Inteligência Artificial para Robótica Móvel

Arquitetura de Agente
(Máquinas de Estados Finitos
e Behavior Trees)

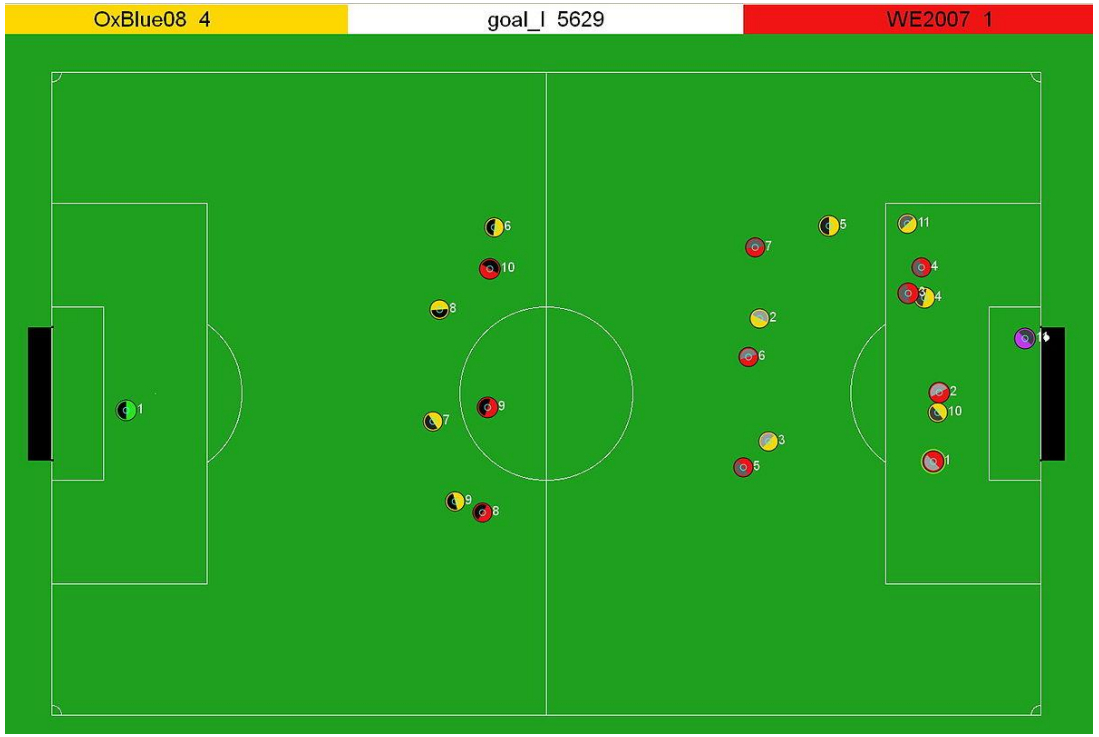
Professor: Marcos Maximo

Roteiro

- Motivação;
- Máquina de Estados;
- Árvores de Comportamento.

Motivação

Como fazer uma IA para jogar futebol?



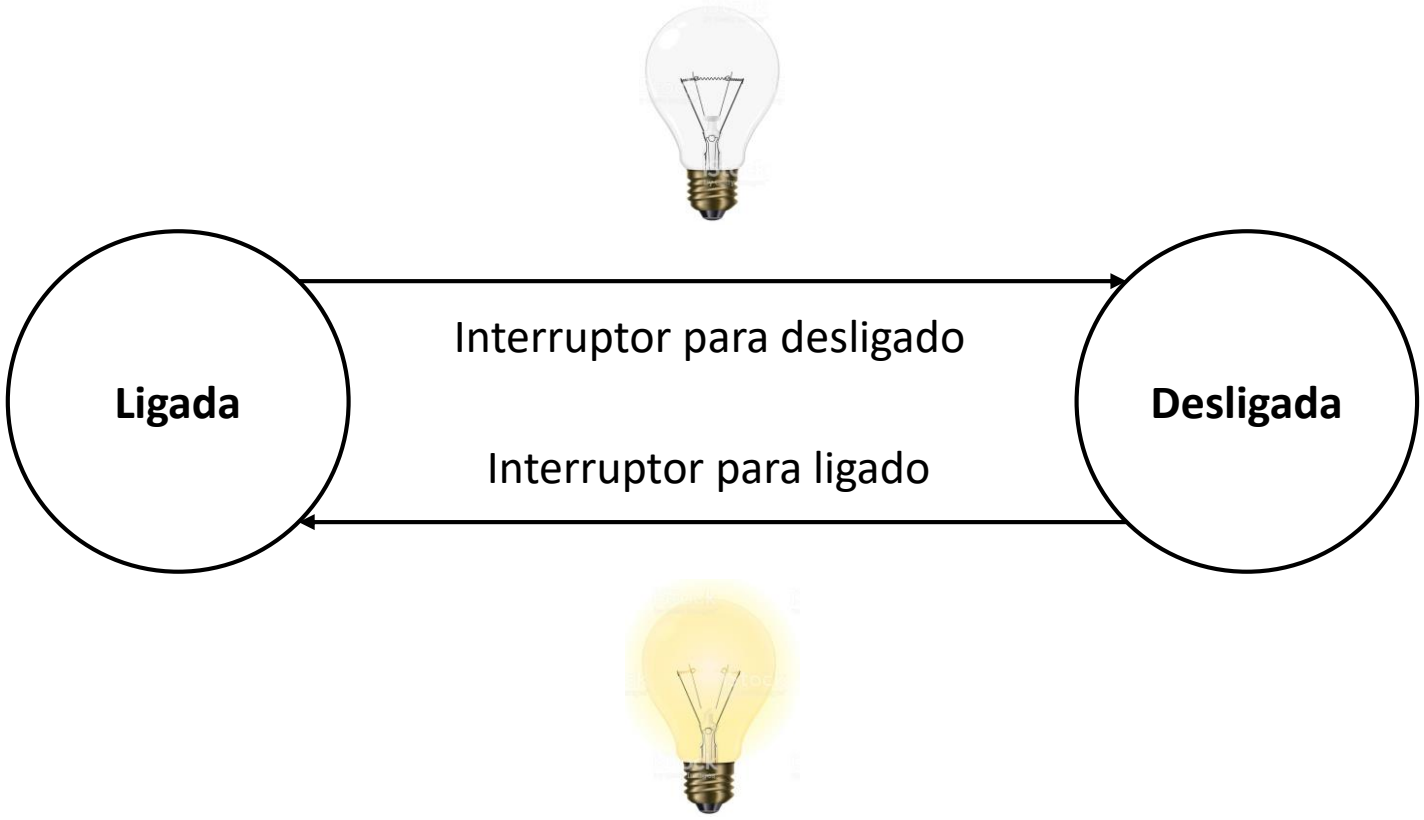
- Difícil pensar de “uma vez”.
- Ser humano tende a pensar em “**situações**”.
- **Dividir** IA em “**comportamentos**” (***behaviors***).
- Necessário **organizar** os comportamentos.
- Comportamentos podem ser **compostos** por comportamentos.

