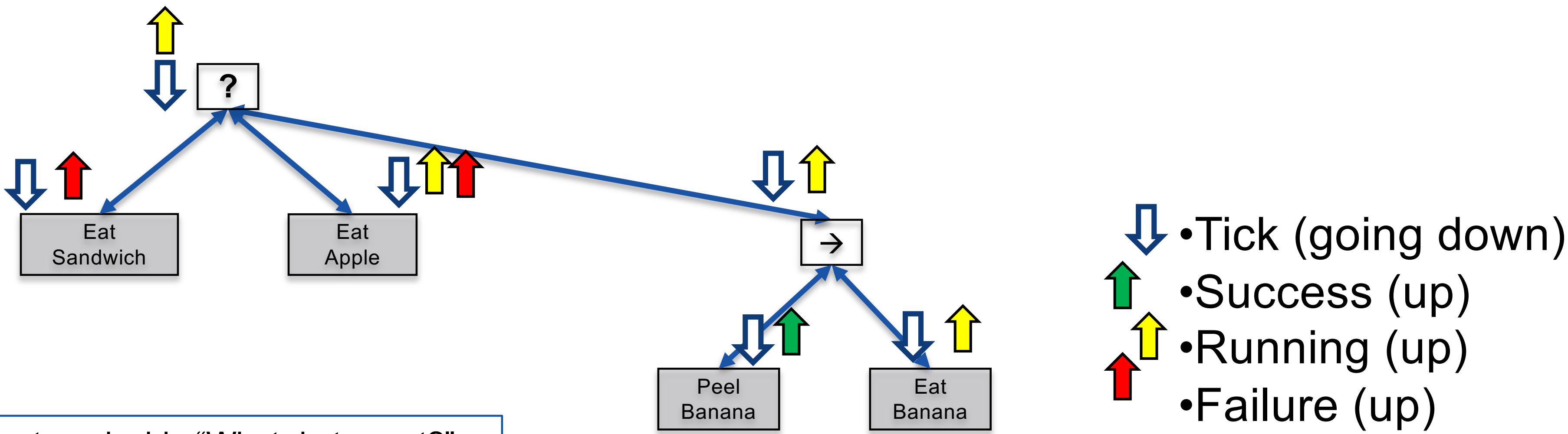
Ancestors decide “What do to next?”

Two Fundamental Compositions of Actions

- **Fallback** (?)(or)
- **Sequence** (→)(and)

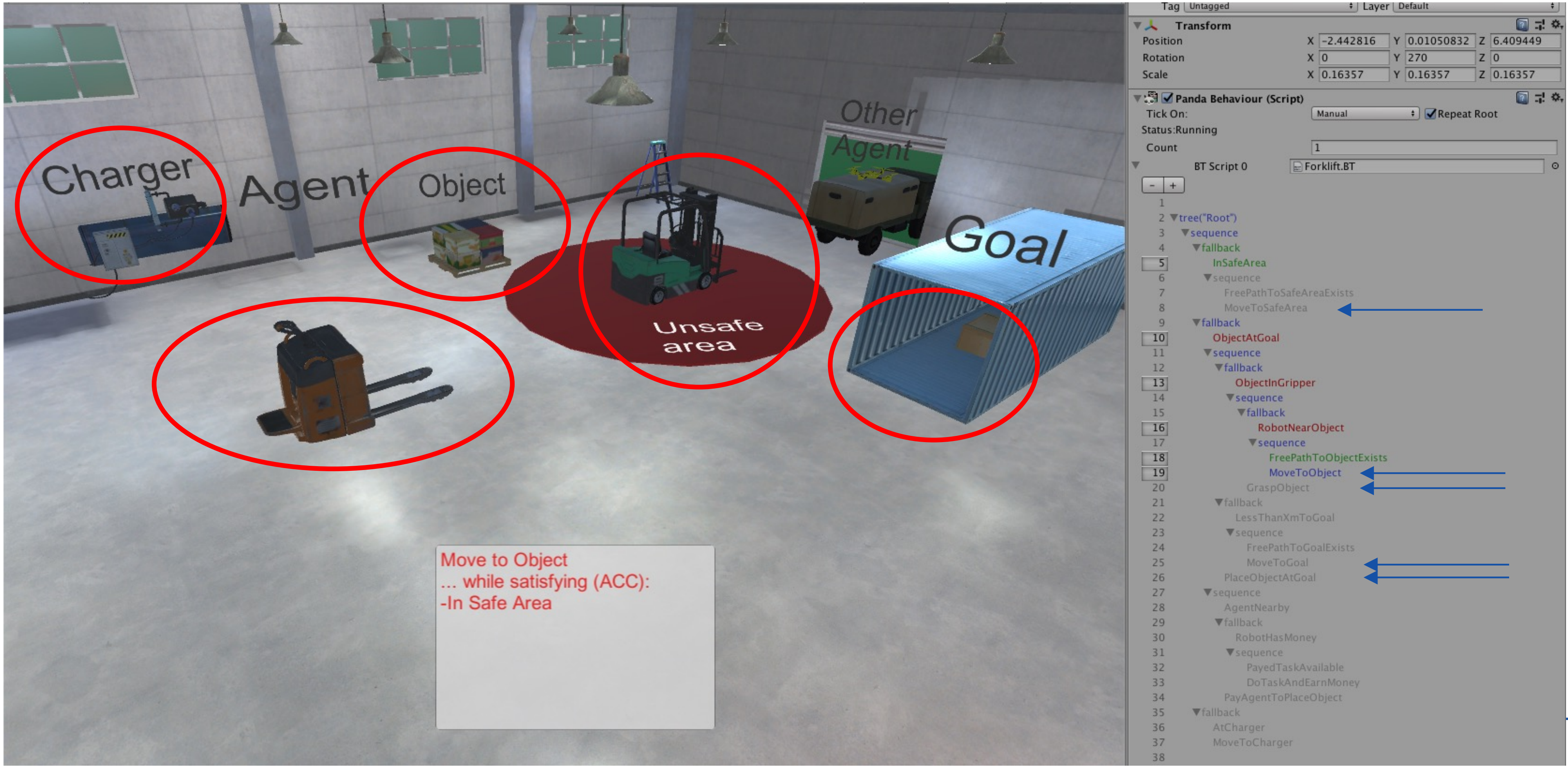
IF Failure then Tick Next
else Return "same as child"

IF Success then Tick Next
else Return "same as child"



Note how Ancestors decide "What do to next?"

Example



Charger
Agent
Object
Other Agent
Goal
Unsafe area
Move to Object
... while satisfying (ACC):
-In Safe Area
Tag Untagged
Layer Default
Transform
Position X -2.442816 Y 0.01050832 Z 6.409449
Rotation X 0 Y 270 Z 0
Scale X 0.16357 Y 0.16357 Z 0.16357
Panda Behaviour (Script)
Tick On: Manual
Repeat Root
Status:Running
Count 1
BT Script 0 Forklift.BT
tree("Root")
sequence
fallback
InSafeArea
sequence
FreePathToSafeAreaExists
MoveToSafeArea
fallback
ObjectAtGoal
sequence
fallback
ObjectInGripper
sequence
fallback
RobotNearObject
sequence
FreePathToObjectExists
MoveToObject
GraspObject
fallback
LessThanXmToGoal
sequence
FreePathToGoalExists
MoveToGoal
PlaceObjectAtGoal
sequence
AgentNearby
fallback
RobotHasMoney
sequence
PayedTaskAvailable
DoTaskAndEarnMoney
PayAgentToPlaceObject
fallback
AtCharger
MoveToCharger

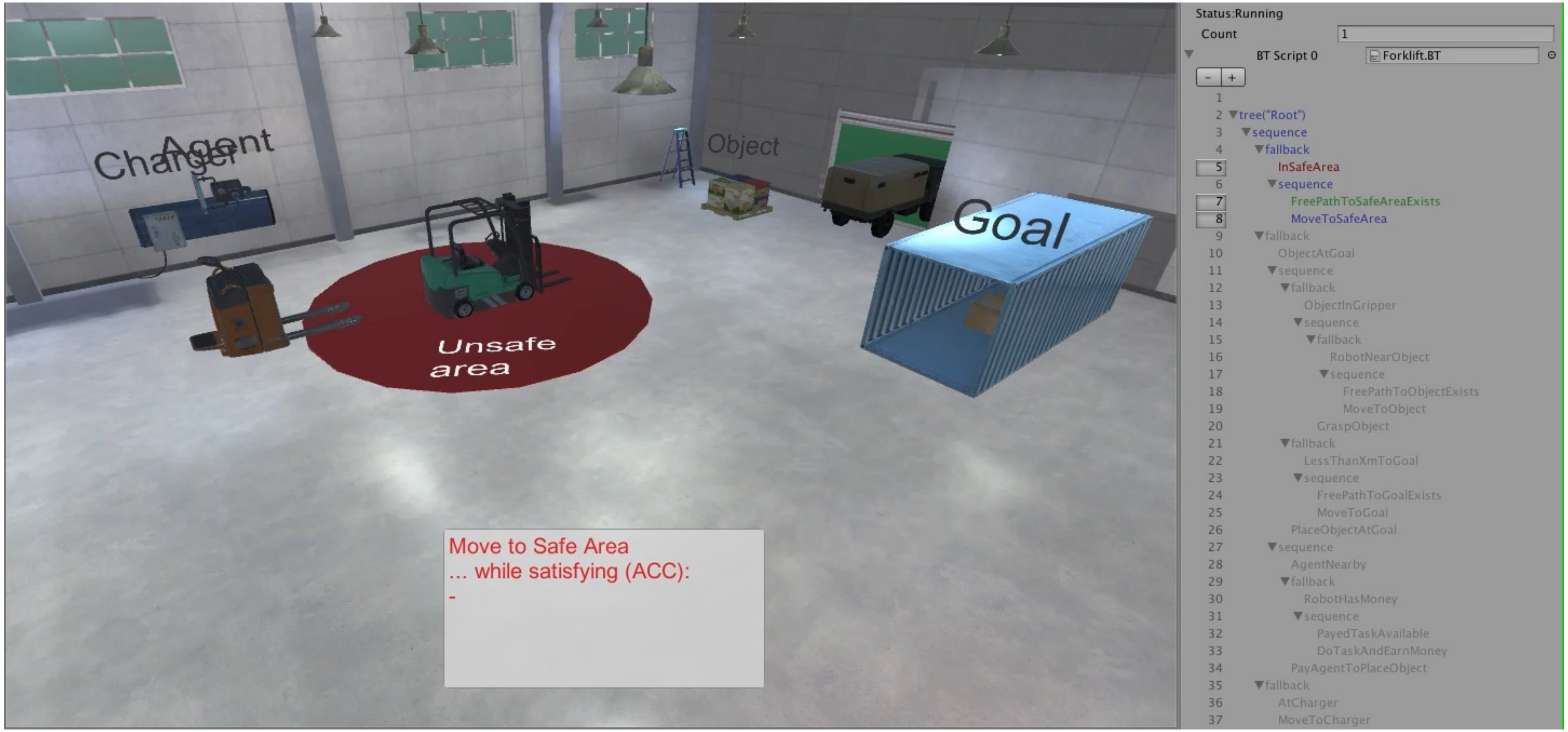
Execution without disturbances

•Success
•Running
•Failure

- Classical Control handles noise disturbances
- Behavior Tree handles event disturbances

Handling disturbances

- Success
- Running
- Failure



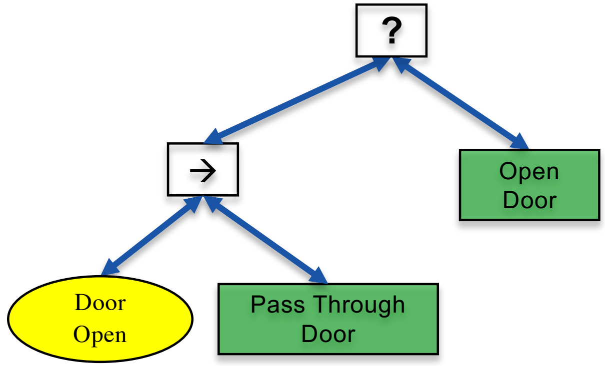
Properties of Behavior Trees :

- Modularity
  - Few dependencies between components (Important for large systems)
  - Optimally modular [1]
- Hierarchical structure
  - Actions exist on many levels of detail (Get tea – opening door – grasp handle – move arm)
  - Hierarchical modularity
- Equally expressive as FSMs [2] (with internal variables)
  - choice a matter of taste (as programming languages)
- BTs generalize [3]
  - Subsumption Architecture
  - Teleo-Reactive Approach
  - Decision Trees

Content

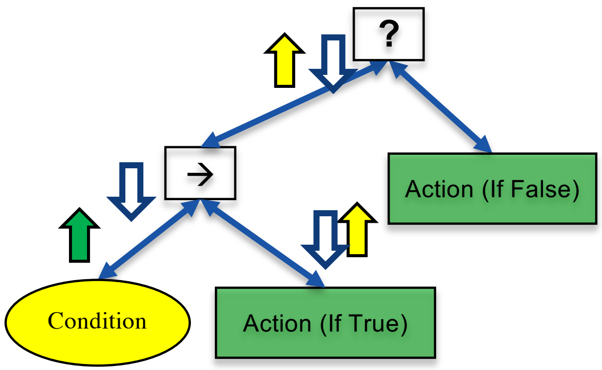
- When to use Behavior Trees (BTs)?
  - When deciding “what to do next”
  - Creating complex controllers/policies
- What are BTs?
  - Hierarchically modular policies
  - Optimally modular [1]
- **How to create BTs?**
  - Improvise
  - Use planning (backward chaining)
- The Big Picture
  - Genetic Algorithms
  - Control Theory (Performance Guarantees)
  - Reinforcement Learning

If-then-else constructs

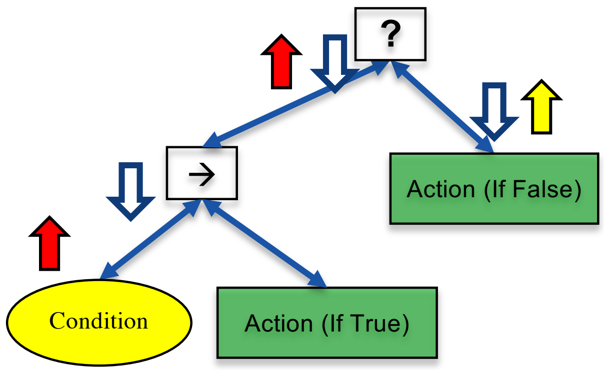


- How to do
  If-then-else?

