Relatório do Lab 1 de CT-213

Trabalho 1 – Arquitetura do Agente

Roomba com *Behavior Tree* e *State Machine*

Aluno:

Bruno Costa Alves Freire

Turma:

T 22.4

Professor:

Marcos Ricardo Omena de Albuquerque Máximo

Data:

06/03/2019

Instituto Tecnológico de Aeronáutica – ITA
Departamento de Computação

1. Implementação da Máquina de Estados

A implementação da arquitetura do agente Roomba em Máquina de Estados foi feita seguindo o modelo fornecido no roteiro do Lab, consistindo de quatro estados: “*MoveForward”*, “*MoveInSpiral*”, “*GoBack*” e “*Rotate*”.

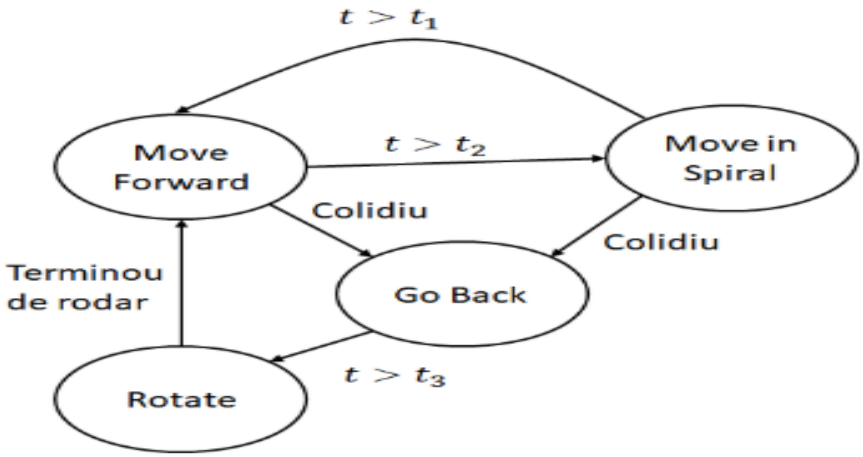

Figura 1: Diagrama da Máquina de Estados utilizada

Em cada estado, foi criada uma variável de controle de tempo no método construtor, chamada `self.running_time`. O objetivo dessa variável é controlar quanto tempo se passou desde que o estado começou a ser executado, e ela é atualizada a cada chamada do método `execute`. Dessa forma, multiplicando essa variável pela constante `SAMPLE_TIME`, obtém-se o tempo de execução do estado. Essa informação é utilizada no método `check_transition` para determinar a transição “natural” dos estados, isto é, caso não haja colisão. No mesmo método, é verificado o estado do *bumper* do agente para detectar colisões, quando nos estados *MoveForward* e *MoveInSpiral*. A colisão nesses estados determina a transição para o estado *GoBack*, e neste, a transição é feita para *Rotate* após decorrido o tempo determinado para a execução deste estado.

No estado *Rotate*, é criado uma variável `self.theta`, que irá receber um ângulo aleatório para o robô girar e seguir limpando. Optou-se por utilizar uma distribuição normal (gaussiana) para escolher um ângulo próximo de 180°, visando reduzir as chances de uma nova colisão em breve. Para determinar a transição nesse estado, é checado se o tempo decorrido em execução vezes a velocidade angular é maior que o ângulo sorteado.

No mais, um detalhe precisa ser mencionado no estado *MoveInSpiral*. Para produzir o movimento no caminho desejado, foi passada uma velocidade linear constante para o robô, e uma velocidade angular variável, em função do raio da espiral (que por sua vez, varia com o tempo de execução). A velocidade transmitida é dada pela equação:

$$\omega = v \cdot \frac{2b^2 + r^2}{\sqrt[4]{(b^2 + r^2)^5}}$$

Onde ω é a velocidade angular, *v* é a velocidade linear, *b* é o parâmetro da espiral, e *r* é dado como função do tempo de execução *t* conforme a equação:

$$r = r_0 + b \cdot t$$

O motivo da escolha dessa fórmula foi pelo fato da mesma ter apresentado comportamento mais próximo de um espiral arquimediana.

A seguir, as figuras 1 e 2 mostram o comportamento do agente Roomba após a implementação da Máquina de Estados.