Máquinas de Estados Finitos

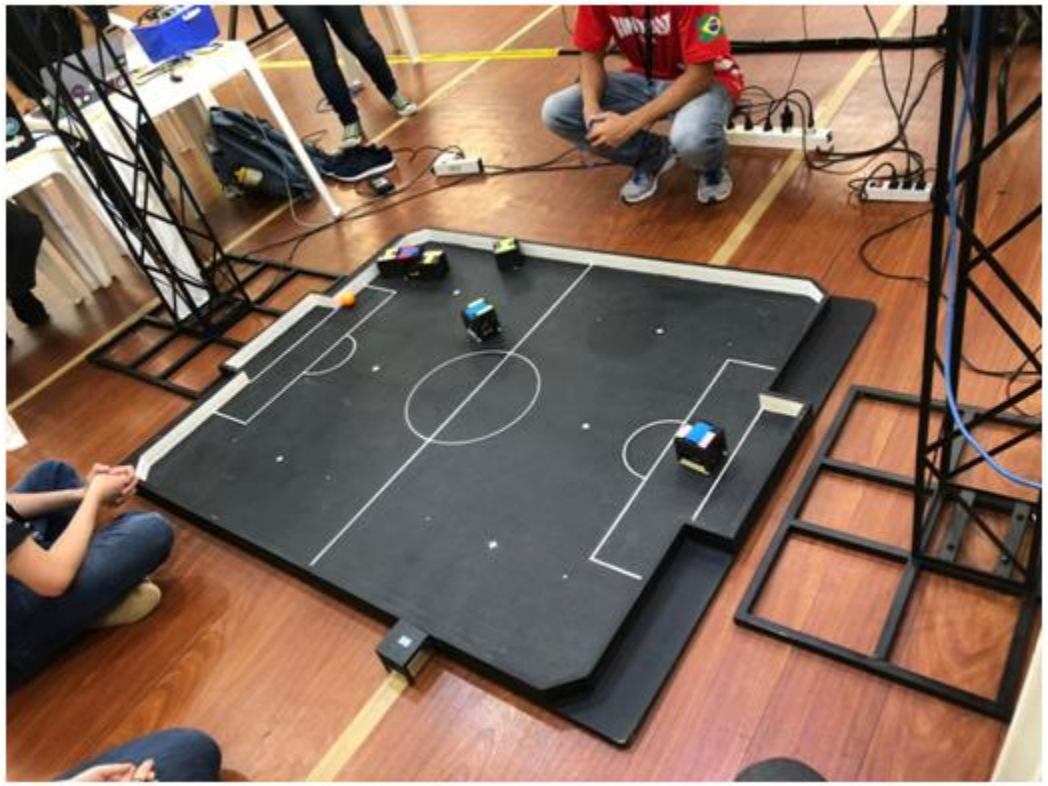
Máquina de Estados Finita

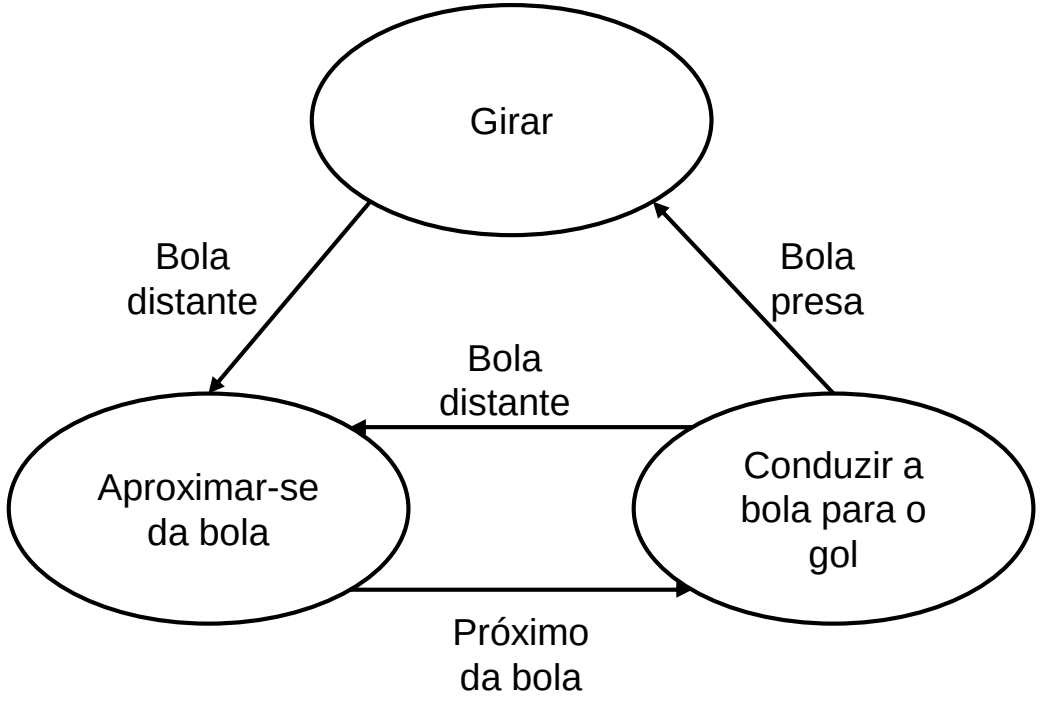
- *Finite State Machine (FSM).*
- Modelo matemático para descrever sistema.
- Apenas um estado a cada momento.
- Acontecimentos levam a mudanças de estado.
- Muitas aplicações em Engenharia e Computação.
- Em IA, cada estado é um comportamento.

Máquina de Estados Finita



Atacante do Very Small Size (VSS)





Máquina de Estados Finita

- Possui limitações teóricas (teoria de autômatos finitos).
- Na prática, memória e máquinas **hierárquicas** (**HFSM**) resolvem limitações.
- HFSM: cada estado pode ser uma FSM.

Máquina de Estados Finita

- **Vantagens:**
  - Intuitiva.
  - Fácil de usar.
  - Praticamente não requer treinamento.
- **Desvantagens:**
  - Difícil gerenciar com IA complexa.
  - Difícil reusar estados (principalmente por conta de transições).