- If True…



- If False …

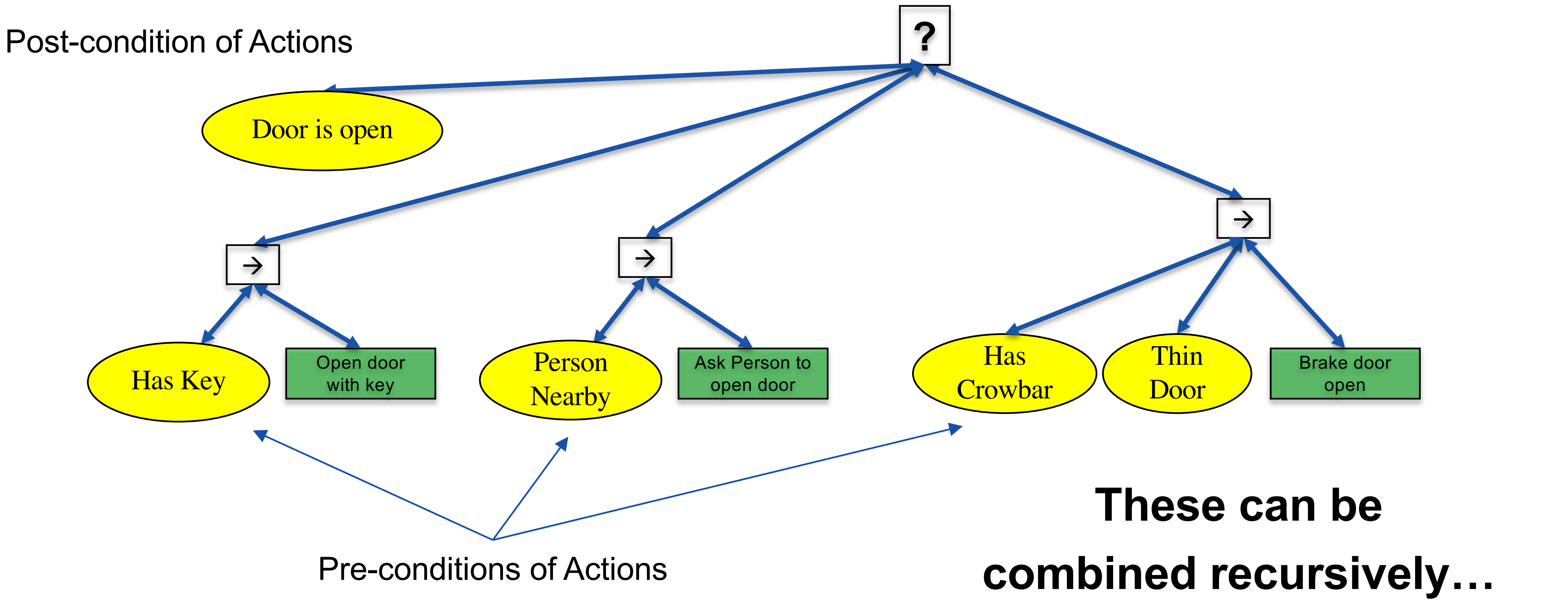


Design BT using Planning (Backward Chaining)

- Backward Chaining
  - Solving an AI Planning Problem by working **backwards from the goal**
- Example:
  - Goal: *Leave the room*
  - To leave I need to **pass through the door**
  - To pass the door I need to **open the door**
  - To open the door I need to **grasp the handle**
  - To grasp the handle I need to **extend my arm**
  - **Plan:**
    - > ***Extend arm***
    - > ***Grasp handle***
    - > ***Open door***
    - > ***Pass through the door***

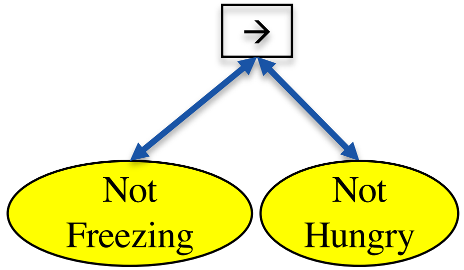
BTs can do this reactively…

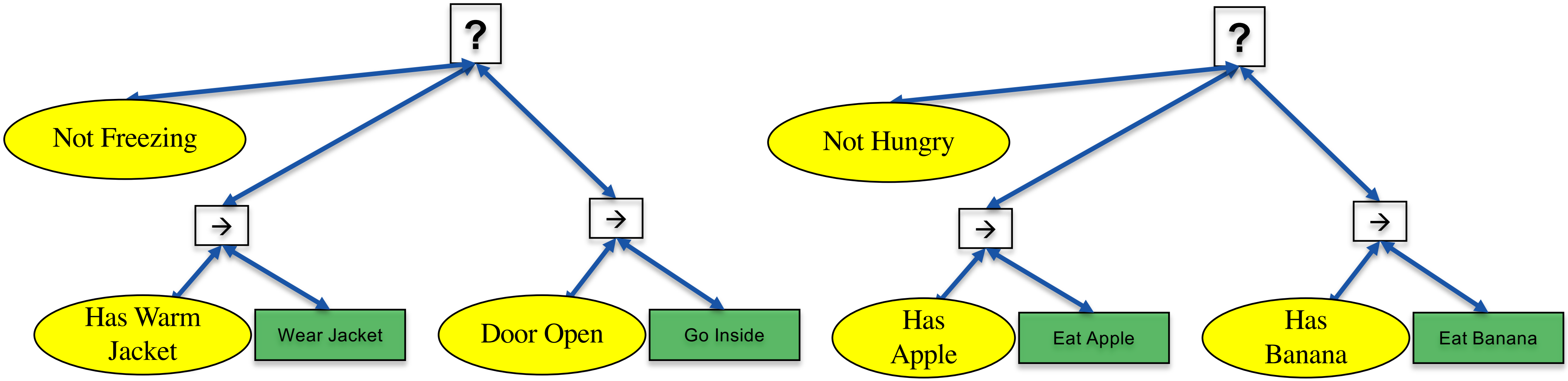
A BT that achieves a single goal (using feedback)



These can be combined recursively…

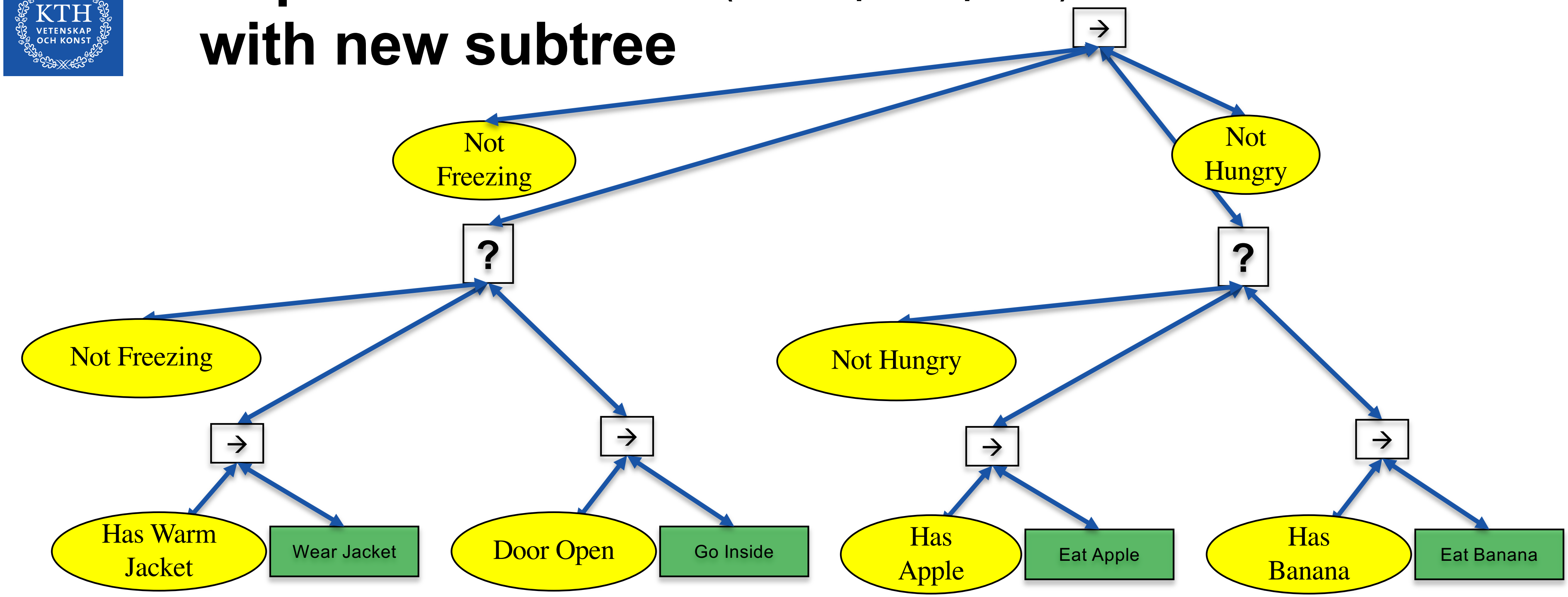
Backward chaining: starting with 2 goal conditions





