**Instituto Tecnológico de Aeronáutica - ITA**
**Inteligência Artificial para Robótica Móvel - CT213**
**Aluno**: Tafnes Silva Barbosa

**Relatório do Laboratório 1 - Máquina de Estados Finita e *Behavior Tree***

## 1 Breve Explicação em Alto Nível da Implementação

Com o código praticamente implementado, neste laboratório foram completadas algumas funções concernentes à inteligência artificial do robô Roomba usando dois métodos: Máquina de Estados Finita e *Behavior Tree*.

### 1.1 Máquina de Estados Finita

A Figura 1 mostra um diagrama de blocos representando a máquina de estados finita utilizada neste projeto.

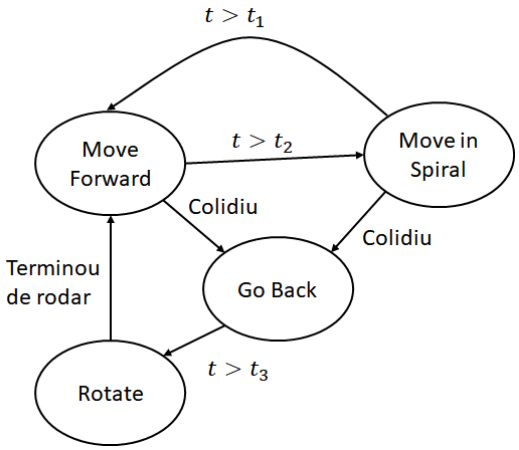


Figura 1: Máquina de estados finita do robô Roomba.

Somente o arquivo `state_machine.py` foi alterado, no qual as classes `MoveForwardState`, `MoveInSpiralState`, `GoBackState` e `RotateState` foram alteradas para realizar a operação esperada.

Para contabilizar o tempo, foi criada uma variável cotadora de número de chamadas do `behavior.update(agent)` chamada `number_calls` para cada classe de estado. A cada chamada do método `execute` de um estado, a respectiva variável é incrementada de 1.

#### 1.1.1 `class MoveForwardState(State)`

Como o código principal começa com este estado, a variável contadora de chamadas foi inicializada em 0 no método `__init__` deste estado.

O método `check_transition(agent, state_machine)` contém a verificação do tempo de execução do estado e da colisão com a parede. A verificação do tempo é realizada comparando o número de chamadas do respectivo estado com o valor obtido pela multiplicação da frequência de amostragem pelo tempo definido na rotina `constants.py`: se é maior ou igual, muda para o estado `MoveInSpiralState()`. A verificação de colisão é realizada através do método do agente `get_bumper_state()`: se há colisão, muda para o estado `GoBackState()`.

O método `execute(agent)` incrementa a variável contadora de chamadas e configura o robô para andar em linha reta setando a velocidade linear para o valor `FORWARD_SPEED` e a velocidade angular para 0, através do método `set_velocity(linear_velocity, angular_velocity)` do agente.

### 1.1.2 `class MoveInSpiralState(State)`

De forma semelhante à anterior, a variável contadora de chamadas foi inicializada em 0 no método `__init__` deste estado e uma variável chamada `spiral_radius`, representando o raio da espiral num dado momento, foi inicializada pelo valor constante `INITIAL_RADIUS_SPIRAL`.

O método `check_transition(agent, state_machine)` contém a verificação do tempo de execução do estado e da colisão com a parede. A verificação do tempo é realizada de forma semelhante ao `MoveForwardState()`: se é maior ou igual, muda para o estado `MoveForwardState()`. A verificação de colisão é realizada da mesma forma do `MoveForwardState()`.

O método `execute(agent)` incrementa a variável contadora de chamadas; incrementa o raio da espiral através da multiplicação do fator de crescimento `SPIRAL_FACTOR` pelo `SAMPLE_TIME`; e configura o robô para andar em espiral setando a velocidade linear para o valor `FORWARD_SPEED` e a velocidade angular para `FORWARD_SPEED / spiral_radius` (velocidade angular em curva de raio `spiral_radius` considerando movimento circular uniforme), através do método `set_velocity(linear_velocity, angular_velocity)` do agente.

