Instituto Tecnológico de Aeronáutica - ITA
Inteligência Artificial para Robótica Móvel - CT-213
Aluno: Danilo de Farias Matos

Relatório do Laboratório 11 - Aprendizado por Reforço Livre de Modelo

1. Breve Explicação em Alto Nível da Implementação

Iniciei a implementação apenas adicionando a lógica da função 'epsilon_greedy_action()' para retornar ações seguindo a seguinte lógica: Sortear um valor aleatório (entre 0 e 1). Se o valor aleatório sorteado for menor que o epsilon definido a função retorna uma ação aleatória (fazendo com que o algoritmo ganhe capacidade de explorar), caso contrário ela retorna a ação com maior valor de Q associado(que implementa a exploitation do algoritmo).

Após isso foi a vez de implementar a função 'greedy_action()' que basicamente retorna a ação que maximiza o 'q' do estado passado como argumento.

1.1. SARSA

Para implementar o SARSA eu iniciei apenas chamando a 'get_exploratory_action()" da Classe abstrata RLAlgotithm na função 'get_greedy_action()' e para implementar a função 'learn()' eu apenas apliquei a fórmula do algoritmo SARSA:

$Q(s, a) \leftarrow Q(s, a) + \alpha * (r + \gamma * Q(s', a') - Q(s, a))$

1.2. Q-Learning

Para implementar o Q-Learning eu iniciei apenas chamando a função 'greedy_action()' definida no mesmo script. Após isso eu apenas segui a regra do algoritmo Q-Learn na função 'learn():

$Q(s, a) \leftarrow Q(s, a) + \alpha * (r + \gamma * \max_{a'} Q(s', a') - Q(s, a))$

2. Figuras Comprovando Funcionamento do Código

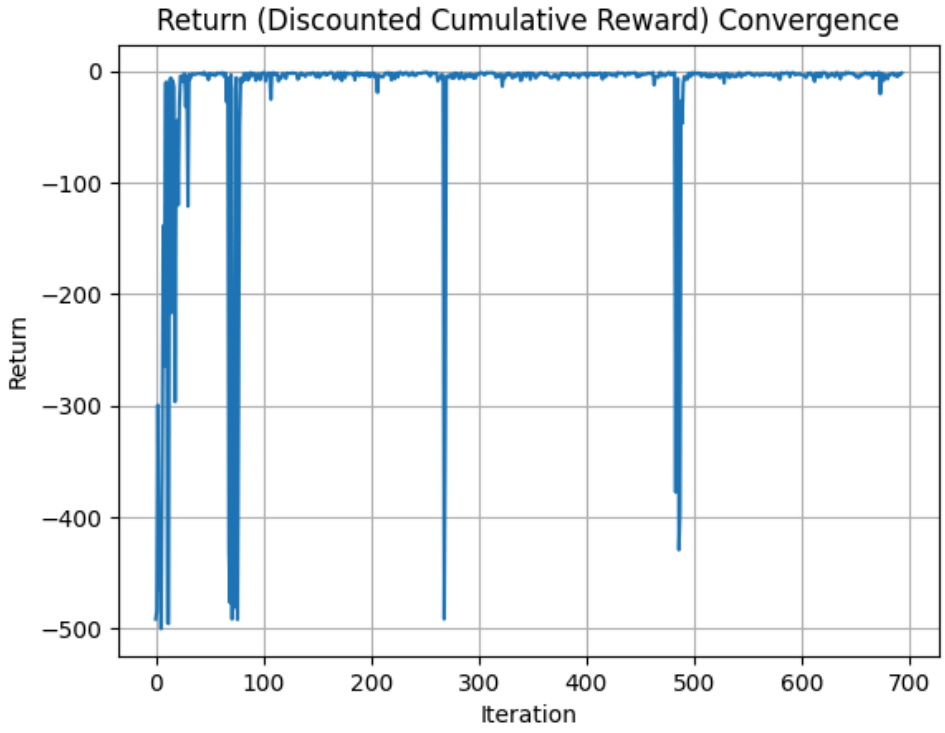
Basta colocar as figuras.