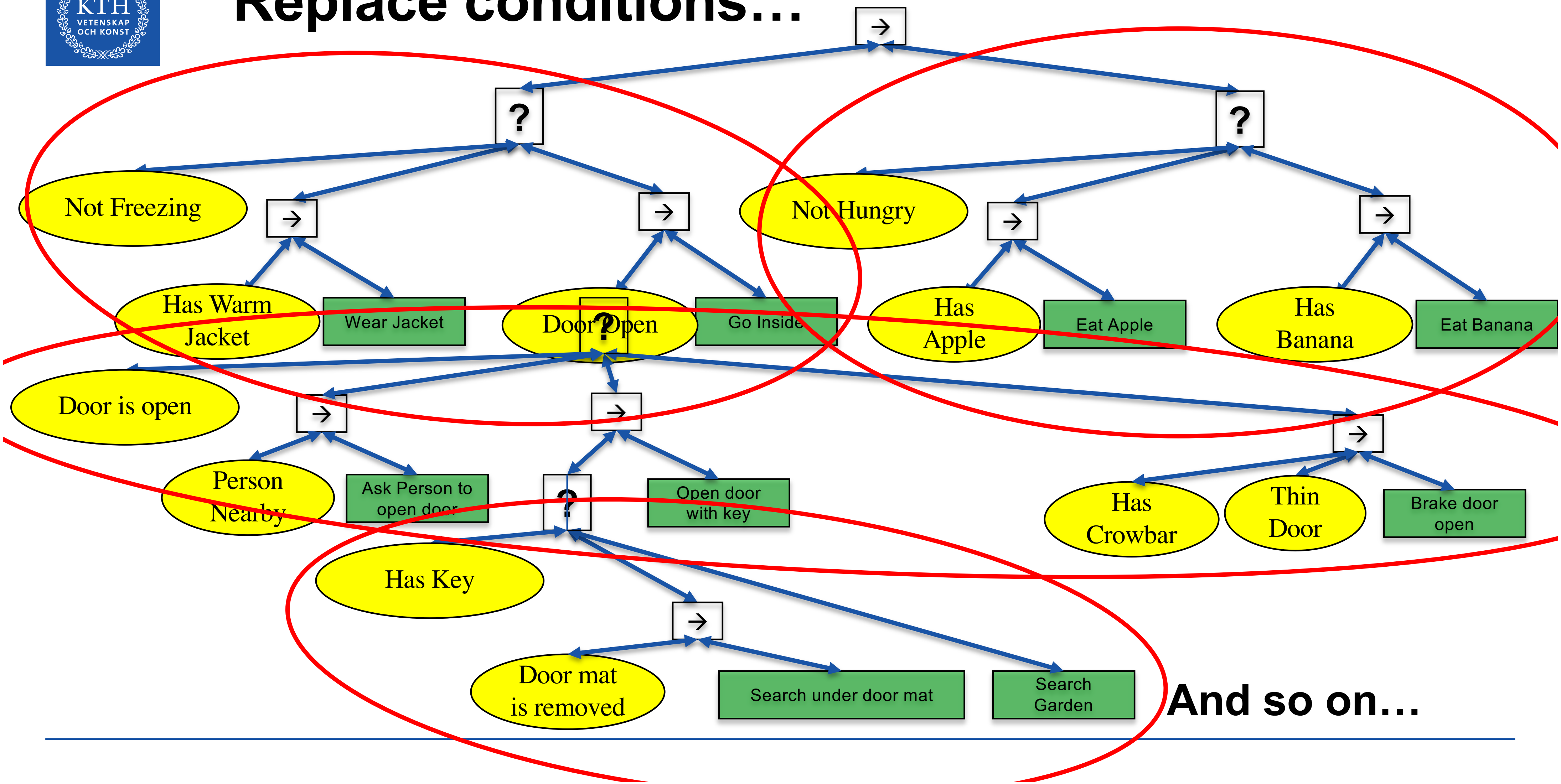
Find BTs that achieve each

Replace condition (with Sequence parent) with new subtree



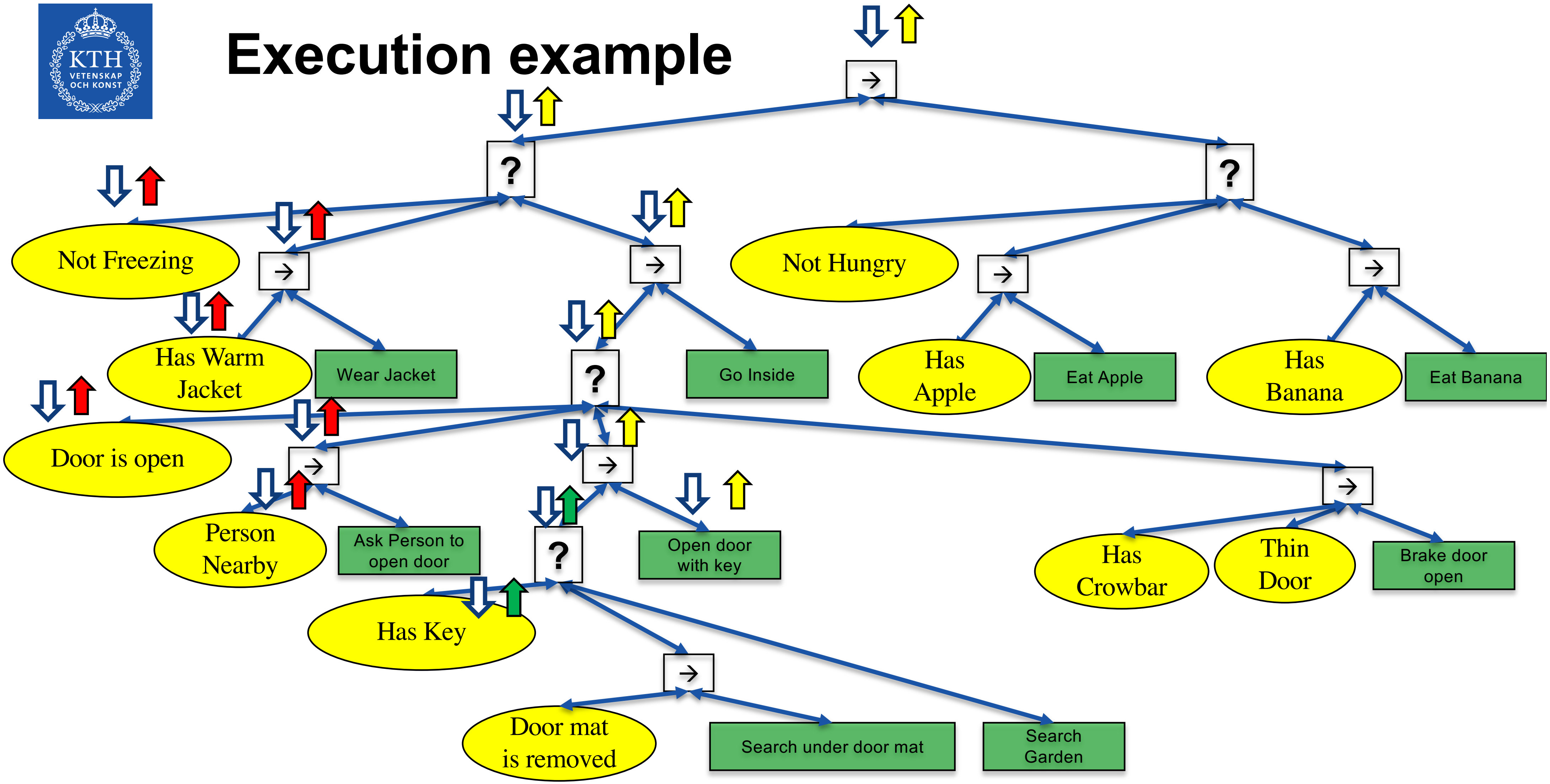
Iterate this…

Replace conditions…

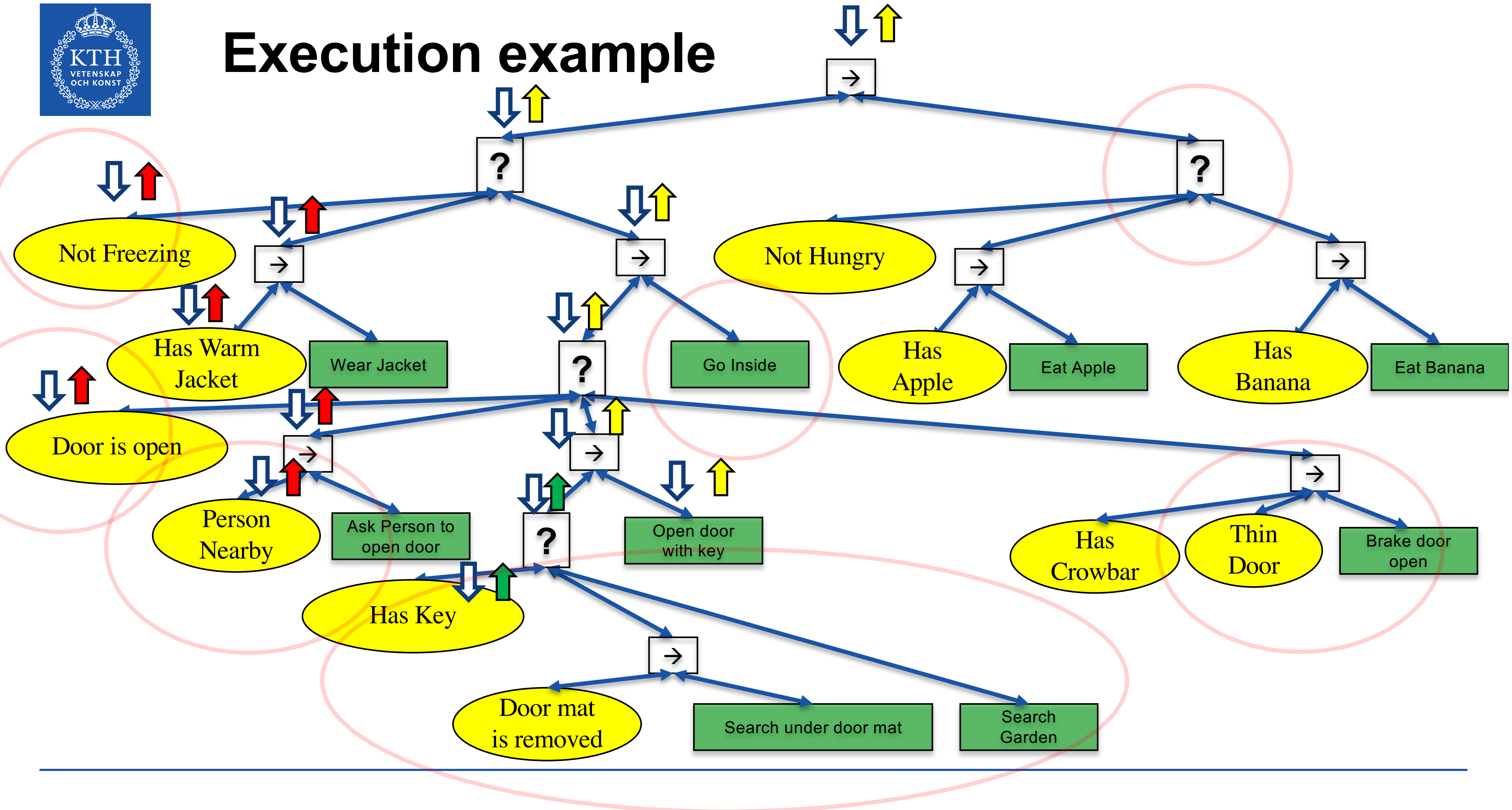


And so on…

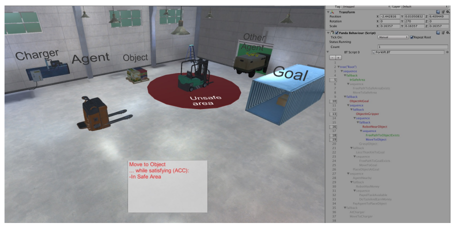
Execution example

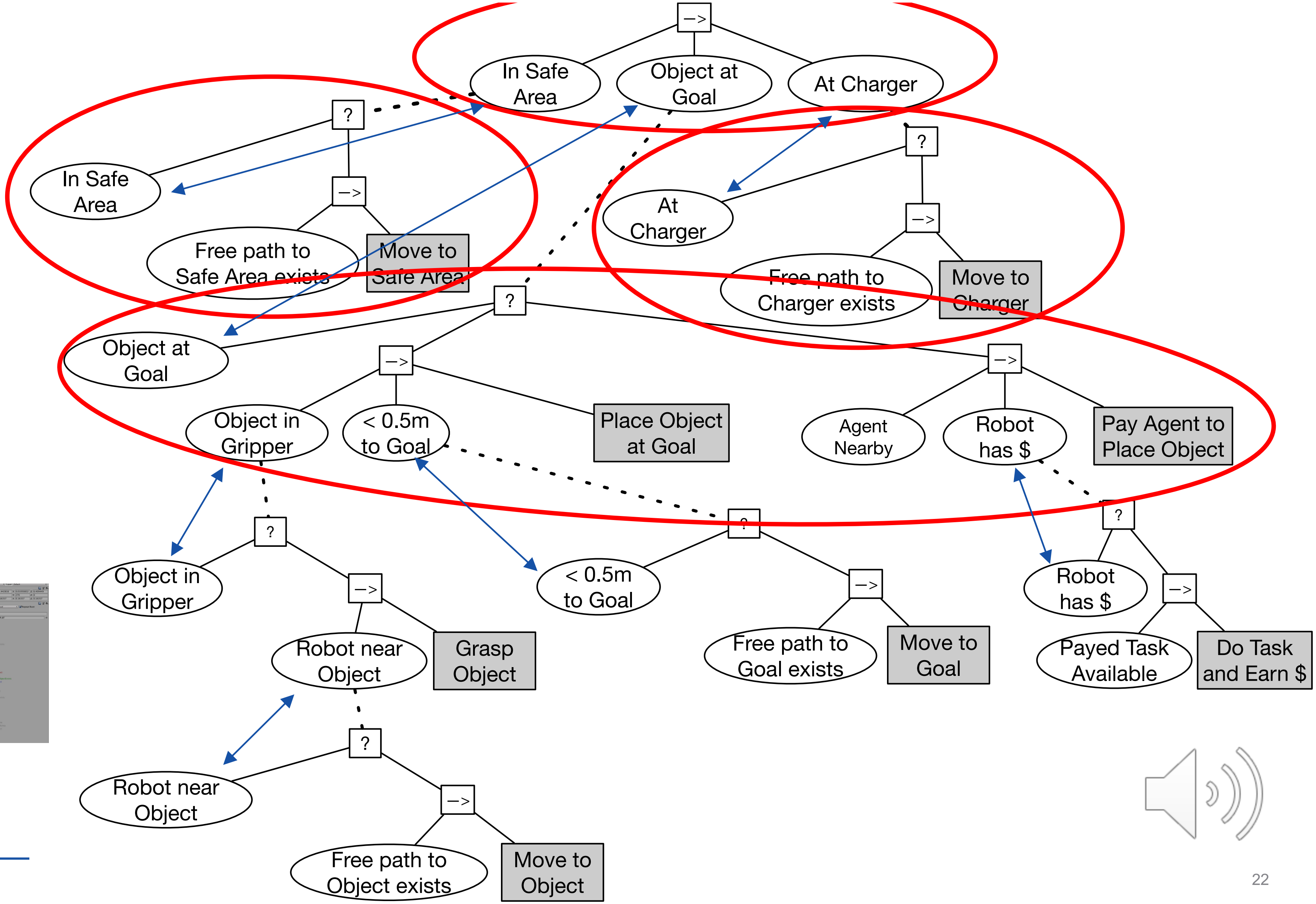


Execution example

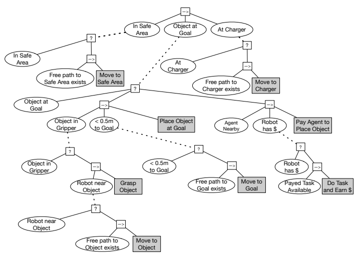


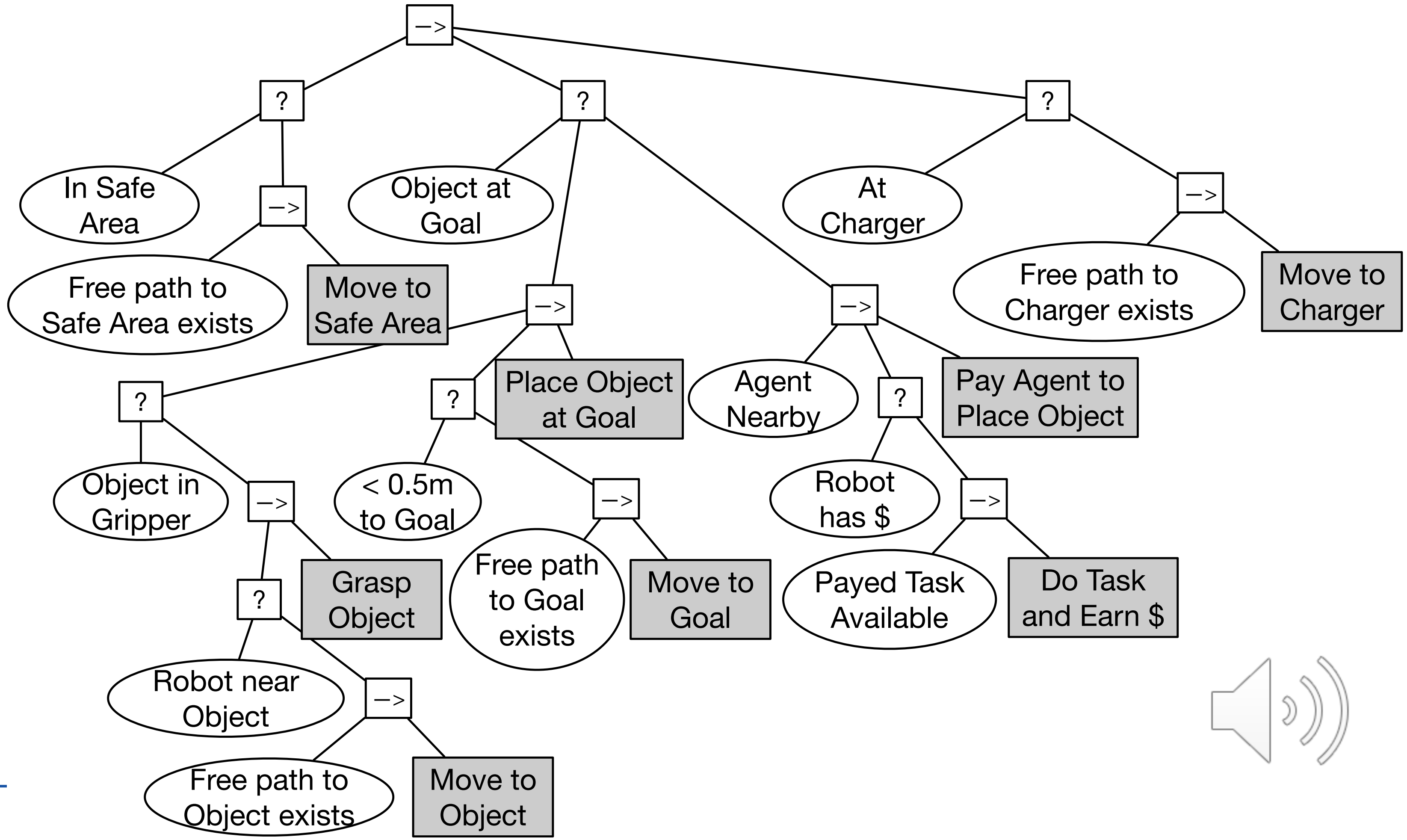
What about this example?



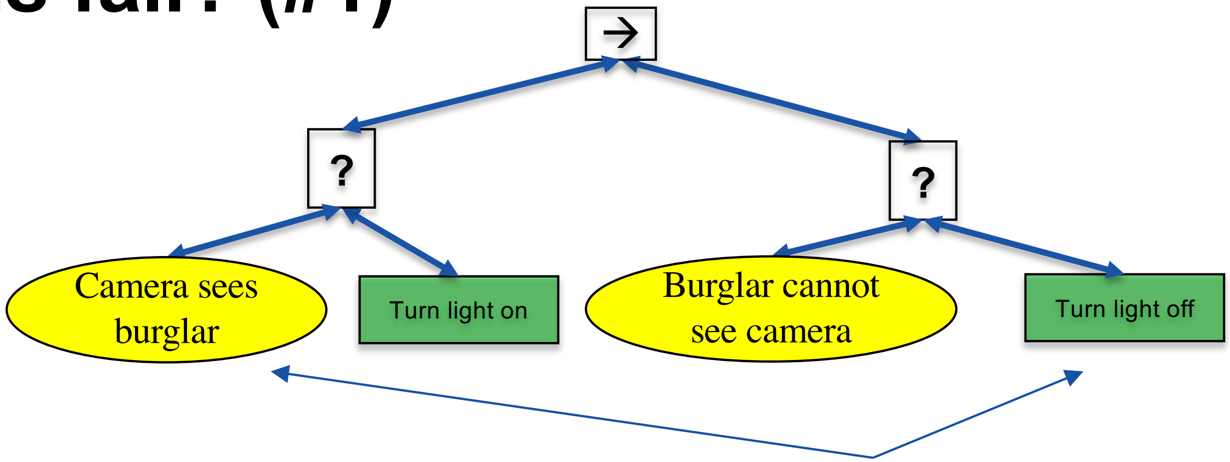


The Backward Chained Behavior Tree

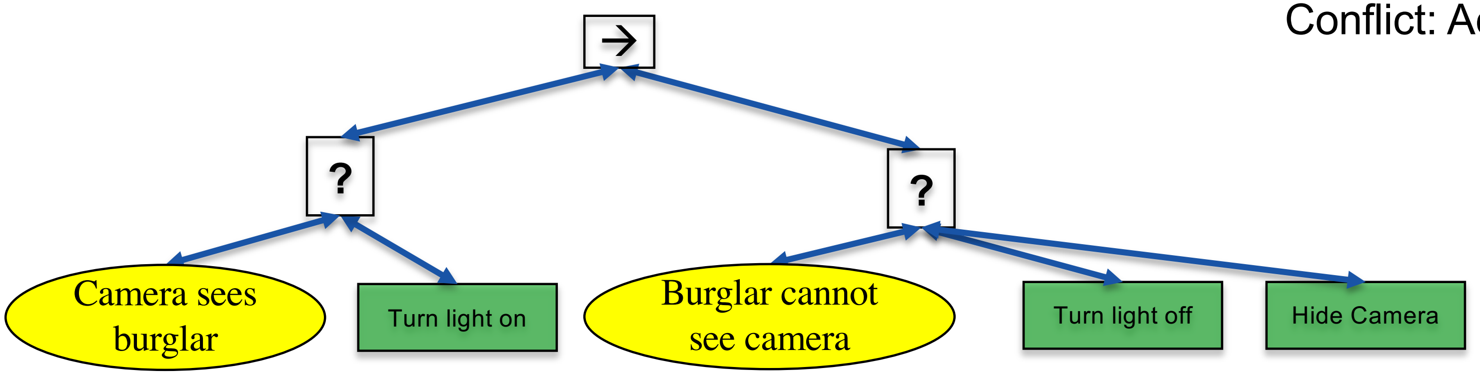