Máquina de Estados Finita

- Antigamente, era muito popular em jogos (e.g. Pacman, Doom, Quake, FIFA, Warcraft).
- Muito popular em robótica
- Muito popular na RoboCup.
- Interessante para modelar IAs “simples”.

Máquina de Estados Finita

- Implementação usando **switch-case**:

```
def change_state():
    if state == State.STATE1:
        if check_condition1():
            self.state = State.STATE2
        elif check_condition2():
            self.state = State.STATE3
    if state == State.STATE2:
        if check_condition3():
            self.state = State.STATE1
```

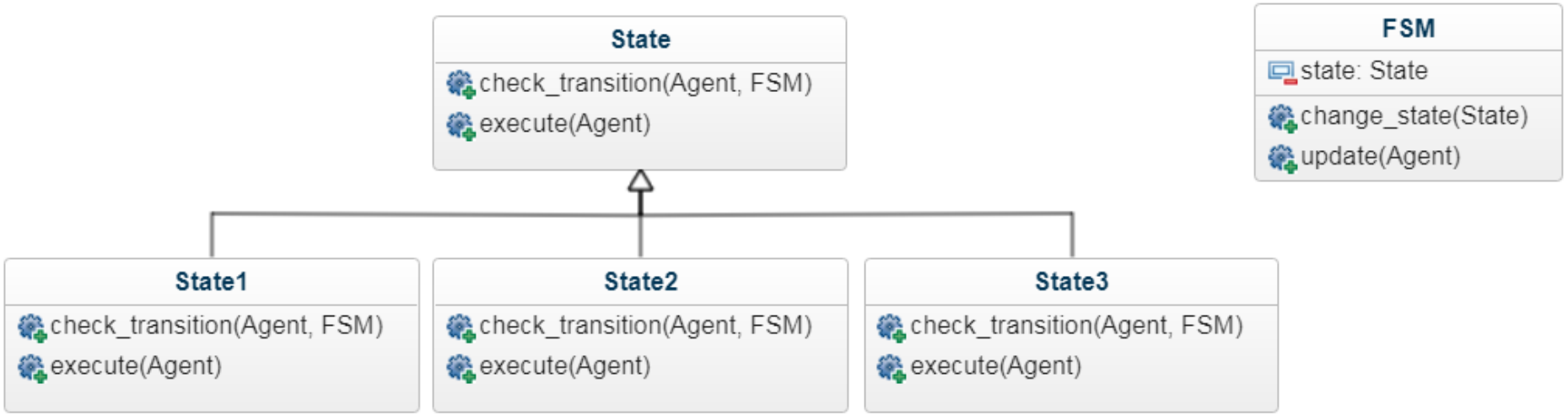
Máquina de Estados Finita

- Implementação usando **switch-case**:

```
def execute_state():
    if state == State.STATE1:
        execute_state1()
    elif state == State.STATE2:
        execute_state2()
    elif state == State.STATE3:
        execute_state3()
```