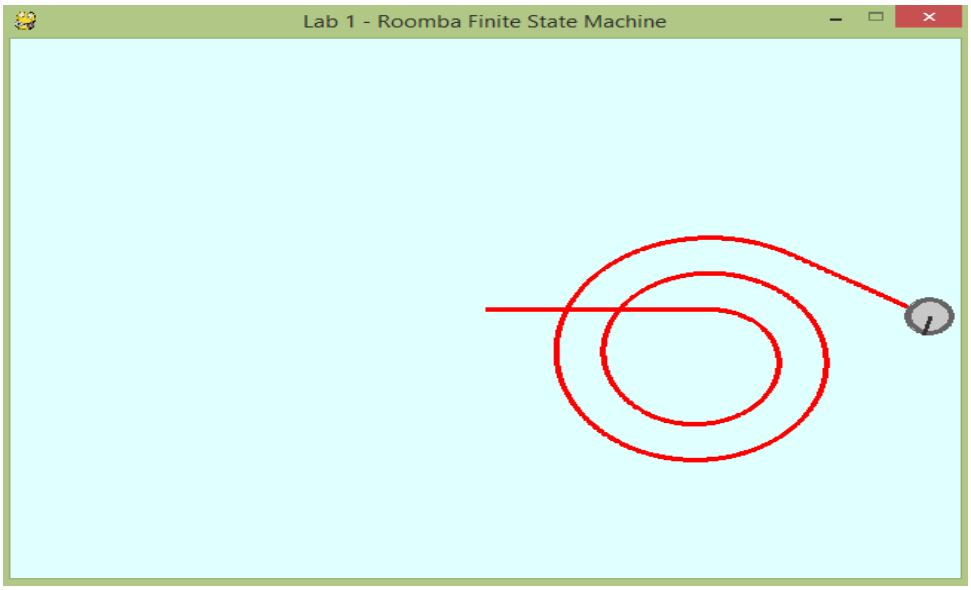

Figura 2: Roomba com FSM exibindo espiral

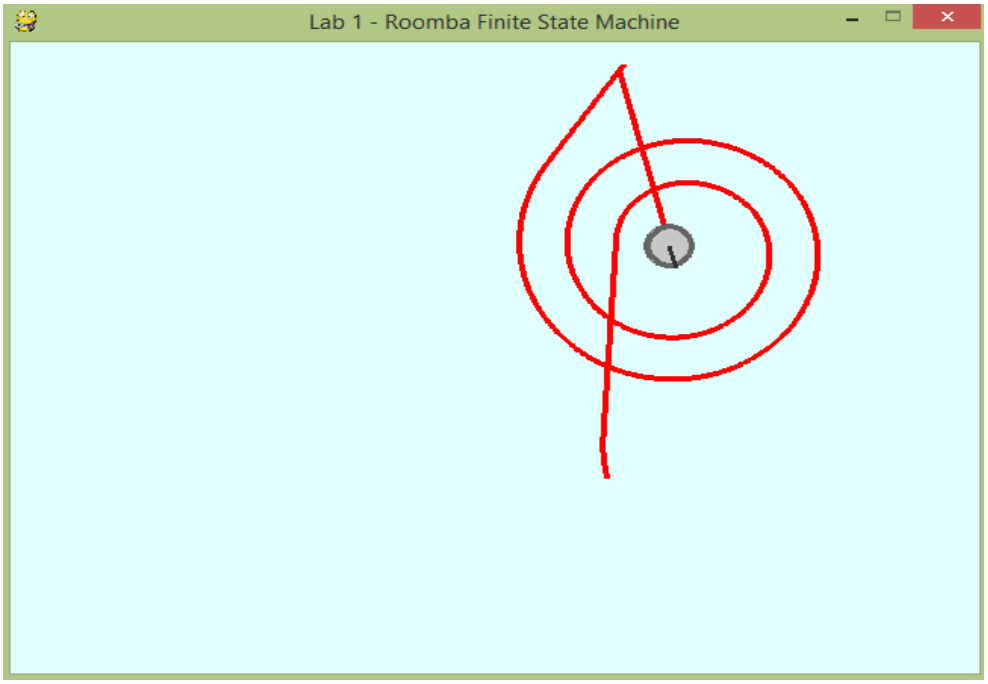

Figura 3: Roomba com FSM após ter colidido e rotacionado

2. Implementação da *Behavior Tree*

Para a implementação da *Behavior Tree*, foi seguido o modelo fornecido no roteiro do Lab, que consiste de uma árvore onde a raiz é um nó *Selector*, com dois filhos *Sequence*, em que o primeiro é pai dos nós-folha *MoveForward* e *MoveInSpiral*, e o segundo é pai dos comportamentos *GoBack* e *Rotate*.