2.1. SARSA

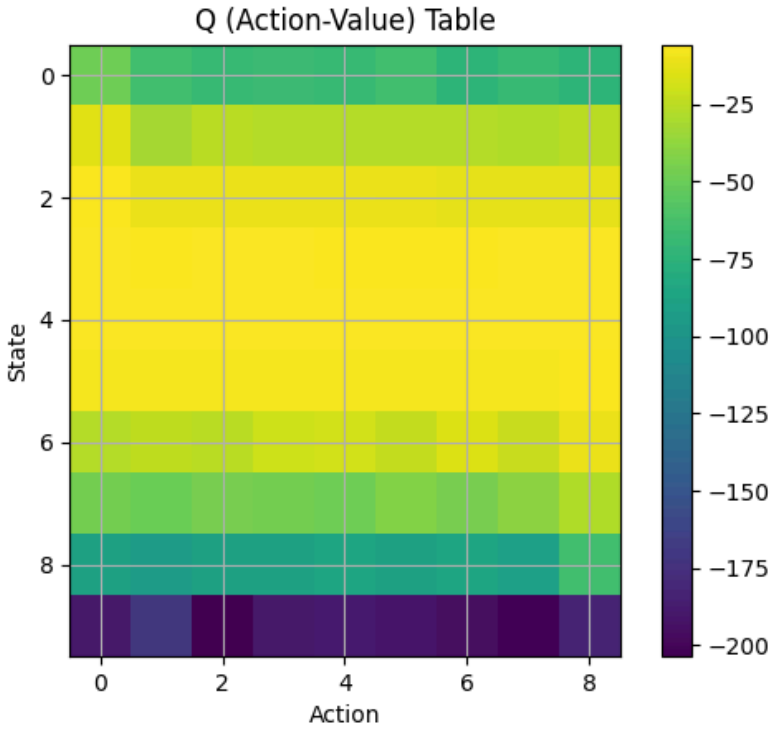
2.1.1. Tabela Ação-Valor e Política *Greedy* Aprendida no Teste com MDP Simples

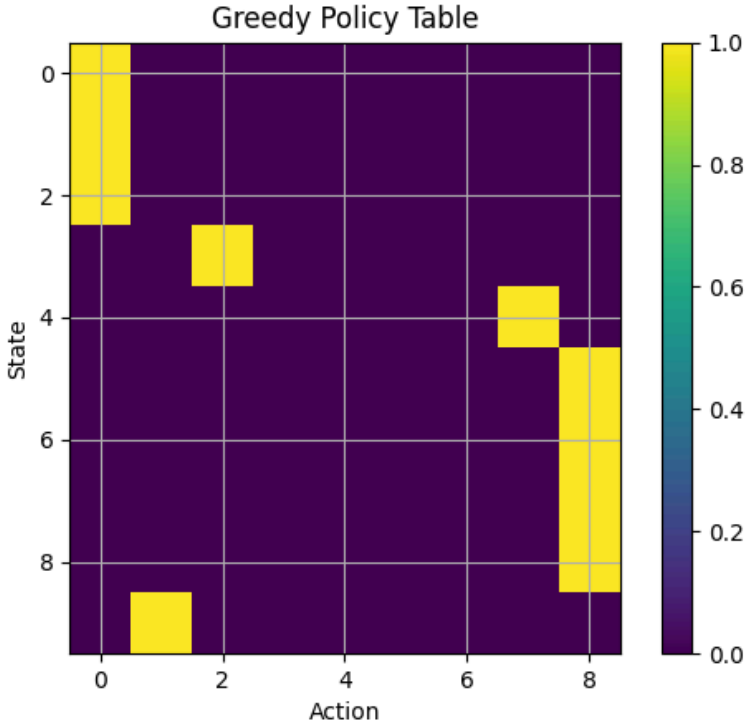
```
C:\Users\matos\PycharmProjects\CT213_Lab11\.venv\Scripts\python.exe C:\Users\matos\PycharmProjects\CT213_Lab11\test_rl.py
Action-value Table:
[[ -9.25123563  -8.38109611 -10.80200985]
 [-10.30515521  -9.45180818 -11.30501294]
 [-10.88384835 -10.46311789 -11.60099333]
 [-11.52063516 -11.35694885 -12.21085911]
 [-12.40902102 -12.29020365 -12.33731621]
 [-11.55166868 -12.34799927 -11.310157  ]
 [-11.0417592  -11.83880533 -10.27199789]
 [-10.26174854 -11.46294645  -9.19436368]
 [ -9.2737535  -10.32949477  -8.38971096],
 [ -7.32063033  -8.49417811  -8.60424969]]
Greedy policy learnt:
[L, L, L, L, L, R, R, R, R, S]

Process finished with exit code 0

