Documentação do Projeto: Navegação de Robô com Q-Learning

Autor: Gabriel Castelo **Algoritmo:** Q-Learning (Reinforcement Learning) **Ambiente:** Grafo de 6 Estados

1. Definição do Problema

Este projeto simula um ambiente simples onde um agente (como um robô) precisa aprender a navegar de qualquer sala (estado) inicial até uma sala de saída (objetivo).

O Ambiente

O ambiente é modelado como um grafo de **6 estados** (numerados de 0 a 5).

- **Estados 0 a 4:** Salas comuns.
- **Estado 5:** O objetivo (Saída/Destino final).

O robô pode se mover entre salas conectadas. O objetivo do algoritmo é aprender o caminho mais curto e eficiente para chegar ao **Estado 5**, independentemente de onde ele comece.

A Matriz de Recompensas (R)

O ambiente fornece feedback ao agente através de recompensas:

- **-1:** Movimento impossível (não há porta entre as salas).
- **0:** Movimento válido, mas não é o objetivo.
- **100:** Chegou ao objetivo (Estado 5).

2. A Solução: Q-Learning

O projeto utiliza **Q-Learning**, uma técnica de aprendizado por reforço *model-free*. O objetivo é preencher uma tabela (Matriz Q) que diz ao robô: *"Se eu estiver no estado X, quão bom é tomar a ação Y?"*.

A Fórmula de Atualização

A "inteligência" do robô vem da atualização dos valores Q usando a Equação de Bellman modificada:

No código, isso é implementado como:

Q[estado, acao] = Q[estado, acao] + alpha * (

```
    recompensa + gamma * melhor_q_prox - Q[estado, acao]
)
```

Onde:

- `alpha` (0.1): Taxa de aprendizado (o quanto o robô aceita novas informações).
- `gamma` (0.8): Fator de desconto (o quanto o robô valoriza recompensas futuras em relação às imediatas).

3. Estrutura do Código

O script `Q_learning.py` realiza as seguintes etapas:

1. **Inicialização:** Define a matriz de recompensas `R` e inicializa a matriz de conhecimento `Q` com zeros.
2. **Treinamento:**
    - Executa 3.000 episódios.
    - Em cada episódio, o agente explora o ambiente aleatoriamente até encontrar o objetivo.
    - A cada passo, a matriz `Q` é atualizada com base na recompensa recebida e na melhor estimativa futura.
3. **Inferência (Teste):** Após o treino, utiliza a matriz `Q` preenchida para determinar a melhor rota a partir de qualquer estado.

4. Exemplo de Uso e Resultados

Ao executar o programa, ele treina o agente e imprime os resultados do aprendizado.

Matriz Q Aprendida

A matriz Q final conterá valores altos para ações que levam diretamente ao objetivo (Estado 5) e valores decrescentes para estados mais distantes.

Saída do Programa (Melhores Rotas)

O sistema calcula e imprime o melhor caminho passo-a-passo para cada estado possível.

Exemplo de Output no Console:

```
Melhores rotas ate o estado 5:
Estado 0: [0, 4, 5]
Estado 1: [1, 5]
Estado 2: [2, 3, 1, 5]  # Ou [2, 3, 4, 5] dependendo do treino
Estado 3: [3, 4, 5]     # Caminho ótimo via estado 4 (ou 1)
Estado 4: [4, 5]
```

Estado 5: [5]

Interpretação

- Se o robô começar no **Estado 2**, ele aprendeu que deve ir para o **3**, depois para o **1** (ou 4) e finalmente para o **5** para maximizar sua recompensa.
- Se começar no **Estado 1**, ele sabe que pode ir direto para o **5**.

5. Requisitos

- Python 3.x
- NumPy (`pip install numpy`)