### 1.1.3 `class GoBackState(State)`

Esta classe tem comportamento semelhando à `MoveForwardState()`, com a diferença de usar a velocidade `BACKWARD_SPEED` no método `execute(agent)`, comparar com o tempo definido para andar para trás e mudar para o estado `RotateState()`. Esta classe não verifica colisão.

### 1.1.4 `class RotateState(State)`

Além de inicializar o contador de chamadas, o método `__init__` desta classe também escolhe um ângulo aleatória e uniformemente no intervalo $[-\pi, \pi)$, o qual será o ângulo de rotação em torno do eixo do robô.

O método `check_transition(agent, state_machine)` verifica se o robô já rotacionou todo o ângulo escolhido inicialmente e ao término muda para o estado `MoveForwardState()`.

Além de incrementar o contador de chamadas, o método `execute(agent)` seta a velocidade angular do robô para que ele rotacione ao redor do seu eixo, com velocidade positiva se o ângulo escolhido é positivo e vice-versa.

## 1.2 *Behavior Tree*

A Figura 2 mostra um diagrama de blocos representando a*behavior tree* utilizada neste projeto.

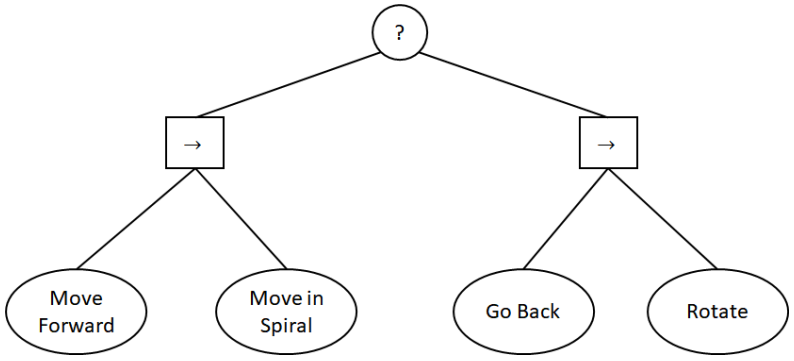


Figura 2: *Behavior tree* de comportamento do robô Roomba.

Somente o arquivo `behavior_tree.py` foi alterado, no qual as classes `RoombaBehaviorTree`, `MoveForwardNode`, `MoveInSpiralNode`, `GoBackNode` e `RotateNode` foram alteradas para realizar a operação esperada.

A contagem de chamadas foi efetuada de forma semelhante à máquina de estados finita, mas, neste caso, toda vez que a raiz da árvore era chamada e em cada nó da árvore.

O que foi feito nos métodos `__init__` dos estados da máquina de estados finita foi feito nos métodos `enter(agent)` das folhas da árvore. Nenhum código foi escrito nos métodos `__init__` das classes das folhas da árvore.

O método `execute(agent)` das folhas `GoBackNode` e `RotateNode` não retornam `FAILURE` dado que não verificam a colisão com as paredes. Por outro lado, o das outras duas folhas retornam `FAILURE` quando há colisão.

O funcionamento macro das folhas é semelhante aos estados da máquina de estados finita. A diferença é que, enquanto o tempo de execução não é alcançado, o método `execute(agent)` das folhas vão retornar `RUNNING`. E quando o tempo de execução finaliza, ele vai retornar `SUCCESS`.

### 1.2.1 `class RoombaBehaviorTree(BehaviorTree)`

O método `__init__` desta classe foi implementada de forma a criar a árvore de comportamentos para o robô Roomba. A raiz da árvore é um nó do tipo *Selector*. Depois se atribui um filho a esta raiz, o qual é do tipo *Sequence*. A este filho se atribui como filhos as folhas `MoveForwardNode` e `MoveInSpiralNode`, nesta ordem.

Depois se adiciona outro filho do tipo *Sequence* à raiz da árvore. A este filho se atribui como filhos as folhas `GoBackNode` e `RotateNode`, nesta ordem.

