Atividade 01 - IC 2024.1

Aluno: Davi de Paula Coelho Raia dos Santos
Curso Ciência da Computação
NOTA: REDAÇÃO JÁ ENVIADA

1. Introdução

Foi utilizado a linguagem Python para desenvolver os agentes e o ambiente.
A arquitetura usada no projeto foi a seguinte:

Foi criado 2 Classes de Alto Nível (Agente e Ambiente):

- a classe Agente é uma classe que contém as funções que são compartilhadas entre os agentes. (agente.py)
- a classe Ambiente é uma classe que contém as leituras do ambiente junto com as variáveis que os definem. (ambiente.py)

A Classe Agente tem 2 classes filhas a classe AgenteReativo1 e a classe AgenteModelo1:

- A classe AgenteReativo1 contém um agente reativo baseado no agente reativo contido no projeto java AI indicado na atividade. Possue chances diferentes de executar ações. (Arquivo reativo_agente.py)
- A classe AgenteModelo1 contém um agente baseado em modelos, foi definido um modelo de ações que o agente deve seguir. (Arquivo b_modelo_agente.py)

2. Decisões

2.1 Classe Agente

Para auxiliar a implementação foi criado uma classe básica de agente que pode ser extendida para alcançar o objeto, essa classe é a Agente (agente.py) que contém os Métodos abstratos

```
def definir_acao():
    'responsável por definir a ação do agente e executá-la'

def criar_modelo():
    'Utilizada apenas por agentes baseados em modelos, para conter a função responsável por
```

Os métodos pré implementadas da classe Agente são as seguintes:

```
def salvar_historico(self, acao, parametro):
    'responsável por salvar o histórico para coletar resultados'
    return  [(posiçãox, posiçãoy), sentido, pontuacao_simples, pontuação_composta, acao, par

def perceber(self):
    'Função importante!!!'
```

```
    'Lê o estado atual da celula em que o agente está contido'
    'Chama um método da Classe Ambiente'

def limpar(self):
    'Alterar o estado do ambiente'
    'Chama um método da Classe Ambiente'

def alterar_sentido(self, direcao):
    'Alterar o sentido, Norte, Sul, Leste, Oeste'

def andar(self):
    'Alterar posição do agente'
    'Chama um método da Classe Ambiente'
```

E a classe possui esses Atributos para guardar informações.

```
def __init__(self, ambiente: Ambiente):
        # Sensor atual [É Home?, Sujo ou Limpo?, Obstáculo?]
        self.vsensor = [True, False, False]
        self.pontuacao = 0  # Guarda o histórico de pontuação
        self.pontuacao_v2 = 0  # Guarda o histórico de pontuação
        self.histmov = []  # Guarda o histórico de movimentos
        self.ambiente = ambiente  # Inicia o Ambiente
        self.head_pointer = "s"  # Inicia apontando para o Sul (Baixo)
        self.counter = 0  # Usado no agente baseado em modelo para contar ações
```

2.2 Classe Ambiente

Uma decisão importante foi que cada Instancia de uma classe Agente precisa ter um ambiente associado a ela. Isso foi feito pois a classe Ambiente (Que será tratada logo abaixo) guarda a informação do tamanho do ambiente, posição do agente no ambiente, e se a célula atual está suja ou limpa.
O Ambiente armazena os dados se sujeira por meio de uma Matriz.

Os atributos da classe são os seguintes:

```
def __init__(self, x, y, caminho_ambiente):
        self.x = x # tamanho x
        self.y = y # tamanho y
        self.caminho_ambiente = caminho_ambiente # Path do arquivo
        self.local = [] # ambiente (Dinamico)
        self.__agente_x = 0 # Posição agente
        self.__agente_y = 0 # Posição agente
        self.carregar_ambiente() # Chamada para ler o arquivo .txt que contem o ambiente.
```

Essa classe não possue métodos abstratos. Porém, contém os seguintes Métodos.

```
def getAgenteX(self):
        return self.__agente_x
```

```
def getAgenteY(self):
    return self.__agente_y

def carregar_ambiente(self):
    """Carrega o ambiente de um arquivo para a variavel local"""

def atualizar_ambiente(self):
    '''Usado na função limpar da classe agente, atualiza o ambiente'''

def recuperar_informacao_local(self):
    '''Recupera a informação de HOME e SUJEIRA da celula atual do agente'''

# As seguintes são utilizas na função andar
# Motivo: a classe Ambiente armazena a posição do agente no ambiente
def mover_norte(self):
def mover_sul(self):
def mover_leste(self):
def mover_oeste(self):
```

2.3 Classe AgenteReativo1

Essa é a Classe em que será executada nos testes, ela herda atributos e métodos da classe Agente.
O método:

```
def definir_acao(self):
```

É definido (feito o Override) dentro da classe AgenteReativo. Ela não implementa o método criar_modelo, uma vez que o agente Reativo não é definido por modelos.
Foi utilizado porcentagens para definir as ações:

- Sempre que o ambiente estiver sujo limpe
- Se encontrar um obstáculo:
  - 45% de virar para a Esquerda
  - 45% de virar para a Direta
  - 10% de chance de Se estiver em 'HOME' cancelar a atividade
- 20% de chance de virar para a Esquerda
- 20% de chance de virar para a Direita
- 50% de chance de andar
- 10% de chance de Se estiver em 'HOME' cancelar a atividade

Antes de cada ação dessa o Agente Percebe o ambiente.