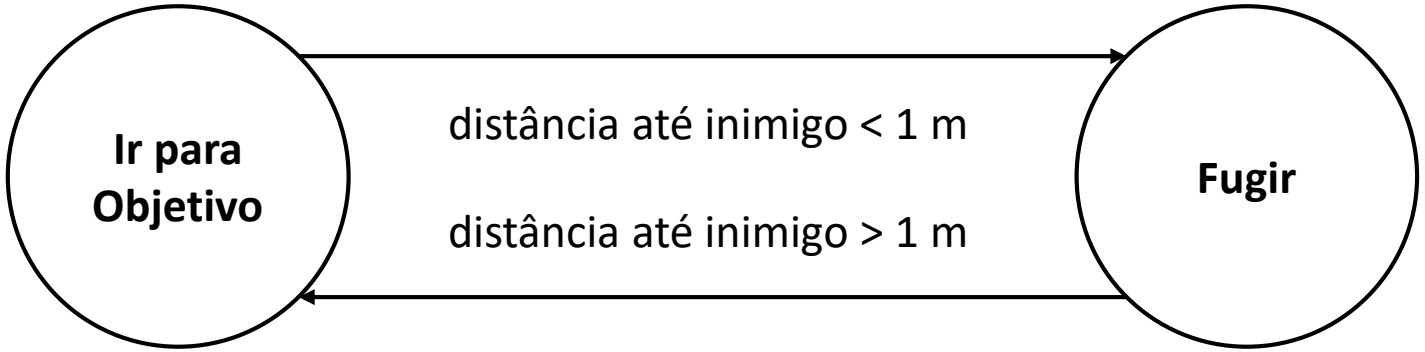
Máquina de Estados Finita

- Com **polimorfismo**:



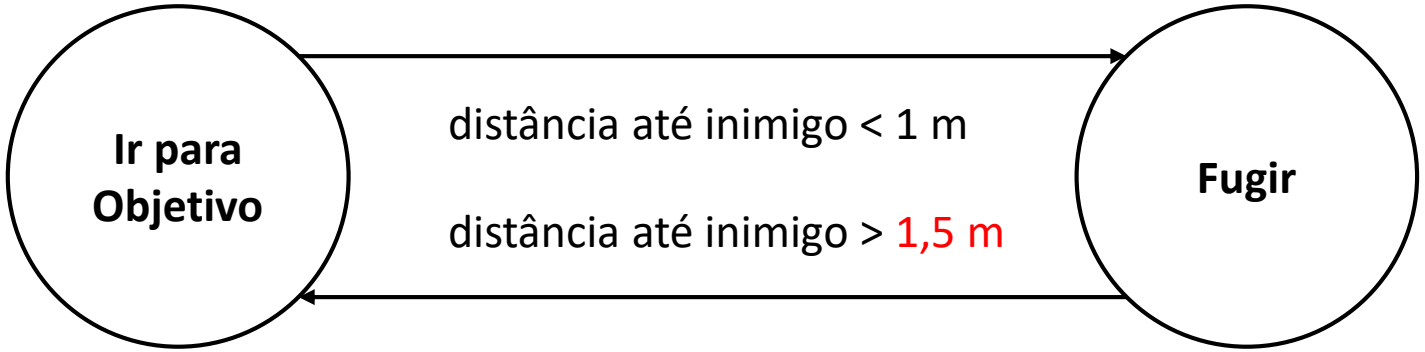
Problema de Histerese

- Agente fica oscilando entre estados e não faz nada!



Problema de Histerese

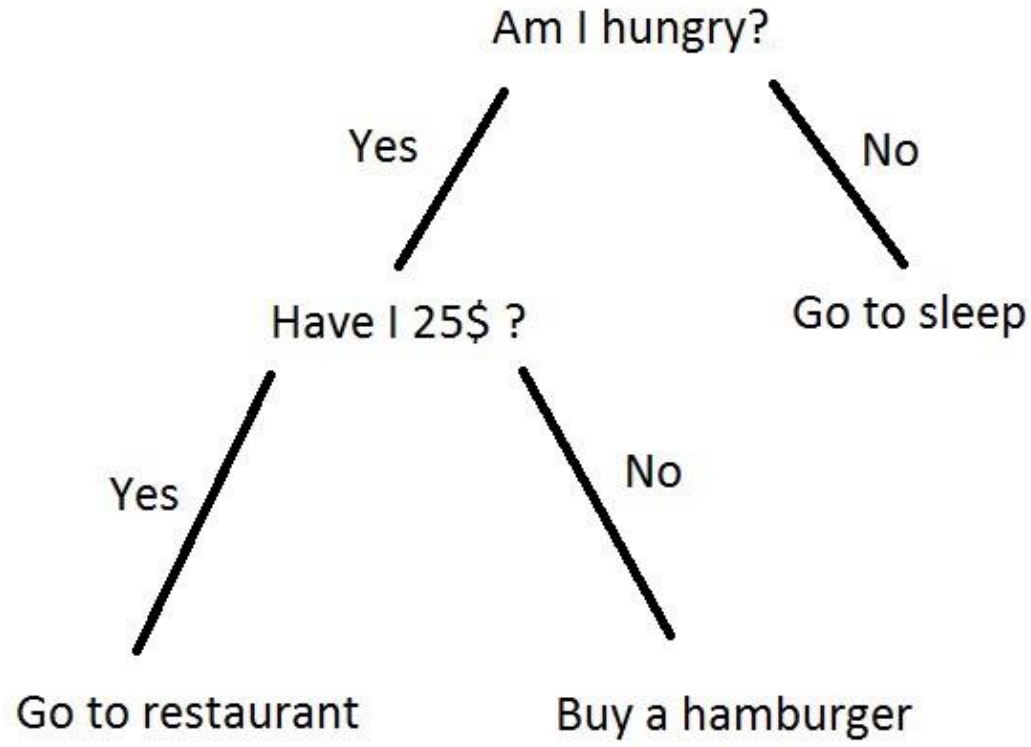
- Solução: **histerese** na transição.