```

2.1.2. Convergência do Retorno

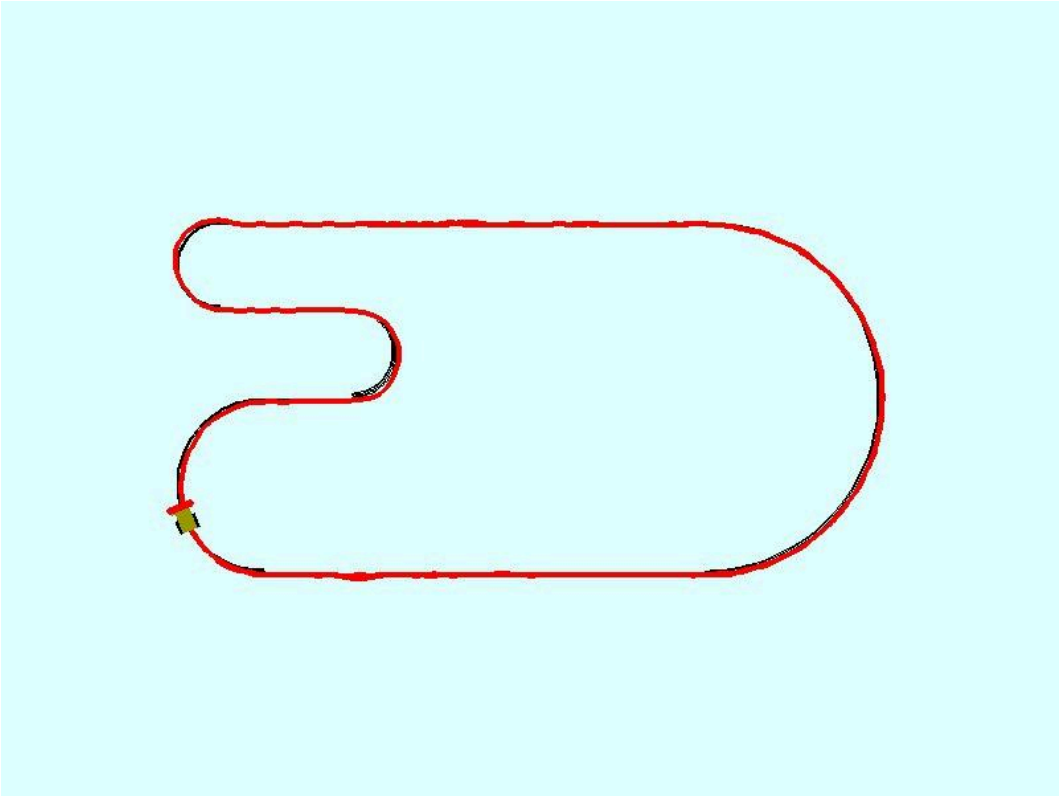


2.1.3. Tabela Q e Política Determinística que Seria Obtida Através de *Greedy*(Q)




Greedy Policy Table
State
Action
0
2
4
6
8
0
2
4
6
8
1.0
0.8
0.6
0.4
0.2
0.0

2.1.4. Melhor Trajetória Obtida Durante o Aprendizado



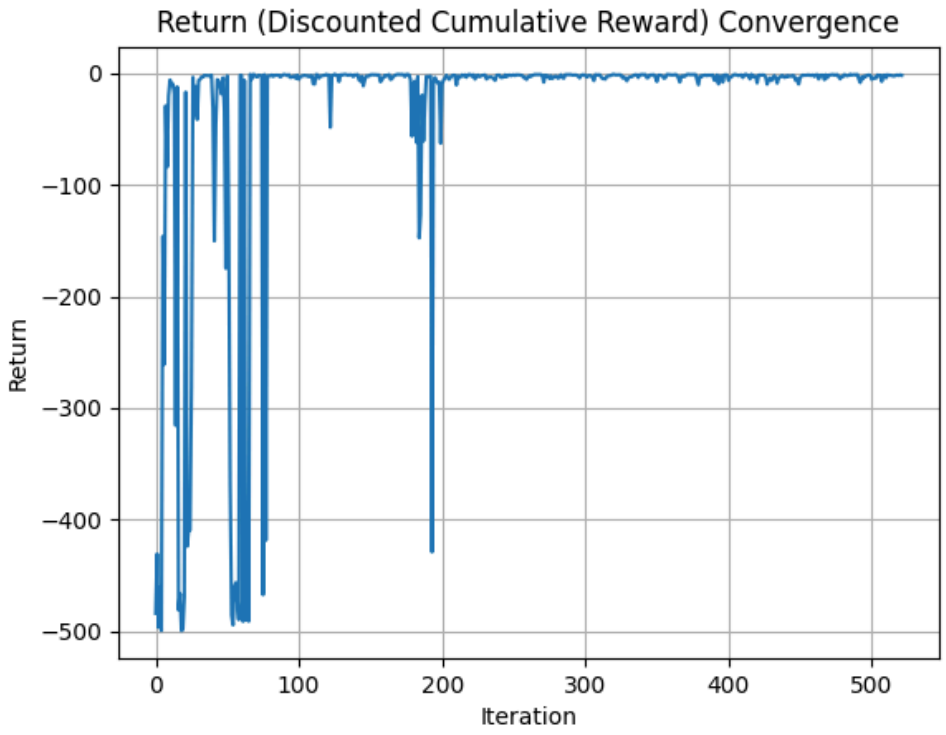
2.2. Q-Learning

2.2.1. Tabela Ação-Valor e Política *Greedy* Aprendida no Teste com MDP Simples

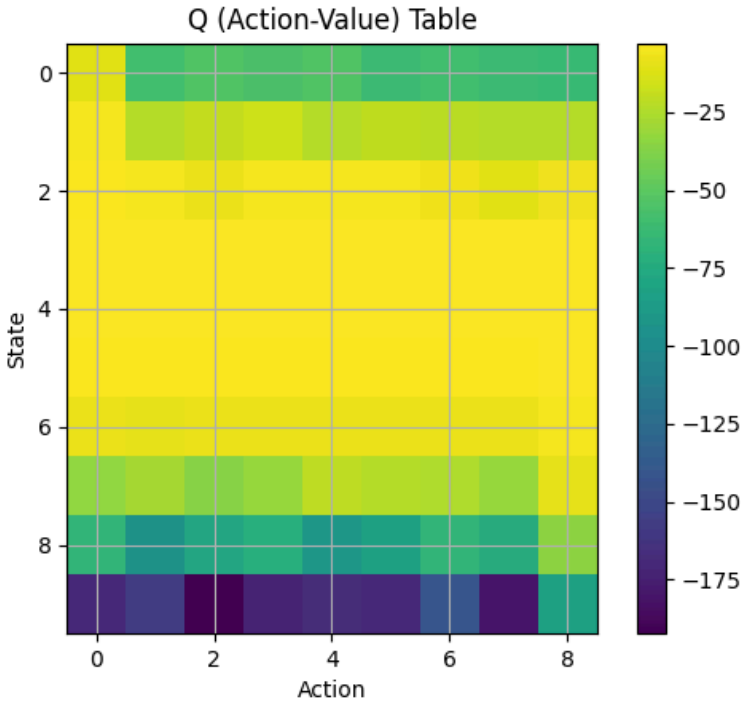
File - test_rl

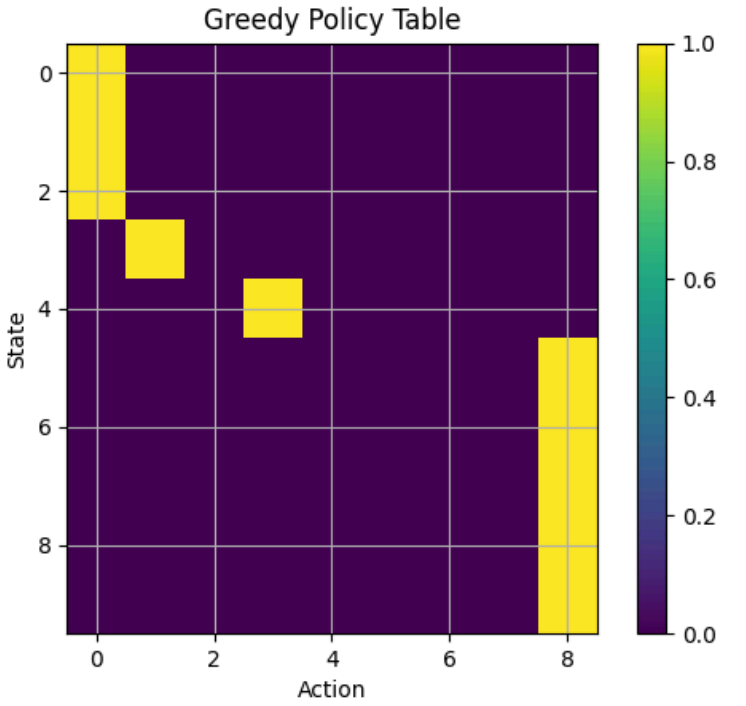
```
C:\Users\matos\PycharmProjects\CT213_Lab11\.venv\
Scripts\python.exe C:\Users\matos\PycharmProjects\
CT213_Lab11\test_rl.py
Action-value Table:
[[-1.99       -1.         -2.9701    ]
 [-2.96896105 -1.99       -3.92807528]
 [-3.75005984 -2.9701     -4.2399278 ]
 [-4.40246234 -3.94039889 -4.95193898]
 [-4.99670325 -4.89816994 -4.89811696]
 [-4.49969535 -4.64063658 -3.94039898]
 [-3.54816513 -4.40689546 -2.9701    ]
 [-2.96831026 -3.93447126 -1.99      ]
 [-1.99       -2.9701     -1.        ]
 [ 0.         -0.99       -0.99      ]]
Greedy policy learnt:
[L, L, L, L, R, R, R, R, R, S]

Process finished with exit code 0