## 2 Figuras Comprovando Funcionamento do Código

### 2.1 Máquina de Estados Finita

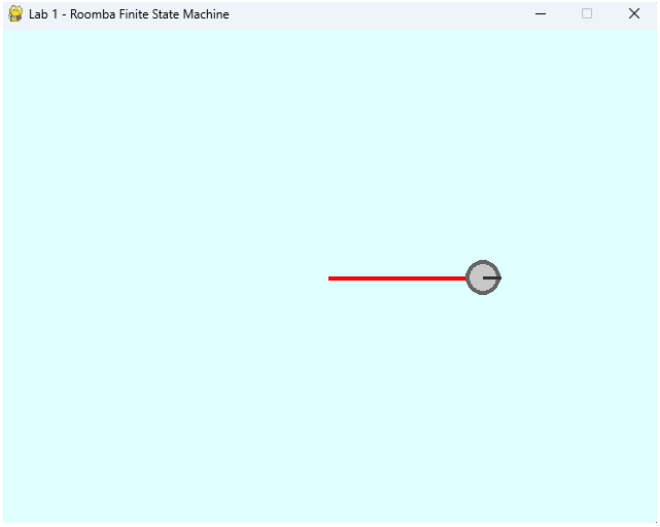


Figura 3: Máquina de estados finita: movendo em frente.

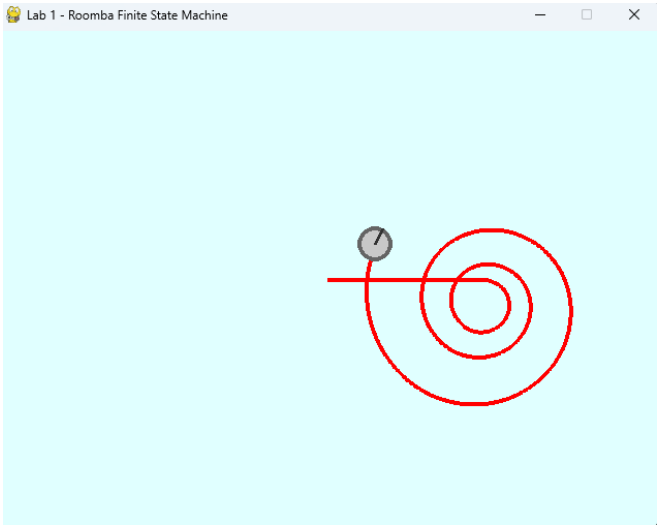


Figura 4: Máquina de estados finita: movendo em espiral.

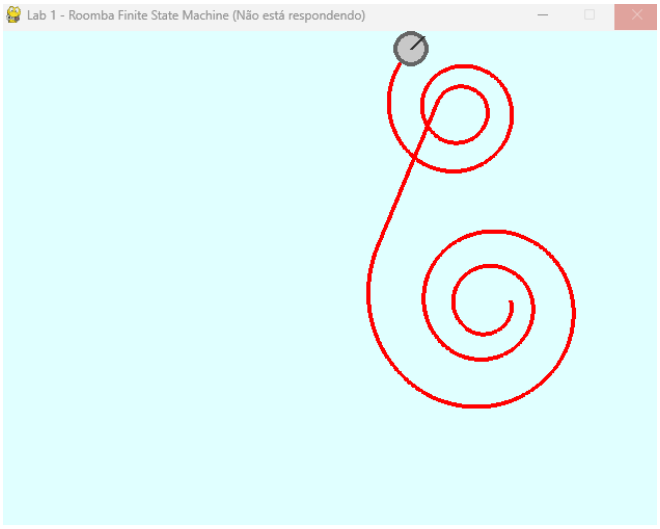


Figura 5: Máquina de estados finita: colidindo com a parede.