Árvores de Comportamento

Árvore de Decisão

- Percussora de árvore de comportamento.
- Decisões nos nós internos.
- Ação nas folhas.
- Muito usada em *Machine Learning*.



Fonte: https://hackernoon.com/what-is-a-decision-tree-in-machine-learning-15ce51dc445d

Árvore de Comportamento

- *Behavior tree*.
- Alguns autores/*frameworks* chamam os nós de “*tasks*” (tarefas).
- Padrão atualmente no mundo de jogos (famosa após Halo 2).
- *Engines* de jogos famosas como Unreal tem implementação.
- Implementações costumam fazer adição ao formalismo básico.

Árvore de Comportamento

- Folhas são comportamentos mais básicos.
- Composição de comportamentos com nós especiais (parte do formalismo).
- Nós especiais (***decorators***) mudam comportamento de nó sem precisar reescrevê-lo (mais modular).
- A execução de cada nó retorna um de três valores:
  - ***Success*:** tarefa terminou com sucesso.
  - ***Failure*:** tarefa falhou.
  - **Running:** tarefa deve continuar execução na próxima iteração.

Nós Folha (*Leaf Tasks*)

- Dois tipos: **ações** e **condições**.
- Nó condicional verifica condição sem executar ação.

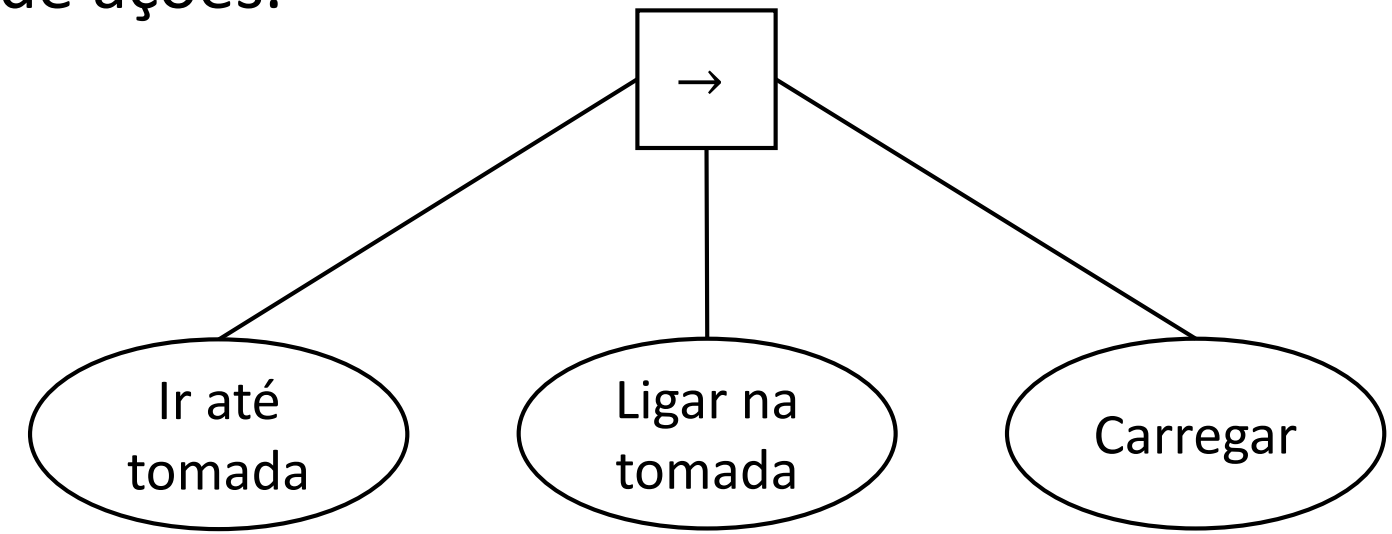
- Nó de ação executa comportamento mais básico.

Nós Compostos (*Composite Tasks*)

- Nós internos da árvore.
- Usados para compor comportamentos.
- 3 tipos principais:
  - ***Sequence*:** executa sequência de comportamentos.
  - ***Selector*:** escolhe um comportamento para execução.
  - ***Parallel*:** executa comportamentos em paralelo.