2.4 Classe AgenteModelo1

A classe também é utilizada nos testes, ela implementa ambos os métodos herdados. O Método criar_modelo cria um modelo circular de limpeza, esse modelo influencia a forma que o método definir_acao funciona.

O método definir ação:

- Sempre limpa se o ambiente estiver sujo
- Caso esteja limpo ele consulta o modelo:
  - Caso o sensor reconheça um obstáculo ele altera a direção
  - Caso o sensor NÃO reconheça o obstáculo ele anda
- Existe uma chance de 10% de encerrar a operação caso o sensor reconheça 'HOME' e o não seja a primeira iteração (iteração = 0)

o método criar modelo:

- Checa o sensor de obstáculo
- Usa um contador de alterações de direções para girar de forma circular no ambiente

Nota: Verificar os códigos no arquivos: ambiente.py, agente.py, b_modelo_agente.py, reativo_agente.py

3. Como foi realizado os testes?

Foram criados 4 Ambientes pré definidos e editáveis pelo 'testador' (Localizados na pasta ambientes) em formato .txt.
O número 1 indica sujeira e o número 0 indica que está limpo (nos arquivos de ambiente).
Foi criado 4 instâncias de cada um dos 2 Agentes definidos acima, cada instância recebe um ambiente.
Os agentes serão comparados no mesmo ambiente.
Existe um limite de 1000 iterações
2 Métodos de pontuação:

- Simples: Um ponto é adicionado a cada limpeza efetuada
- Complexo: Um ponto é adicionado a cada limpeza efetuada e removido a cada movimento (Andar e altera sentido)

É gerado um relatório de cada decisão efetuada pelo agente (Localizado na pasta results), cada arquivo contém o histórico de:

- Posição
- Sentido
- Pontuação Simples
- Pontuação Complexa
- Ação tomada

- Parametro passado na ação

Com base nesses dados coletados conseguimos verificar os resultados.

4. Resultados

Os arquivos que contém os resultados brutos estão contidos no diretório results.

De forma breve temos:

- Ambiente 1 (Verificar arquivo ambientes/ambiente.txt):
  - Agente Reativo (iterações: 251):
    - Pontuação Simples: 5
    - Pontuação Complexa: -240
  - Agente Modelo (iterações: 20):
    - Pontuação Simples: 5
    - Pontuação Complexa: -9
- Ambiente 2 (Verificar arquivo ambientes/ambiente2.txt):
  - Agente Reativo (iterações: 53):
    - Pontuação Simples: 5
    - Pontuação Complexa: -42
  - Agente Modelo (iterações: 24):
    - Pontuação Simples: 5
    - Pontuação Complexa: -13
- Ambiente 3 (Verificar arquivo ambientes/ambiente3.txt):
  - Agente Reativo (iterações: 34):
    - Pontuação Simples: 5
    - Pontuação Complexa: -23
  - Agente Modelo (iterações: 51):
    - Pontuação Simples: 16
    - Pontuação Complexa: -18
- Ambiente 4 (Verificar arquivo ambientes/ambiente4.txt):
  - Agente Reativo (iterações: 238):
    - Pontuação Simples: 8
    - Pontuação Complexa: -221
  - Agente Modelo (iterações: 25):
    - Pontuação Simples: 6
    - Pontuação Complexa: -12

5. Conclusões

Usando os resultados obtidos é possível observar:

- O modelo implementado no Agente em modelos permite que ele seja deterministico (Irá reproduzir o mesmo resultado se for o mesmo ambiente) ainda que ele se adapte a vários ambientes.
- O agente Reativo tem característica de aleatoriedade em certos momentos portanto não é consistente (Exemplo no Ambiente 3, o agente possui menos iterações que o agente baseado em modelos, o que não ocorre nos outros ambientes).
- Em ambientes pequenos o agente baseado em modelo desempenha melhor
- O agente baseado em modelos tem uma consistência maior, a pontuação dele não tem picos.

Nesse tipo de Ambiente (Mundo Aspirador de Pó) o Agente baseado em modelos é uma opção melhor para economizar recursos, justamente por ser mais consistente, pode ser desenvolvidos Modelos e Padrões de tomadas de decisão que melhorem ainda mais a sua eficiencia, fazendo que ele visite apenas uma vez cada célula.