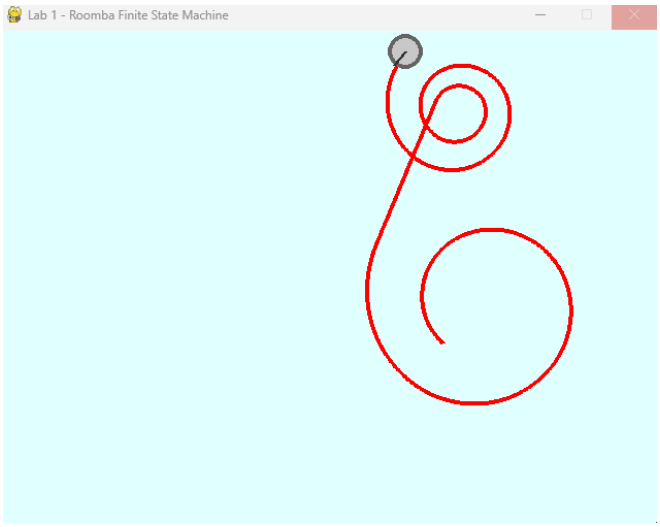


Figura 6: Máquina de estados finita: rotacionando depois de vir para trás.

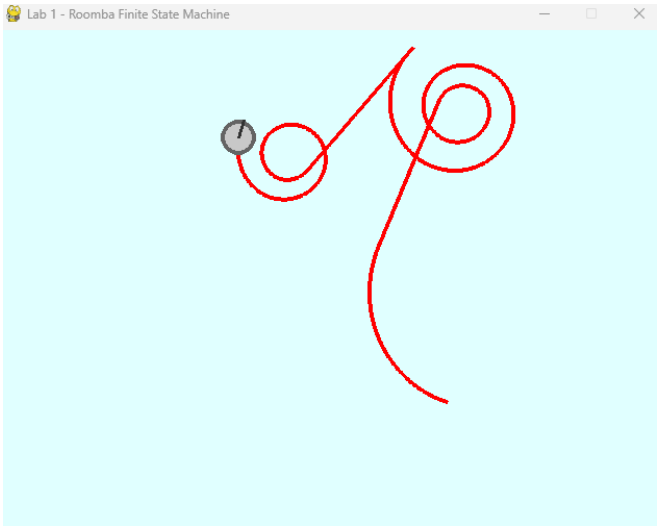


Figura 7: Máquina de estados finita: continuando depois de rotacionar.

## 2.2 *Behavior Tree*

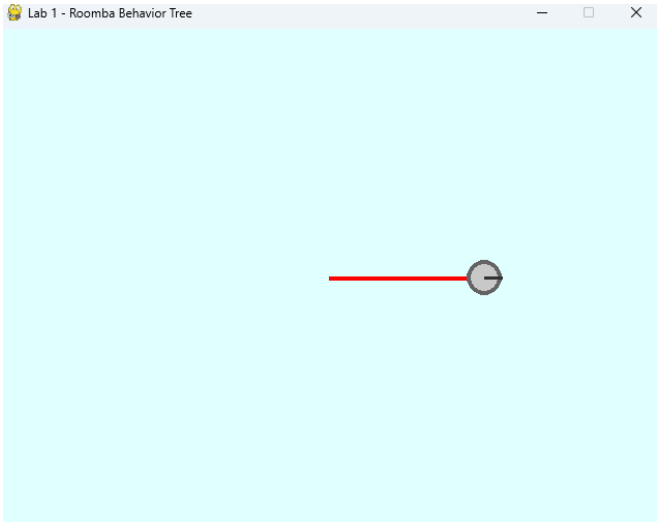


Figura 8: Máquina de estados finita: movendo em frente.

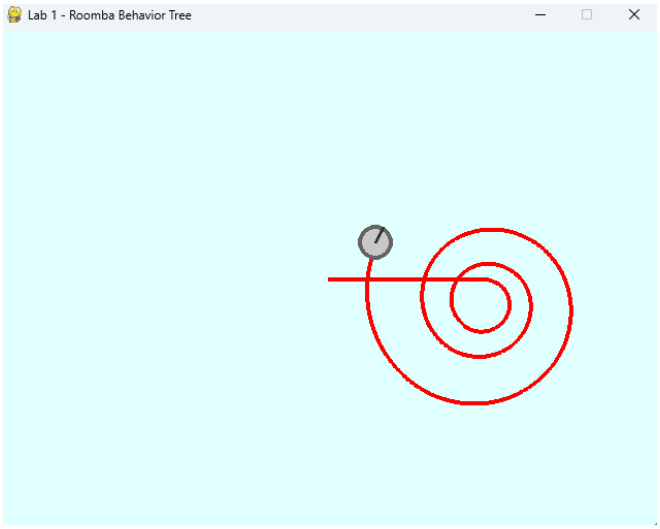


Figura 9: Máquina de estados finita: movendo em espiral.