When does this fail? (#1)



Conflict: Action brakes already satisfied Objective

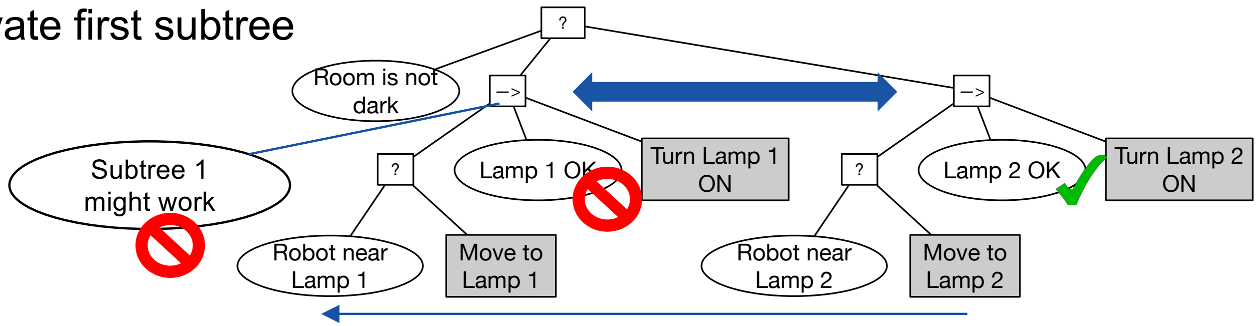


Solution: Avoid braking already achieved goals (if possible)

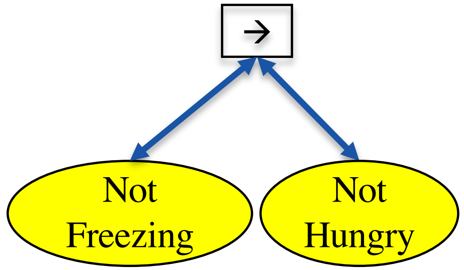
When does this fail? (#2)

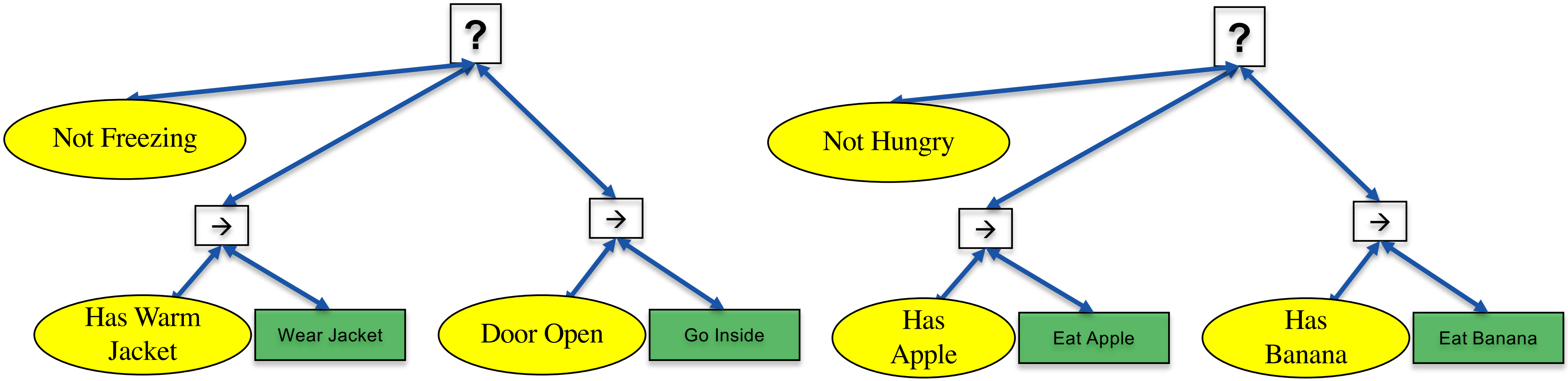
Avoid starting a sub-plan that is doomed to fail

- If Lamp 1 is broken, the policy will still try to move to Lamp 1…
- Solution:
  - Swap order of Fallbacks (so Lamp 2 is first option after initial fail)
  - Add precondition to deactivate first subtree
- When?
  - Count # fails
  - (Use AndOr-tree)