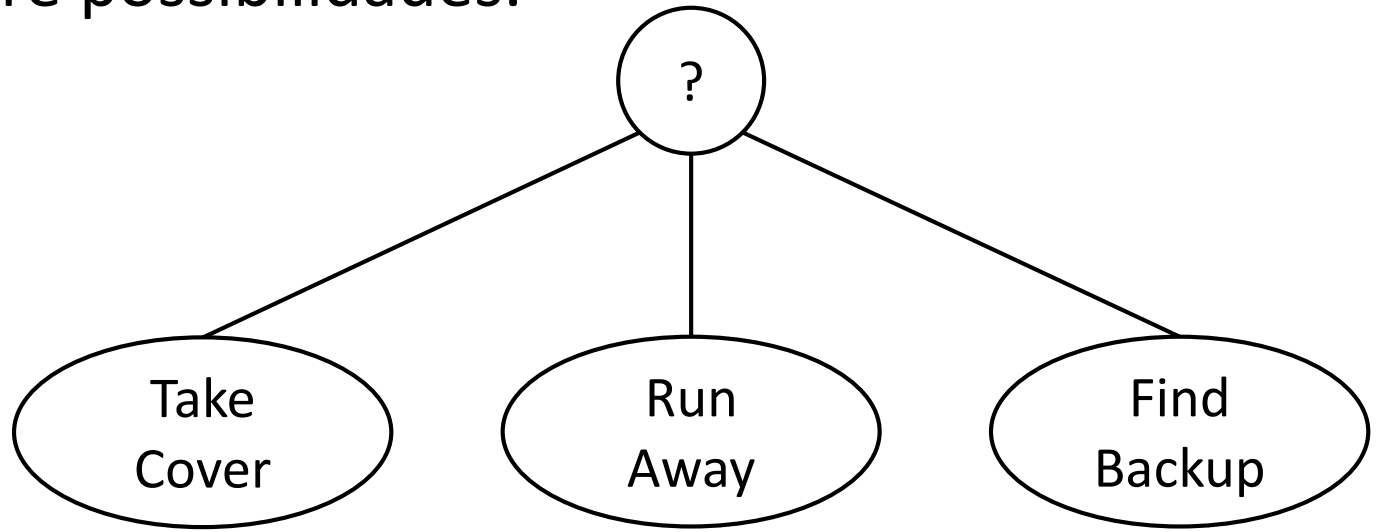
Sequence

- Executa cada filho em sequência.
- Retorna *success* quando o último filho retorna *success*.
- Retorna *failure* se algum filho retornar *failure*.
- Sequência de ações.



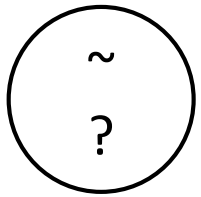
Selector

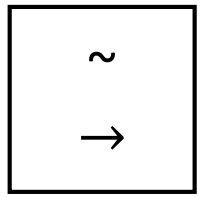
- Executa cada filho em sequência.
- Retorna *success* quando algum filho retorna *success*.
- Retorna *failure* se todos os filhos retornarem *failure*.
- Escolha entre possibilidades.



Random Sequence/Selector

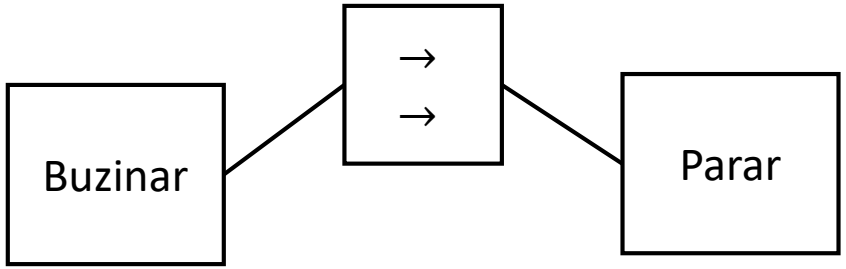
- Variações de *sequence/selector* em que a ordem de execução é aleatória.
- Pode-se definir probabilidades para cada nó.





Parallel

- Executa todos os filhos ao mesmo tempo.
- Implementação pode usar *multi-threading* ou não.
- Política para definir *success* ou *failure*. Exemplos:
  - **Política *sequence***: *failure* assim que algum filho falhar. Se todos os filhos retornam *success*, então o *parallel* retorna *success*.
  - **Política *selector***: *success* assim que algum filho retornar *success*. Se todos os filhos falham, então o *parallel* falha.