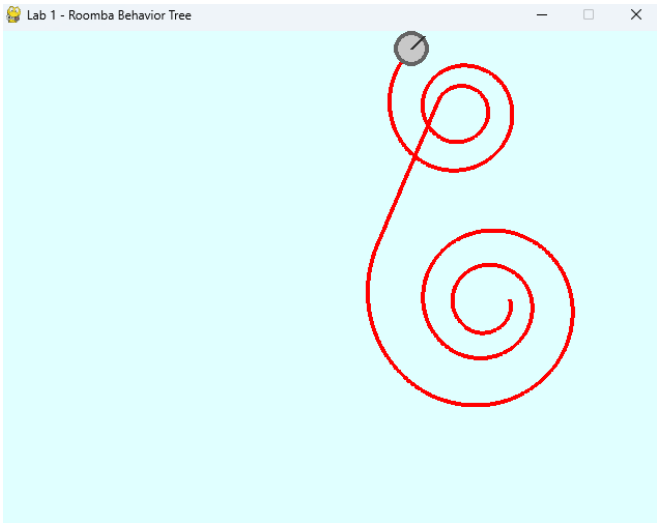


Figura 10: Máquina de estados finita: colidindo com a parede.

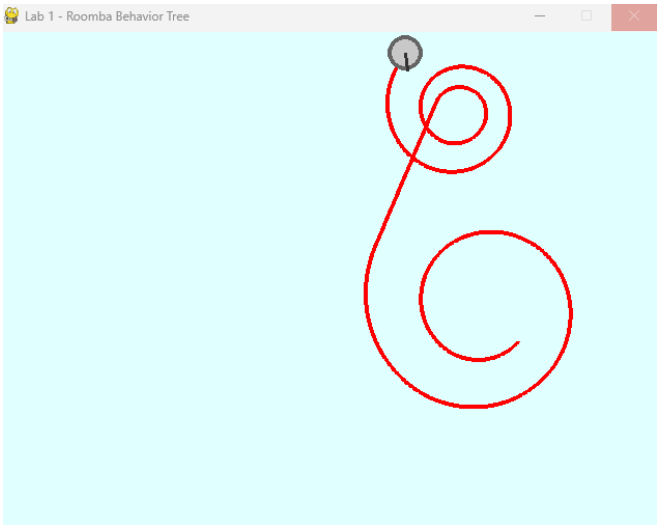


Figura 11: Máquina de estados finita: rotacionando depois de vir para trás.

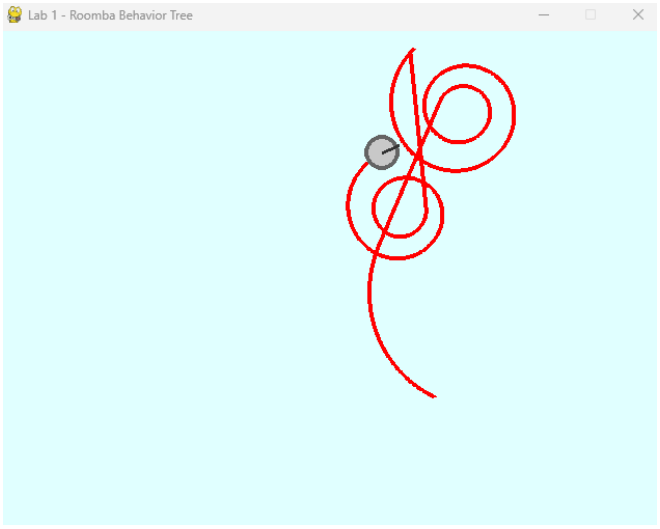


Figura 12: Máquina de estados finita: continuando depois de rotacionar.