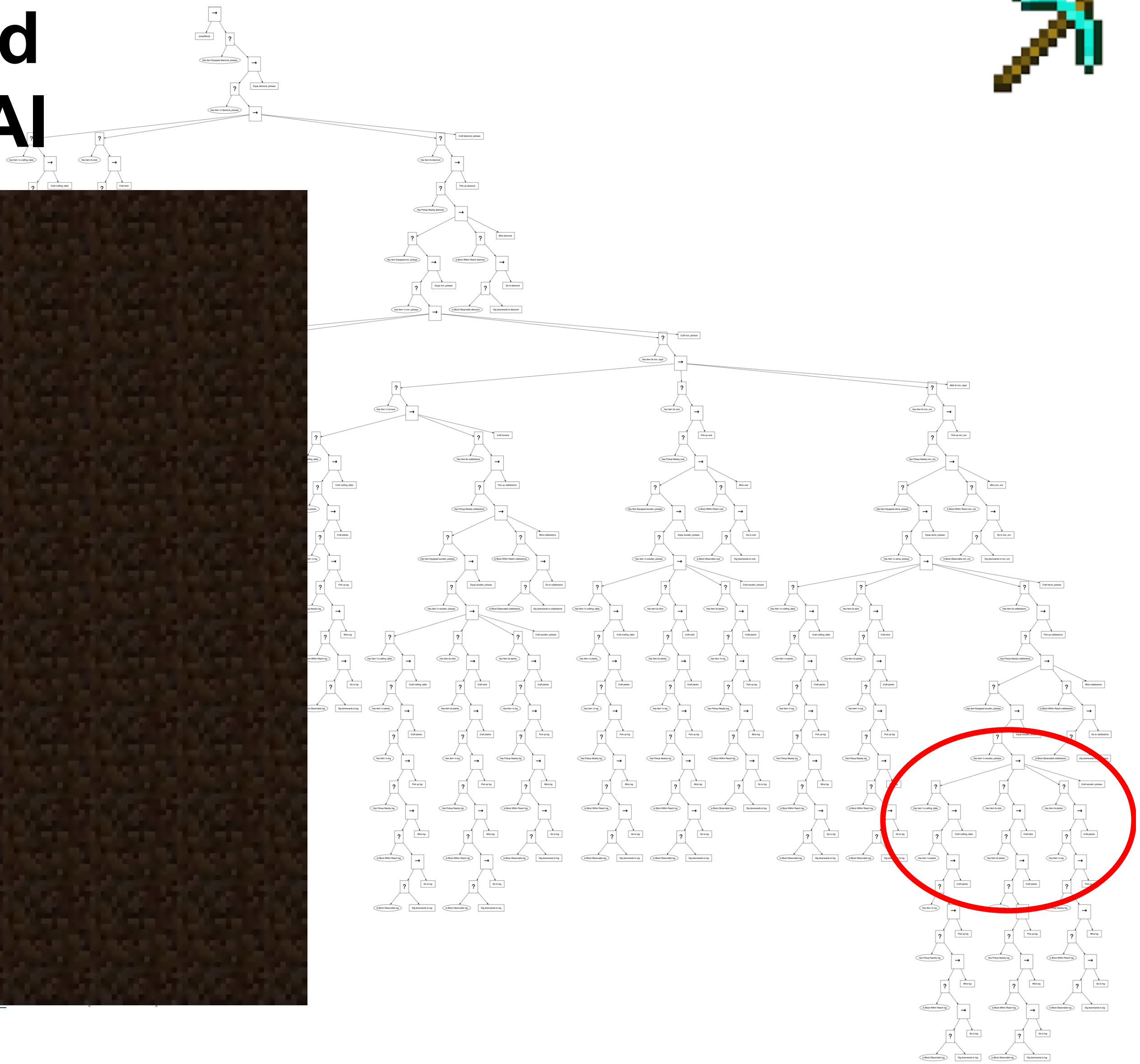


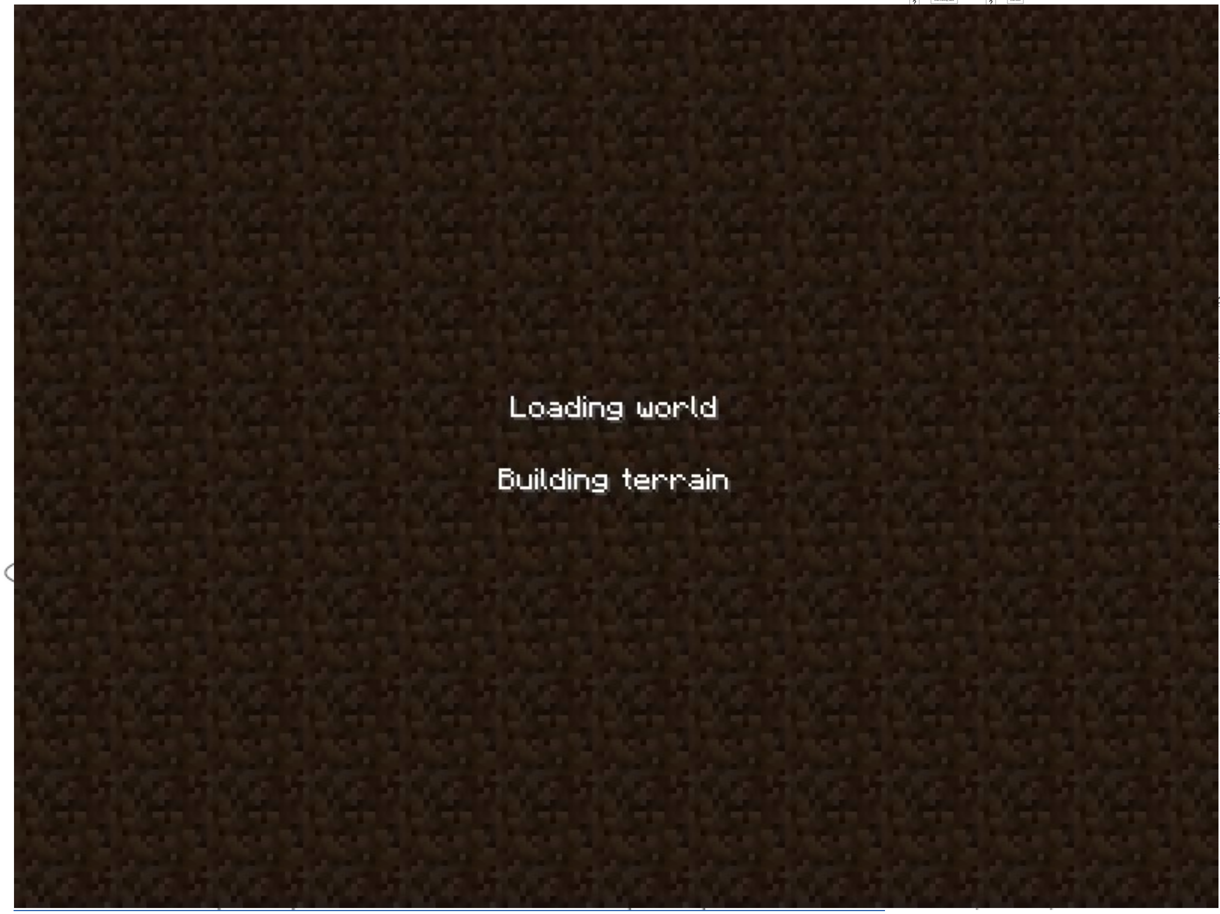
Key idea:
Replace conditions with BT that achieve them!





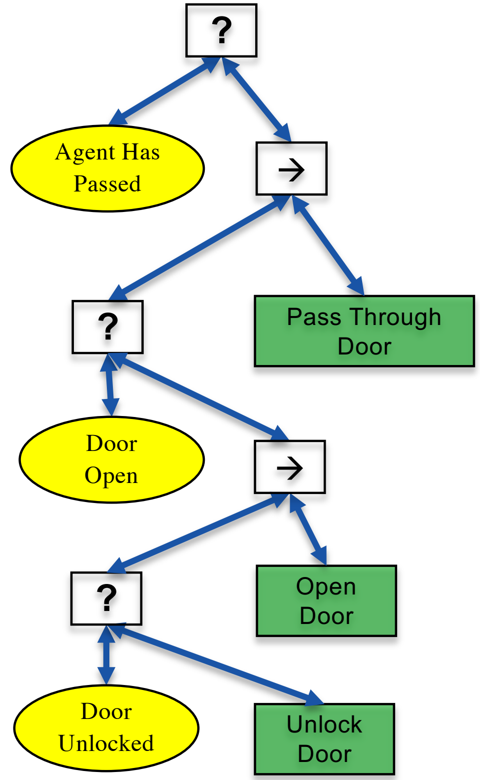
Backward chained BT for Minecraft AI



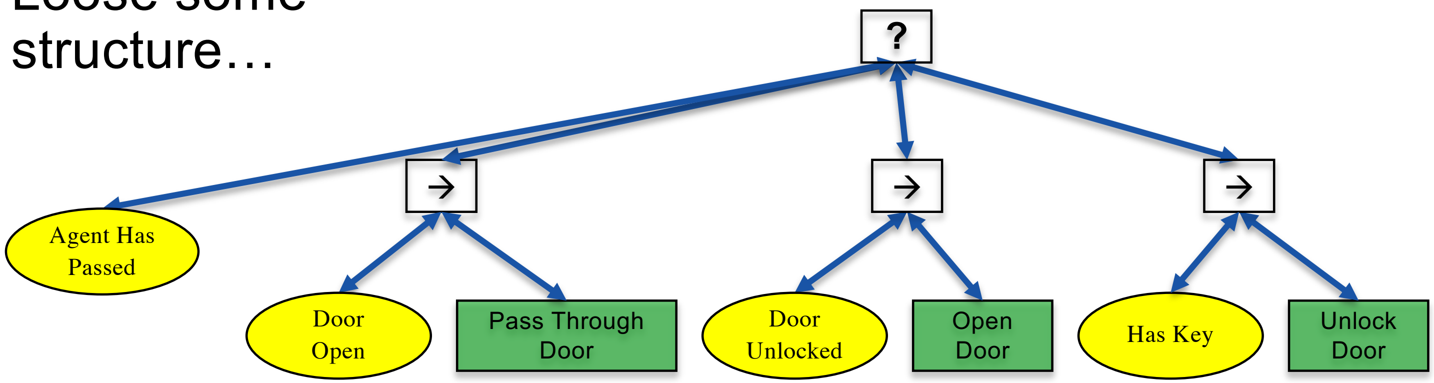


Sometimes we can simplify Backward Chaining (Implicit Sequences)

- Only 2 levels

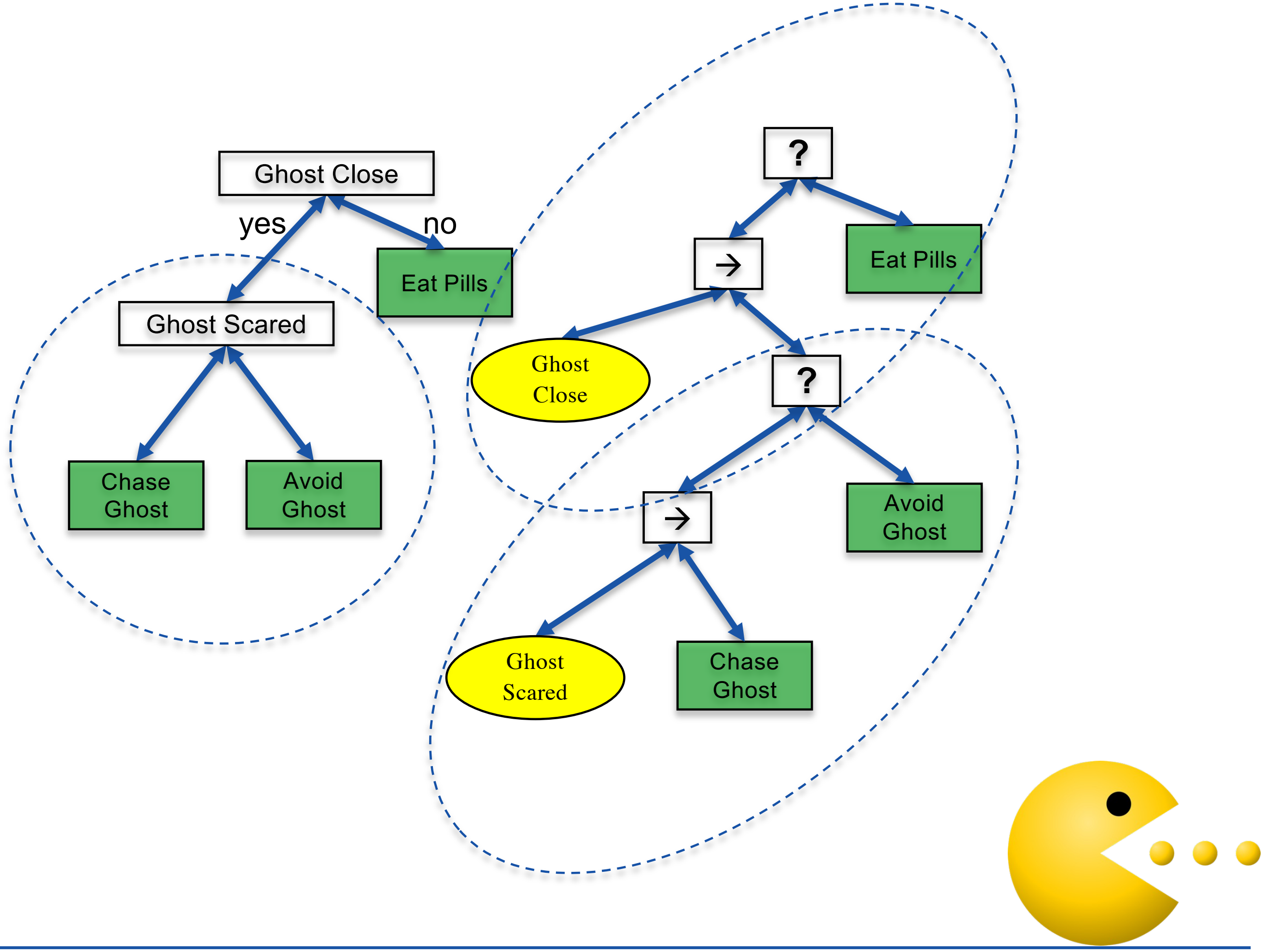


- Works if
  - Action satisfies Condition to Left
- Loose some structure…



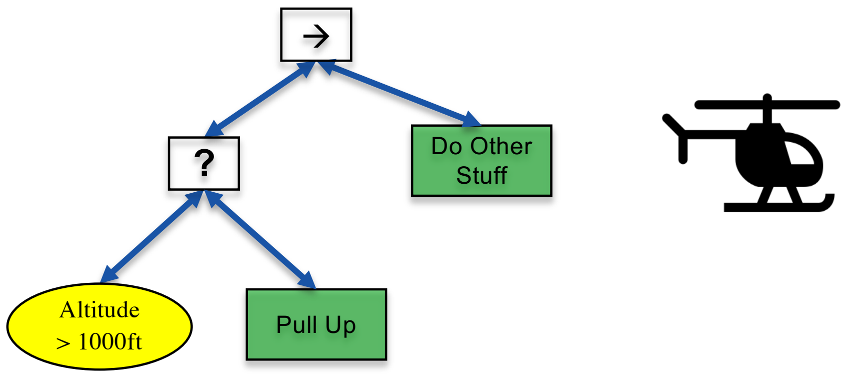
Decision Tree Principle → Divide and Conquer

- Can Behavior be divided into Cases? (and sub-cases)
- Think "Decision Tree"
- Use If-then-else…



Sequences → Improve Safety

- BTs enable Safety-Guarantees using the following construction...