```

2.2.2. Convergência do Retorno

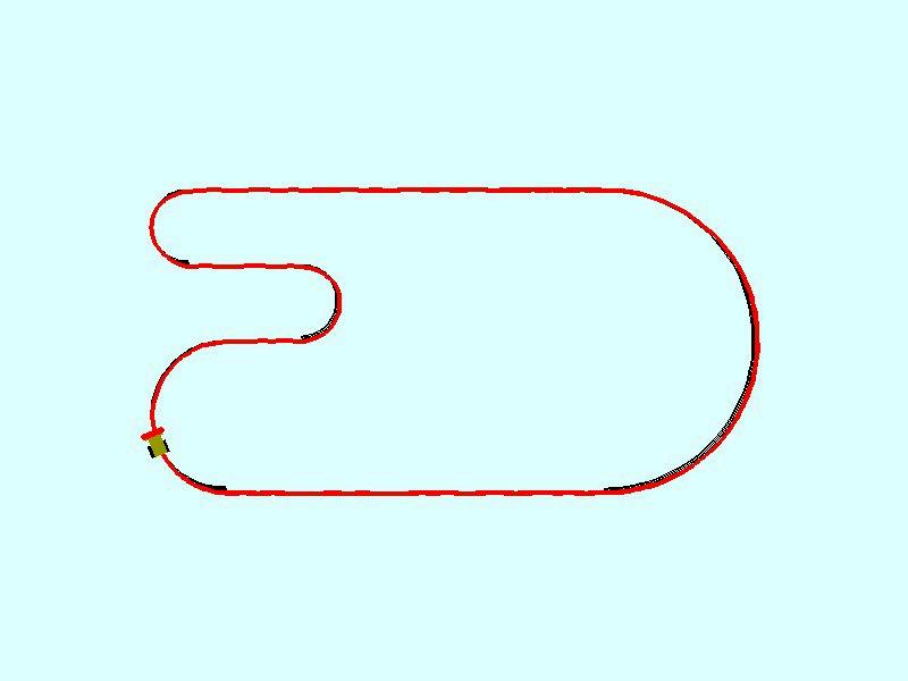


2.2.3. Tabela Q e Política Determinística que Seria Obtida Através de *Greedy*(Q)





2.2.4. Melhor Trajetória Obtida Durante o Aprendizado



3. Discussão dos Resultados

Ao rodar os testes ficou nítido que o Q-Learning consegue chegar na política ótima com os maiores valores de recompensa (a sua tabela de 'action-value' apresenta valores bem menores que os do SARSA.

Percebi também que o Q-Learn após chegar em uma convergência nas recompensas se mantém por lá (não acontecem picos de valores como ocorre no SARSA) pois após encontrar uma política greedy que possibilite a maior recompensa o Q-Learn não explora mais nada, apenas se mantém na execução da política ótima encontrada. Já o SARSA apresenta picos de valores de custo (recompensas negativas) porque sempre explora alguma ação diferente de uma política ótima encontrada (por meio da epsilon_greedy()). Outra coisa interessante foi notar que ambos os algoritmos aprenderam que o "bizú" é ficar entre os estados 3 e 6 para maximizar a recompensa mesmo seguindo políticas diferentes.