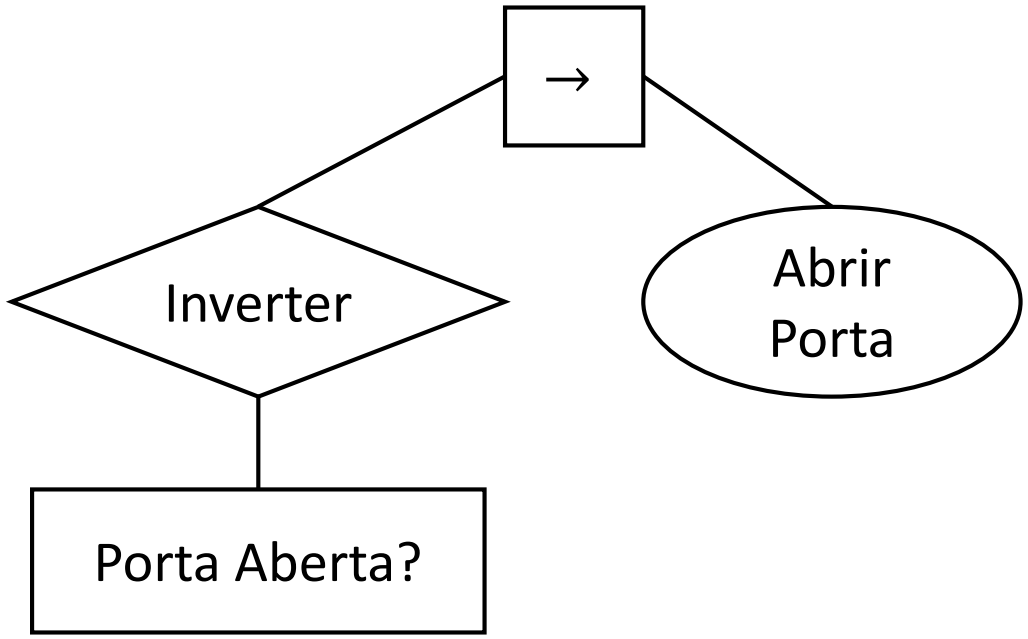


Parallel

- Difícil de usar e de implementar.
- Usos mais comuns:
  - Ações não conflitantes.
  - Verificação contínua de condições: verificar se condições continuam válidas enquanto executa ação (e.g. verificar obstáculo enquanto se move).
  - Comportamento coletivo: controlar grupo de agentes executando simultaneamente.

Decorator

- Nome inspirado no padrão *decorator* de Engenharia de *Software*.
- Modifica comportamento sem precisar alterar sua implementação.
- Facilita reuso.



Alguns Tipos de *Decorators*

- ***Always Fail*:** retorna *failure* independente do resultado da tarefa.
- ***Always Succed*:** retorna *success* independente do resultado da tarefa.
- ***Invert*:** retorna *success* se tarefa retornar *failure* e vice-versa.
- ***Limit*:** executa tarefa até no máximo um número de vezes (evita realizar infinitamente tarefa inútil).
- ***Repeat*:** repetir tarefa um certo número de vezes.
- ***Until Fail*:** executa tarefa até esta falhar, quando então o *decorator* retorna *success*.
- ***Until Success*:** executa tarefa até esta ser bem sucedida, quando então o decorator retorna *success*.

FMS + BT

- Também é possível juntar FMS e BT.
- Um estado de um FMS executar uma BT.
- Uma tarefa de ação de uma BT executar uma FMS.

Blackboard

- Um conceito comum em BT é o de *blackboard*.
- Memória compartilhada entre os comportamentos.
- Cada comportamento pode ler ou escrever do *blackboard*.
- Em geral, estrutura tipo dicionário (chave-valor).

Chave	**Valor**
Mais próximo da bola?	Sim
Companheiro mais próximo	2
Oponente mais próximo	4
Posição no campo	(10,2; 5,2)

Para Saber Mais

- Uso de *behavior trees* em jogos: https://www.youtube.com/watch?v=6VBCXvfNlCM

Laboratório 1

Laboratório 1

- Implementar comportamento de um Roomba (simplificado).

Laboratório 1

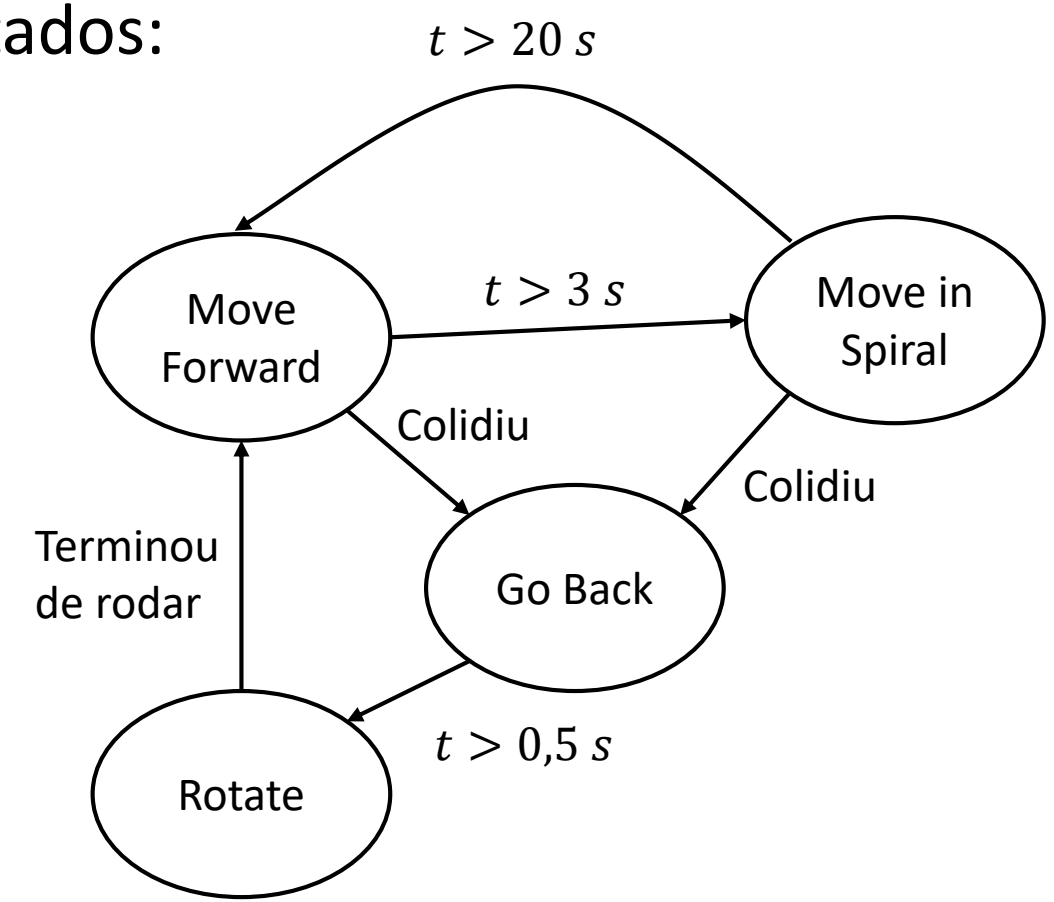
- Máquina de estados e behavior tree.
- Enquanto limpa, alterna entre dois comportamentos:
  - Seguir reto para frente.
  - Limpar em espiral.
- Quando bate numa parede, volta para trás, gira por um ângulo aleatório e então segue reto.
- Equação da espiral: $r(t) = r_0 + b * t$.
- Controle sobre velocidades linear e angular do robô.
- Sensor: *bumper*.
- Tempo de amostragem: 1/60 s