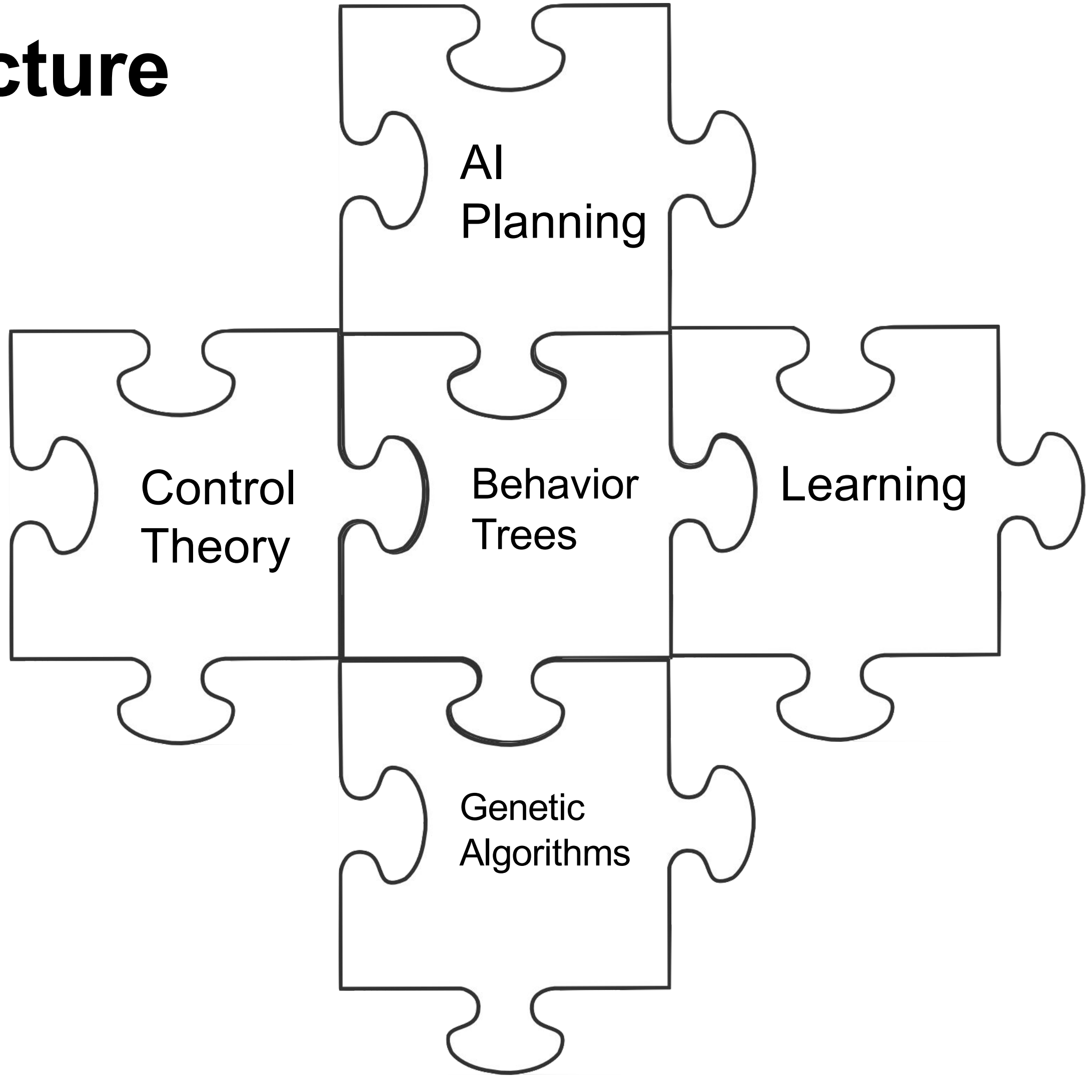


- If-not-then-else…
- Special case of Backward Chaining

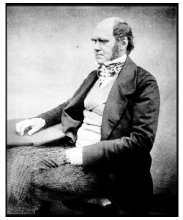
Content

- When to use Behavior Trees (BTs)?
  - When deciding “what to do next”
  - Creating complex controllers/policies
- What are BTs?
  - Hierarchically modular policies
  - Optimally modular [1]
- How to create BTs?
  - Improvise
  - Use planning (backward chaining)
- **The Big Picture**
  - Genetic Algorithms
  - Control Theory (Performance Guarantees)
  - Reinforcement Learning

The Big Picture

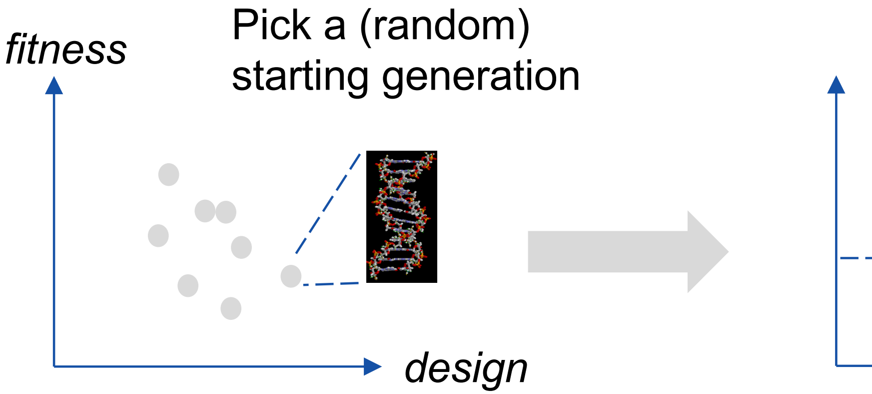


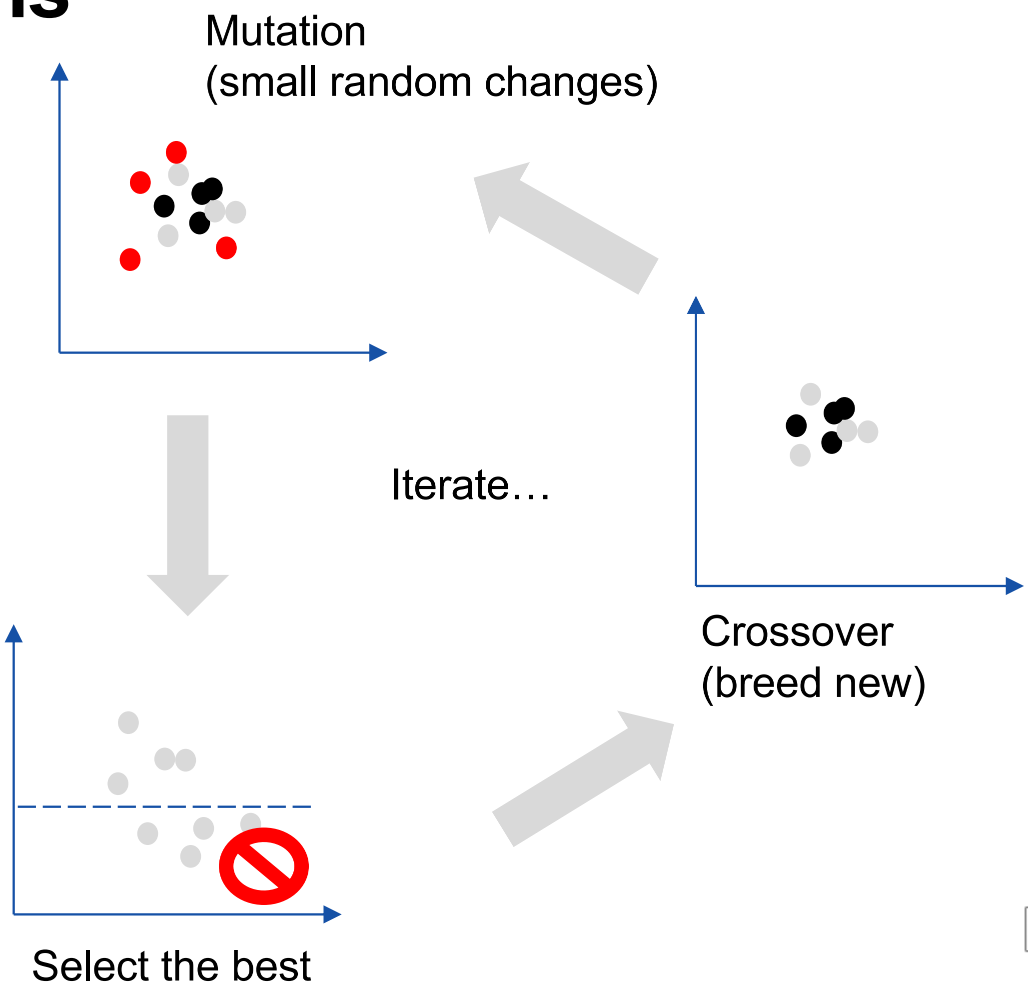
Genetic Algorithms



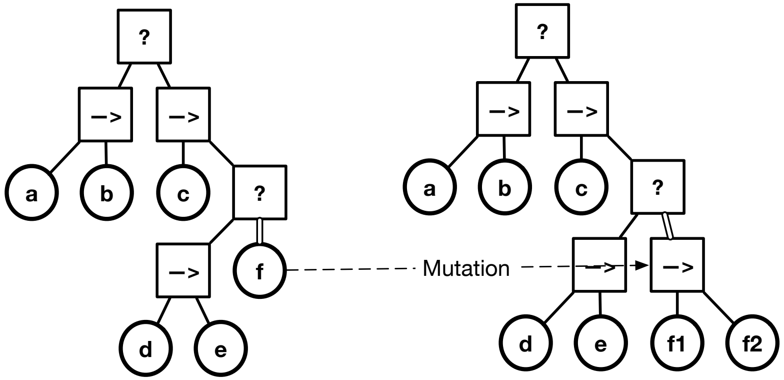
Applies to

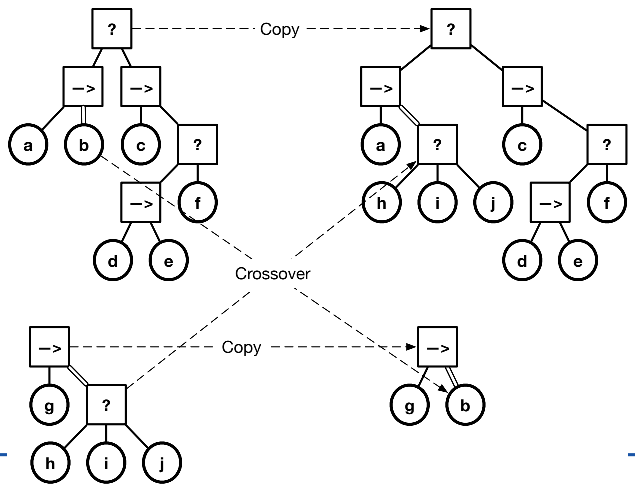
- Viruses
- Humans
- Any Optimization problem
- Behavior Trees

