**Instituto Tecnológico de Aeronáutica – ITA**

**Inteligência Artificial para Robótica Móvel – CT-213**

**Professor: Marcos Ricardo Omena de Albuquerque Maximo**

**Laboratório 1 – Máquina de Estados Finita e Behavior Tree**

1. **Introdução**

Nesse laboratório, seu objetivo é implementar o comportamento de um robô Roomba. O Roomba é um robô de limpeza desenvolvido pela empresa iRobot. Seu uso é para limpeza de chão de lugares fechados. No caso, será implementado um comportamento idealizado e simplificado do Roomba em um simulador. A Figura 1 mostra um Roomba.



**Figura 1:** um gato vestido de tubarão pegando carona em um Roomba.

1. **Descrição do Comportamento do Agente**

Em alto nível, o comportamento do Roomba simulado é o seguinte:

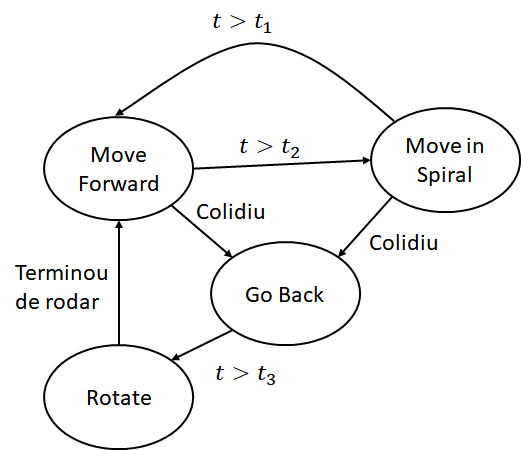
* Enquanto limpa (funcionamento normal), o robô alterna entre dois comportamentos:
  + Seguir reto para frente.
  + Limpar em espiral.
* Quando colide com uma parede, o robô executa os seguintes movimentos em sequência:
  + Volta para trás um pouco.
  + Gira de um ângulo aleatório.
  + Volta a limpar, começando do comportamento em que “segue para frente”.

Para detectar se colidiu, o Roomba é dotado de um “*bumper*”. Esse *bumper* fornece uma informação do tipo booleana (verdadeiro ou falso).

A espiral executada pelo robô é tal que seu raio segue a seguinte lei: , em que é o raio inicial, é um fator que determina o quão rápido o raio da espiral cresce e é o tempo desde que o comportamento de espiral começou.

* 1. **Máquina de Estados Finita**

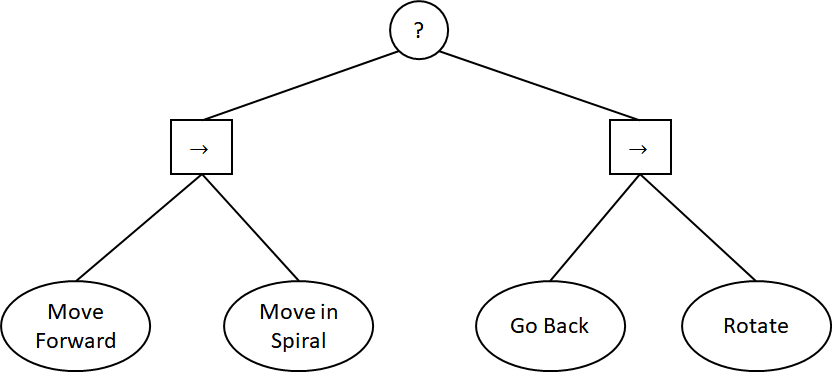
A Figura 2 mostra uma máquina de estados finita para modelar o comportamento do Roomba descrito acima. Os tempos , e são parâmetros do projetista, mas será dado uma sugestão para valores de cada um deles mais adiante (obtidos por tentativa e erro).



**Figura 2:** máquina de estados finita do comportamento do Roomba.

* 1. **Behavior Tree**

A Figura 3 mostra uma *behavior tree* para modelar o comportamento do Roomba.

****

**Figura 3:** *behavior tree* do comportamento do Roomba.

1. **Código Base**

Junto com esse roteiro, foi entregue um código base. Esse código base contém boa parte da implementação do laboratório. A ideia é que você preencha apenas as partes relativas à implementação de máquina de estados finita e *behavior tree*. O código base entregue nesse laboratório consiste dos seguintes arquivos:

* roomba.py: classe que simula um robô Roomba.
* simulation.py: classe responsável pela simulação, tanto pela lógica quanto pelo gráfico.
* constants.py: constants usadas no código, inclusive constantes relativas ao comportamento do robô.
* utils.py: classes utilitárias de geometria.
* state\_machine.py: implementação do comportamento do robô usando máquina de estados finita.
* behavior\_tree.py: implementação do comportamento do robô usando *behavior tree*.
* state\_machine\_test.py: script “main” para testar a implementação usando máquina de estados finita.
* behavior\_tree\_test.py: script “main” para testar a implementação usando *behavior tree*.

Os arquivos que você precisará completar a implementação são: state\_machine.py e behavior\_tree.py. Perceba que todos os métodos que requerem implementação da sua parte possuem o marcador “Todo”. Nesses métodos, fique à vontade para alterar o que desejar. Os arquivos “executáveis” são: state\_machine\_test.py e o behavior\_tree\_test.py. É interessante que você use as constantes definidas em constants.py (*Sample Time Parameters* e *Behavior Parameters*). Finalmente, você precisará usar os métodos set\_velocity() e get\_bumper\_state() da classe Roomba.