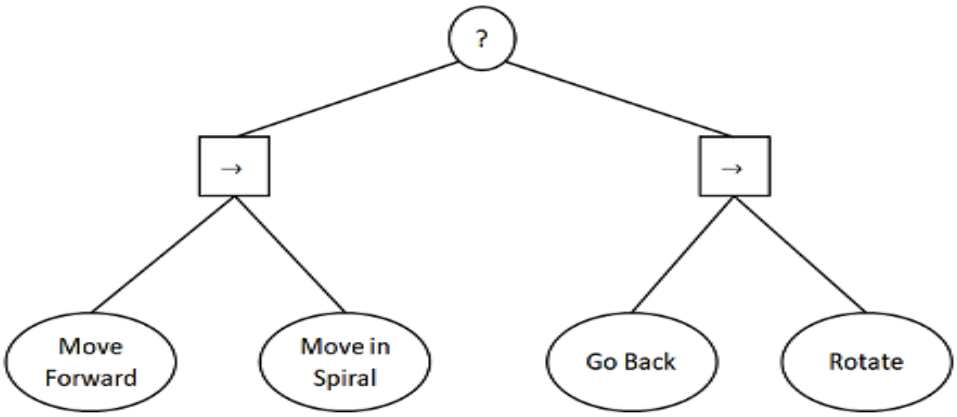

Figura 4: Diagrama da Árvore de Comportamentos usada

Primeiramente, foi construída a árvore no método construtor da classe `RoombaBehaviorTree`, conforme o diagrama. Para implementar o funcionamento de cada *behavior*, lança-se mão novamente de uma variável para monitorar o tempo de execução, `self.running_time`. No método `execute` de cada nó implementado, verifica-se se o tempo de execução do comportamento é superior ao tempo estipulado no projeto, e caso o seja, é retornado o status de sucesso. Caso não tenha transcorrido o tempo estipulado para a execução, retorna-se o status `RUNNING`.

Para os comportamentos *MoveForward* e *MoveInSpiral*, é verificado também se não houve colisão. Caso haja, é retornado status de falha. A implementação cinemática destes comportamentos é igual à da máquina de estados.

Um detalhe importante que deve ser salientado é a inicialização da variável `self.running_time`, que deve ser feita sempre no método `enter` do comportamento, para evitar contagens acumuladas de tempo.

A seguir, as figuras 5, 6 e 7 mostram o resultado da implementação no comportamento do agente.