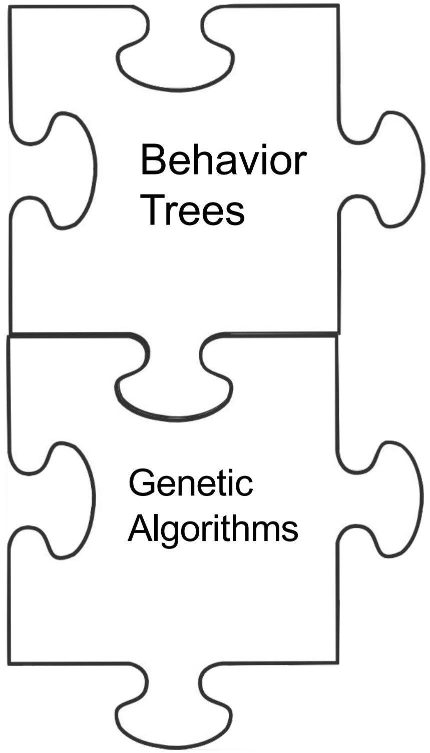




Genetic Algorithms

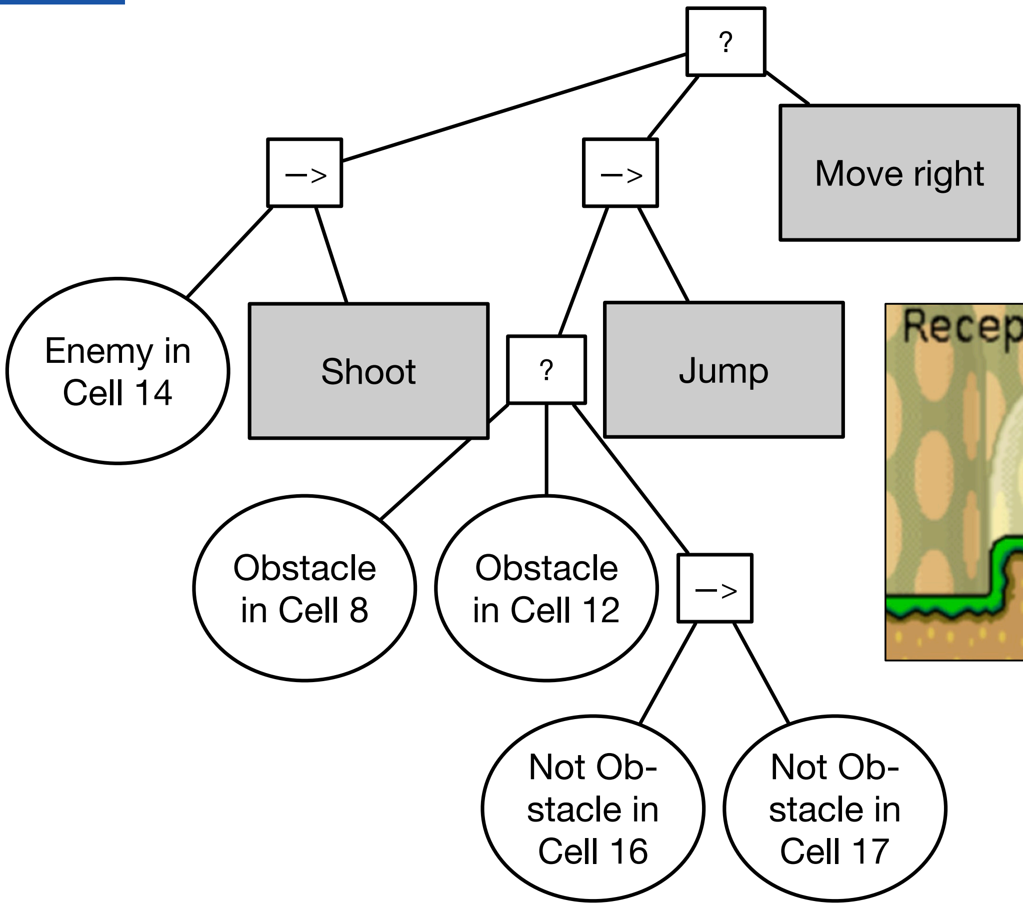


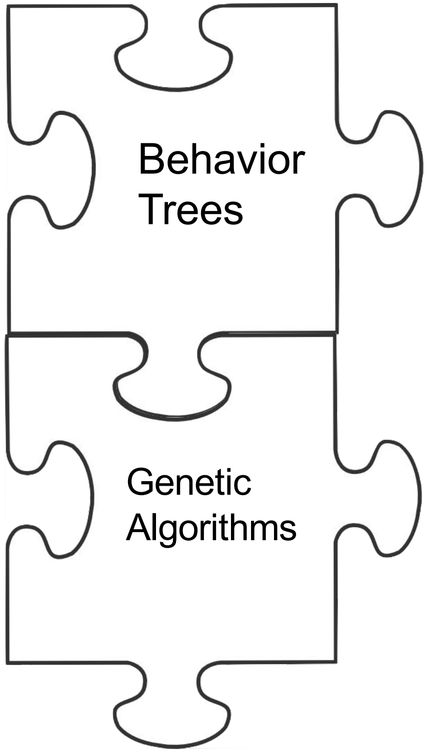




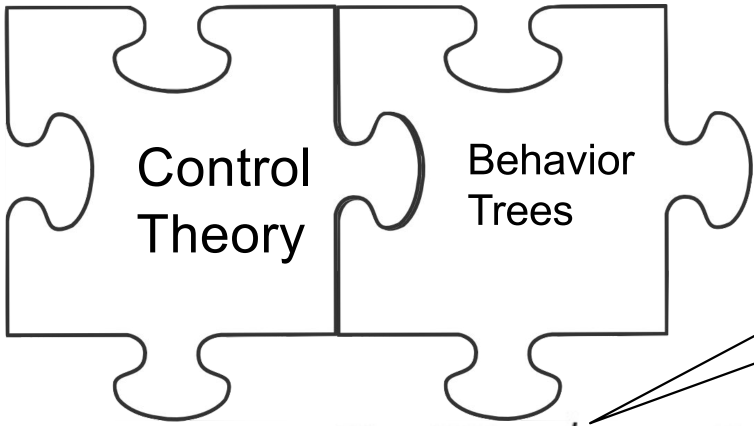
- BT trivially maps to Genes
- Mutation/Crossover easy

Genetic Algorithm for Mario AI BT



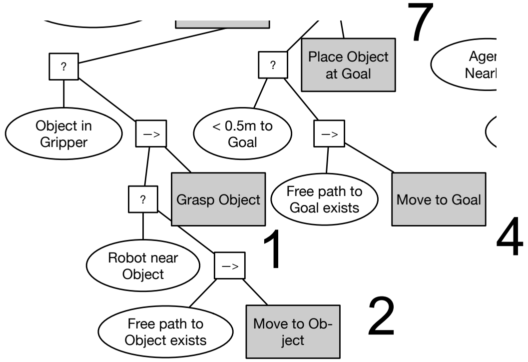


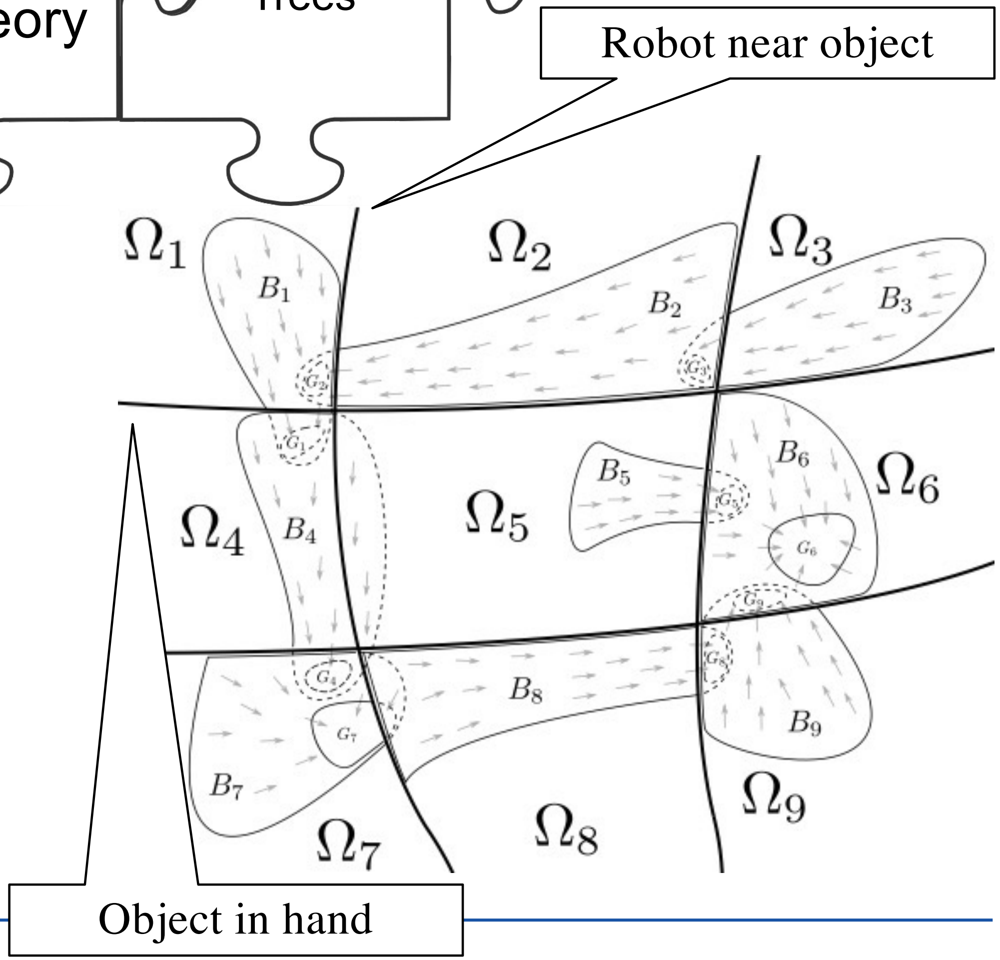
The Big Picture



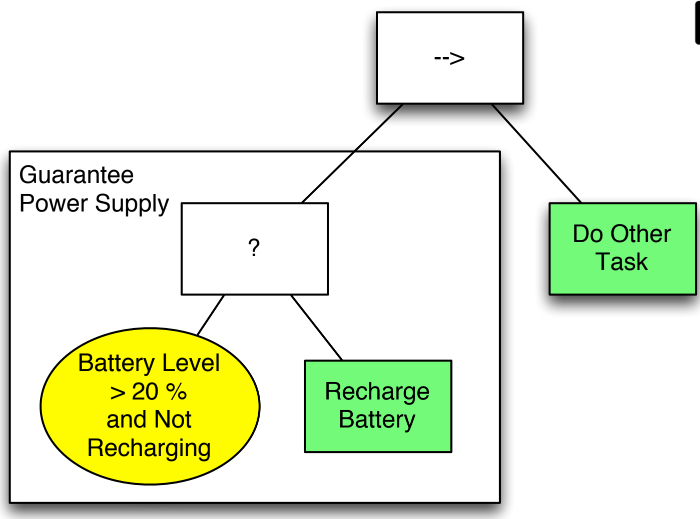
- Let actions have indexes i
- Let Ω_i be states where i executes
- Let B_i be domain of attraction

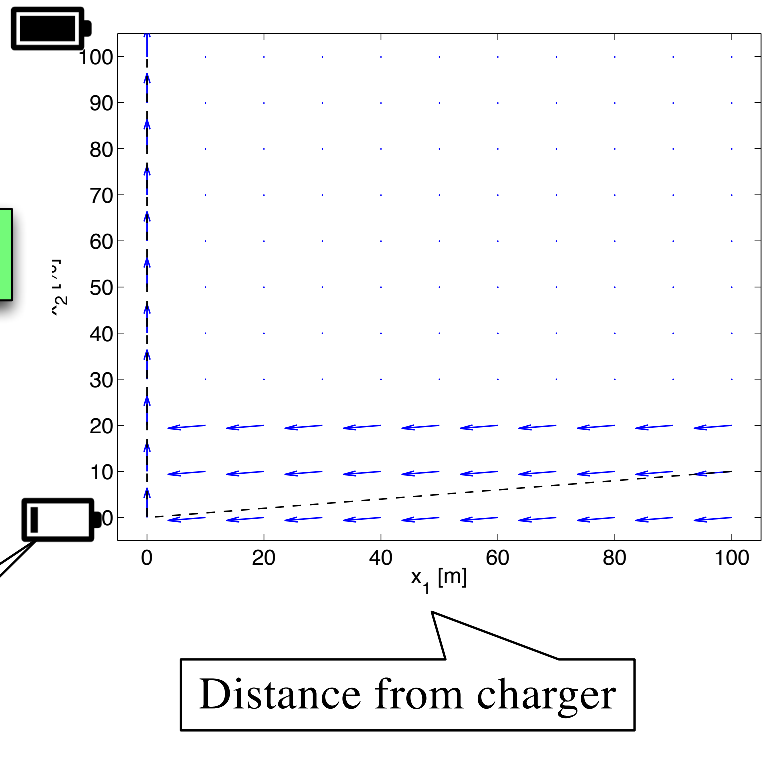
- Then we see that many states end up in G_6 and G_7