Laboratório 1

- Código base.
- Funções para implementar indicadas com comentário.
- Pode modificar código base, desde que comportamento dos scripts principais permaneça o mesmo.
- Entrega de relatório sucinto com figuras mostrando o robô executando os comportamentos (print screen do rastro).

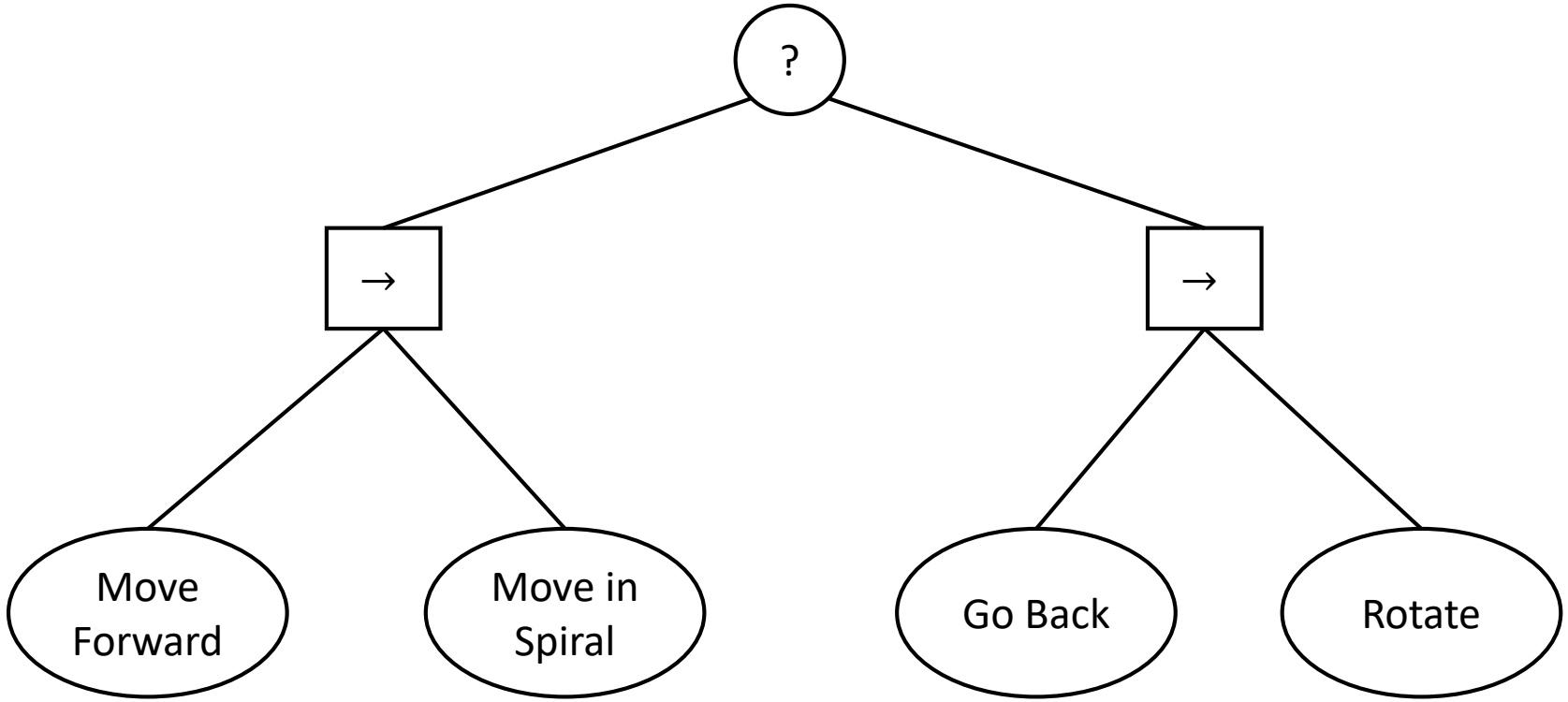
Laboratório 1

- Máquina de estados:



Laboratório 1

- Behavior tree:



Laboratório 1

- Demonstração.