1. **Tarefas**

O código base já possui estruturas de código para implementação da máquina de estados finita e da *behavior tree*. Algumas dicas gerais para facilitar sua implementação:

* Para fazer o Roomba se mover em espiral, sugere-se calcular o raio da espiral no instante de tempo atual. Então, dada uma velocidade linear pré-definida e usando conhecimentos básicos de cinemática (movimento circular uniforme), calcular a velocidade angular para que o robô execute o raio calculado.
* Para medir tempo de simulação, recomenda-se contar quantas vezes a IA do Roomba foi executada (através da chamada dos métodos de execução da máquina de estados ou da *behavior tree*) e então usar o parâmetro SAMPLE\_TIME para transformar esse tempo discreto em segundos.
* Para mudar a velocidade do Roomba, use o método set\_velocity(linear\_speed, angular\_speed).
* Para verificar o estado do bumper, use o método get\_bumper\_state(), que retorna um booleano.
* A atualização da simulação já está implementada para você, você deve apenas se preocupar em comandar velocidades e ler o *bumper* do robô (você está implementando apenas a IA).
* Parâmetros adequados relativos ao comportamento do robô foram obtidos pelo professor por tentativa e erro. Sugestões de valores desses parâmetros encontram-se no arquivo constants.py. Por facilidade, os parâmetros relevantes para implementação do laboratório estão indicados na Tabela 1. O ângulo aleatório usado no comportamento Rotate foi amostrado uniformemente no intervalo , porém outros valores são perfeitamente aceitáveis.
* Não é necessário se preocupar em seu Roomba se comportar exatamente igual ao do professor, desde que tenha um comportamento adequado e semelhante ao descrito no roteiro.

**Apresente no seu relatório capturas da tela do simulador mostrando o funcionamento correto dos comportamentos implementados**. Para isso, aproveite-se do rastro deixado pelo robô durante seu movimento.

| **Parâmetro** | **Significado** |
| --- | --- |
| SAMPLE\_TIME | Tempo de amostragem da IA do Roomba, i.e. o agente toma uma nova decisão a cada “SAMPLE\_TIME” de tempo. |
| MOVE\_FORWARD\_TIME | Tempo que o comportamento de se mover para frente deve executar antes de mudar para o movimento em espiral. |
| GO\_BACK\_TIME | Tempo que o comportamento de ir para trás após uma colisão deve executar. |
| FORWARD\_SPEED | Velocidade linear quando o robô está se movendo para frente ou em espiral. |
| BACKWARD\_SPEED | Velocidade linear quando o robô está se movendo para trás. |
| INITIAL\_RADIUS\_SPIRAL | Raio inicial da espiral. |
| SPIRAL\_FACTOR | Fator de crescimento da espiral. |
| ANGULAR\_SPEED | Velocidade angular quando o robô está no comportamento de girar por um ângulo aleatório. |

**Tabela 1:** parâmetros de constants.py relevantes para implementação do laboratório.

* 1. **Máquina de Estados Finita**

Em state\_machine.py, implemente os métodos check\_transition() e execute() das classes MoveForwardState, MoveInSpiralState, GoBackState e RotateState. Dicas:

* Os nomes das classes são relativos à máquina de estados da Figura 2.
* Os métodos tem o seguinte significado:
  + check\_transition(): usado para verificar condições e executar as transições entre estados.
  + execute(): executa a lógica do estado.
* Os parâmetros dos métodos tem os seguintes significados:
  + agent: instância do robô Roomba em que a IA está sendo executada.
  + state\_machine: instância da máquina de estados em que o estado está sendo executado.
* Para testar sua implementação, basta executar o script state\_machine\_test.py.
  1. **Behavior Tree**

Em behavior\_tree.py, implemente os métodos enter() e execute() das classes MoveForwardNode, MoveInSpiralNode, GoBackNode e RotateNode. Além disso, será necessário construir a *behavior tree* no método \_\_init\_\_() da classe RoombaBehaviorTree. Dicas:

* Os nomes das classes são relativos à *behavior tree* da Figura 3.
* Os métodos tem o seguinte significado:
  + enter(): executado uma vez quando se “entra” nessa tarefa, i.e. assim que a tarefa é reiniciada.
  + execute(): executado a cada iteração em que a tarefa é executada.
* Os parâmetros dos métodos tem os seguintes significados:
  + agent: instância do robô Roomba em que a IA está sendo executada.
* Para testar sua implementação, basta executar o script behavior\_tree\_test.py.
* Para construir a árvore, você precisará instanciar cada nó da árvore usando o respectivo construtor. Perceba que nós *sequence* e *selector* são representados pelas classes SequenceNode e SelectorNode. Para adicionar um filho a um nó composto, use o método add\_child().

1. **Entrega**

A entrega consiste do código e de um relatório, submetida através do Google Classroom. Modificações nos arquivos do código base são permitidas, desde que o nome e a interface dos scripts “main” não sejam alterados. A princípio, não há limitação de número de páginas para o relatório, mas pede-se que seja **sucinto**. O relatório deve conter:

* **Breve descrição em alto nível da sua implementação.**
* **Figuras que comprovem o funcionamento do seu código.**

Por limitações do Google Classroom (e por motivo de facilitar a automatização da correção), entregue seu laboratório com todos os arquivos num único arquivo **.zip** (**não** utilize outras tecnologias de compactação de arquivos) com o seguinte padrão de nome: “**<login\_email\_google\_education>\_labX.zip**”. Por exemplo, no meu caso, meu login Google Education é **marcos.maximo**, logo eu entregaria o lab 1 como “**marcos.maximo\_lab1.zip**”. **Não** crie subpastas para os arquivos da sua entrega. Os relatórios devem ser entregues em formato **.pdf**.