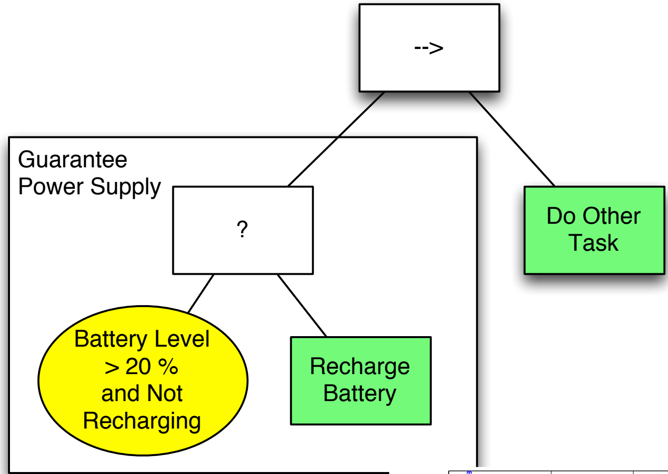


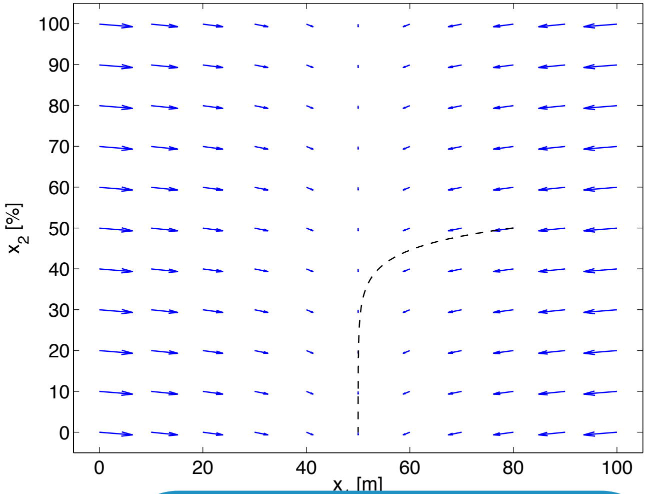
Example: Avoiding Empty Batteries

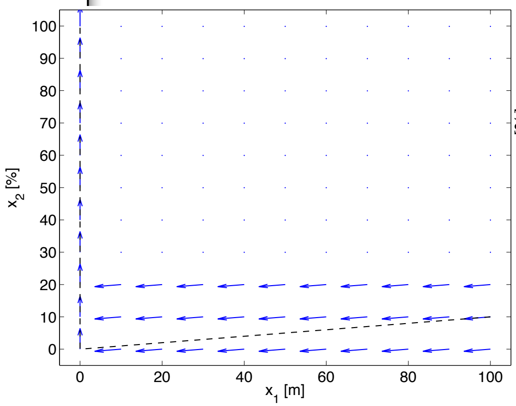


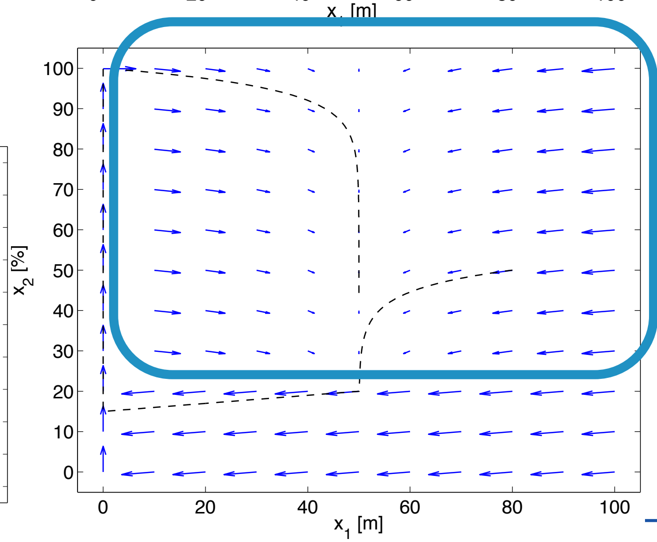


Avoiding Empty Batteries



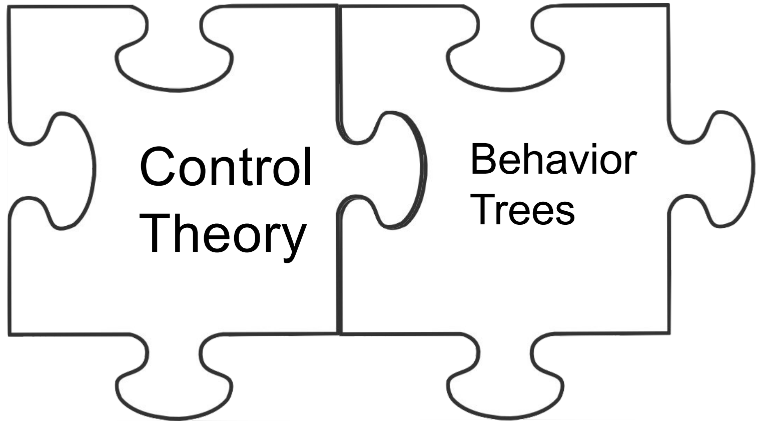






The Big Picture

- Can we give Performance Guarantees?
  - Stability/Convergence?



IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 5, NO. 4, OCTOBER 2020 6073

Convergence Analysis of Hybrid Control Systems in the Form of Backward Chained Behavior Trees

Petter Ögren

***Abstract*—A robot control system is often composed of a set of low level continuous controllers and a switching policy that decides which of those continuous controllers to apply at each time instant. The switching policy can be either a Finite State Machine (FSM), a Behavior Tree (BT) or some other structure. In previous work we have shown how to create BTs using a backward chained approach that results in a reactive goal directed policy. This policy can be thought of as providing disturbance rejection at the task level in the sense that if a disturbance changes the state in such a way that the currently running continuous controller cannot handle it, the policy will switch to the appropriate continuous controller. In this letter we show how to provide convergence guarantees for such policies.**

***Index Terms*—Behavior-based systems, robot safety, control architectures and programming.**

I. Introduction

BEHAVIOR Trees (BTs) were created by computer game programmers as a way to improve modularity and reactivity in the control policies of so-called Non-Player Characters (NPCs) in games [1]. Since then, BTs have been receiving an increasing amount of attention in Robotics [2]–[9]. The reason is that robotics share many high level planning and control problems with game AI, while at the same time, the low

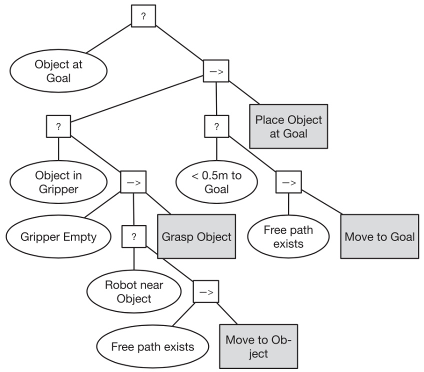
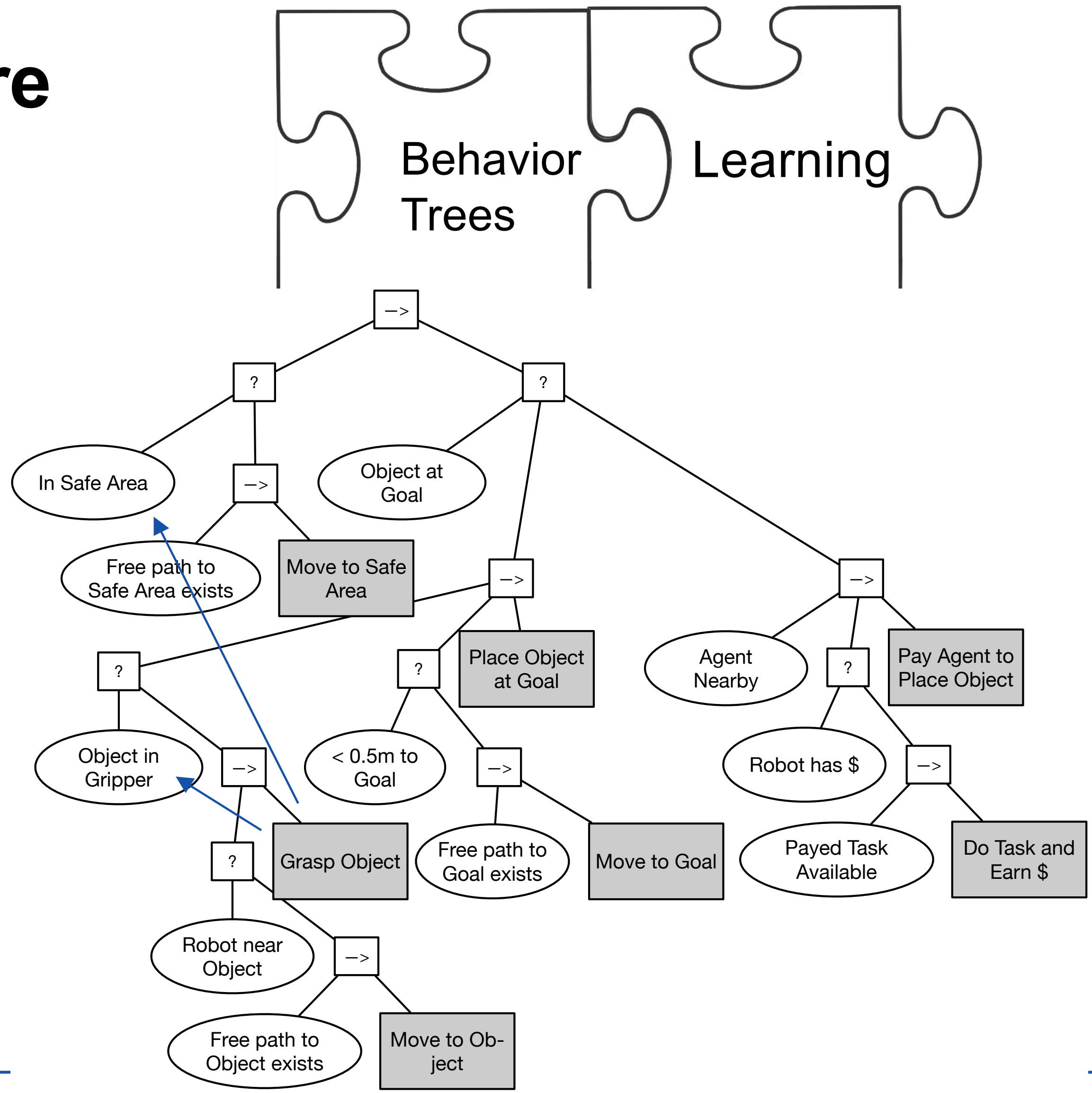


Fig. 1. A BT including the four actions *Move to Object*, *Grasp Object*, *Move to Goal*, *Place Object at Goal*, designed in a way to provide disturbance rejection at the task level. An extended version, including additional objectives and alternative ways to achieve subgoals, can be found in Fig. 5.

The Big Picture

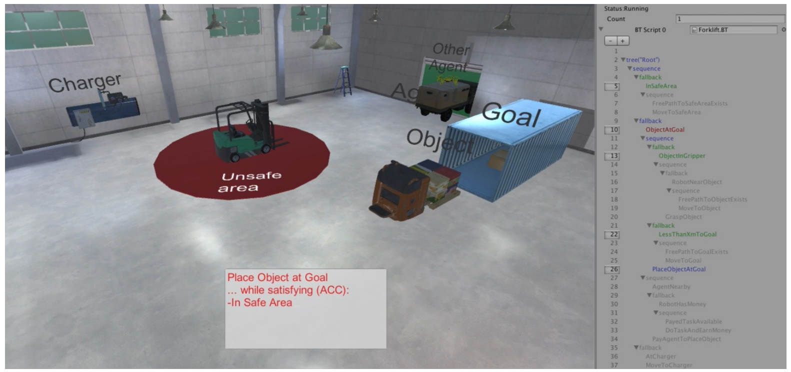
- Some tasks are “easy” other “hard”
- Use Reinforcement learning to do hard task
- Get rewards from BT structure
  - Reach post condition
  - Avoid “wrong” switching

  - (research in progress…)



Content

- When to use Behavior Trees (BTs)?
  - When deciding “what to do next”
  - Creating complex controllers/policies
- What are BTs?
  - Hierarchically modular policies
  - Optimally modular [1]
- How to create BTs?
  - Improvise
  - Use planning (backward chaining)
- The Big Picture
  - Genetic Algorithms
  - Control Theory (Performance Guarantees)
  - Reinforcement Learning



References

- [1] Biggar, Oliver, Mohammad Zamani, and Iman Shames. "On modularity in reactive control architectures, with an application to formal verification." ACM Transactions on Cyber-Physical Systems (TCPS) 6.2 (2022).
- [2] Biggar, Oliver, Mohammad Zamani, and Iman Shames. "An expressiveness hierarchy of Behavior Trees and related architectures." *IEEE Robotics and Automation Letters* 6.3 (2021): 5397-5404.
- [3] Colledanchise, Michele, and Petter Ögren. "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees." *IEEE Transactions on robotics* 33.2 (2016): 372-389.

Questions?