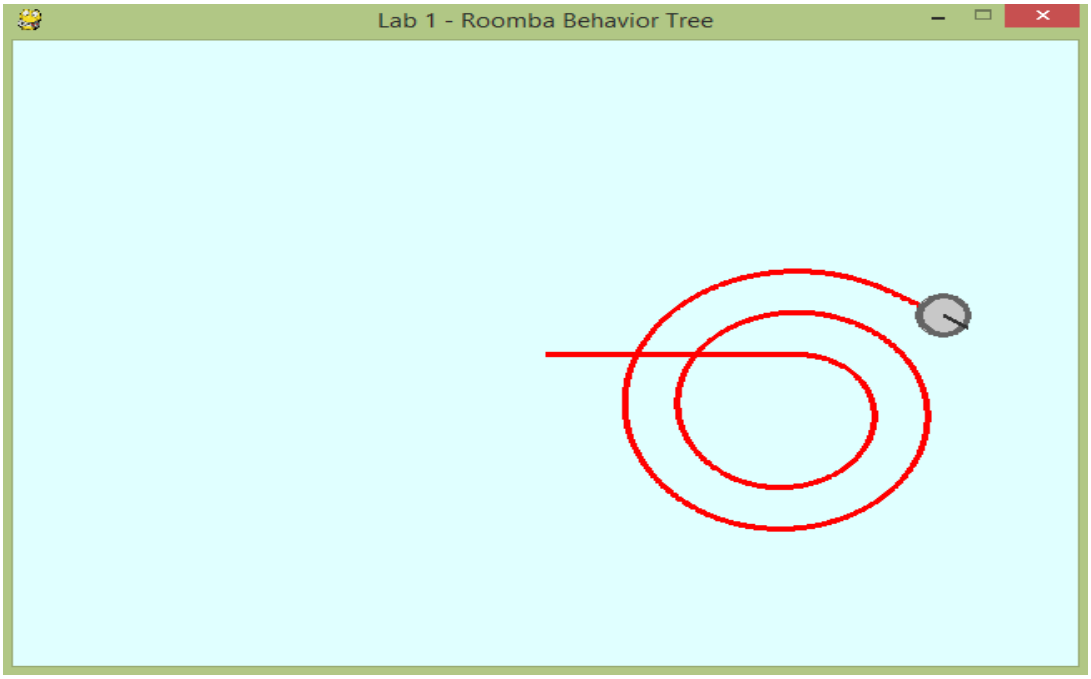

Figura 5: Roomba com BT exibindo uma espiral

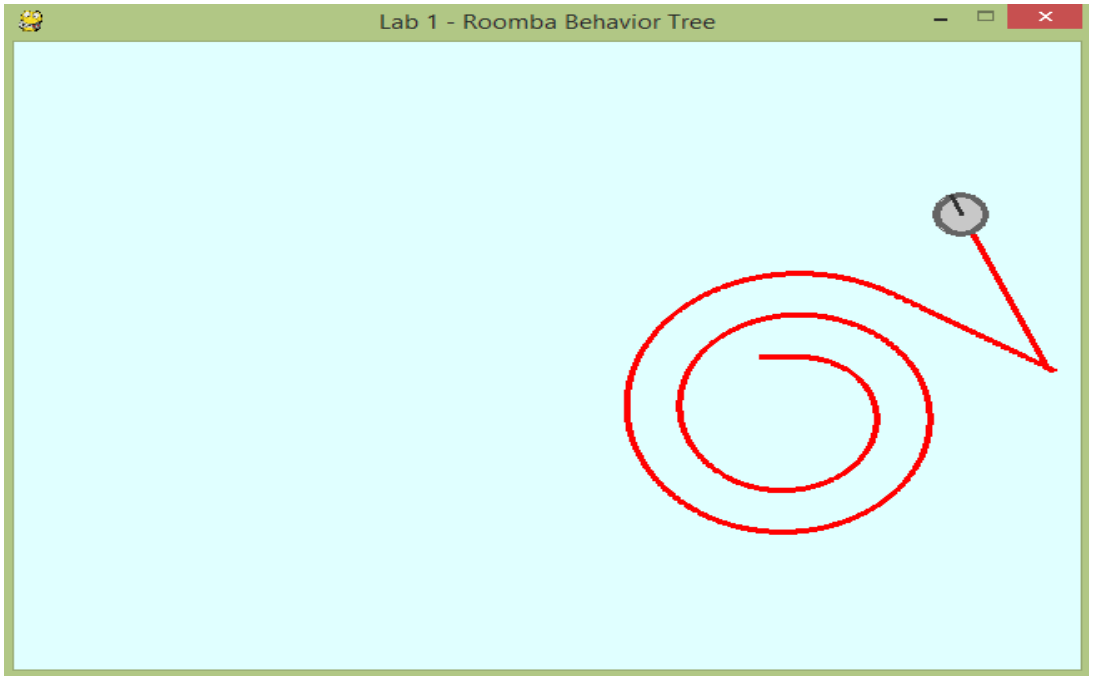

Figura 6: Roomba com BT após colidir e rotacionar

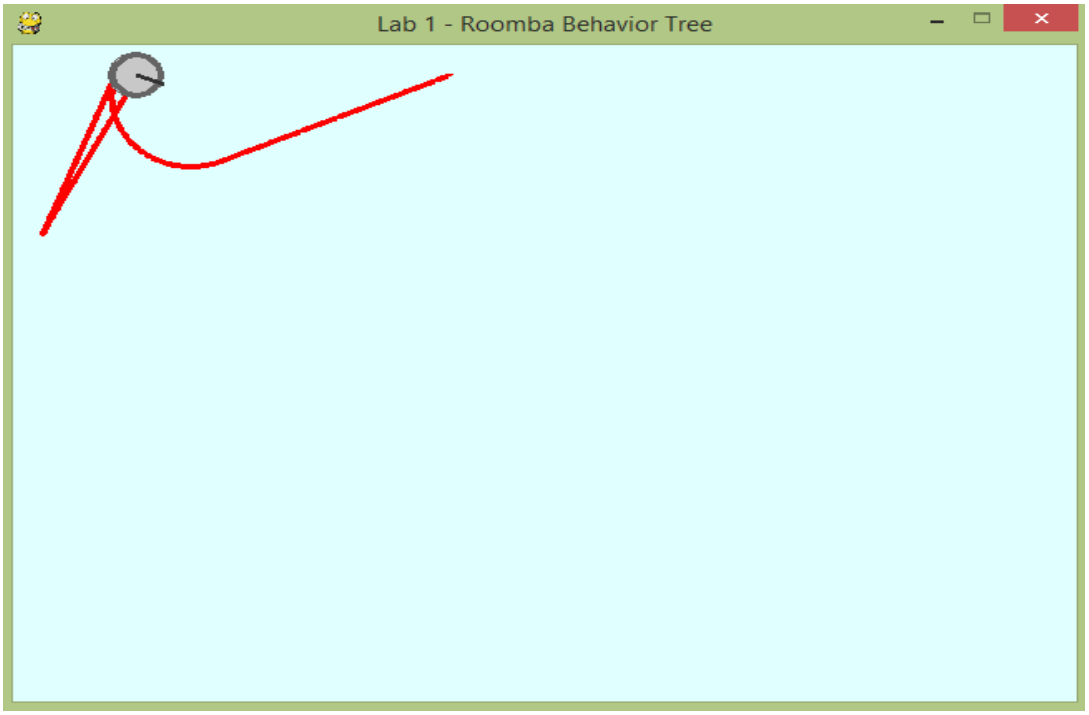

Figura 7: Roomba com BT no exato instante em que executa a rotação