Centro Universitário FEI

***Aprendizado por Reforço***

**Aula 9 - Exercícios**

Disciplina:

Tópicos Especiais em Aprendizagem (Prof. Reinaldo A. C. Bianchi)

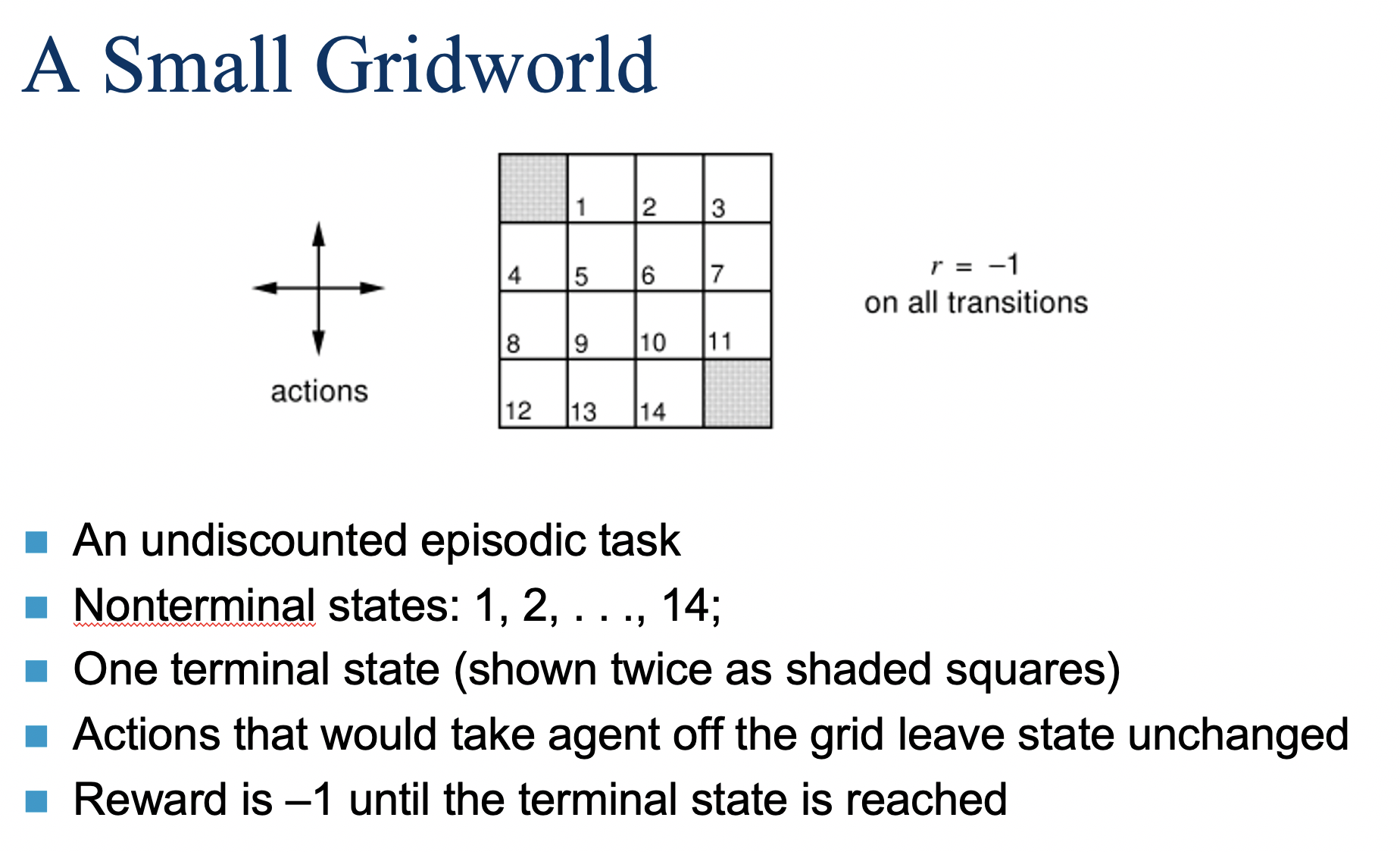
Aluno:

Claudio Aparecido Borges Junior (RA 120122-7)

São Bernardo do Campo

2o semestre / 2020

**EXERCÍCIO:** *Implemente o Small Grid World do slide 58 conforme imagem abaixo. Calcule a política ótima e o V para este problema, usando os algoritmos: (1) iteração de política, (2) iteração de valor e (3) método das diferenças temporais TD(0).*

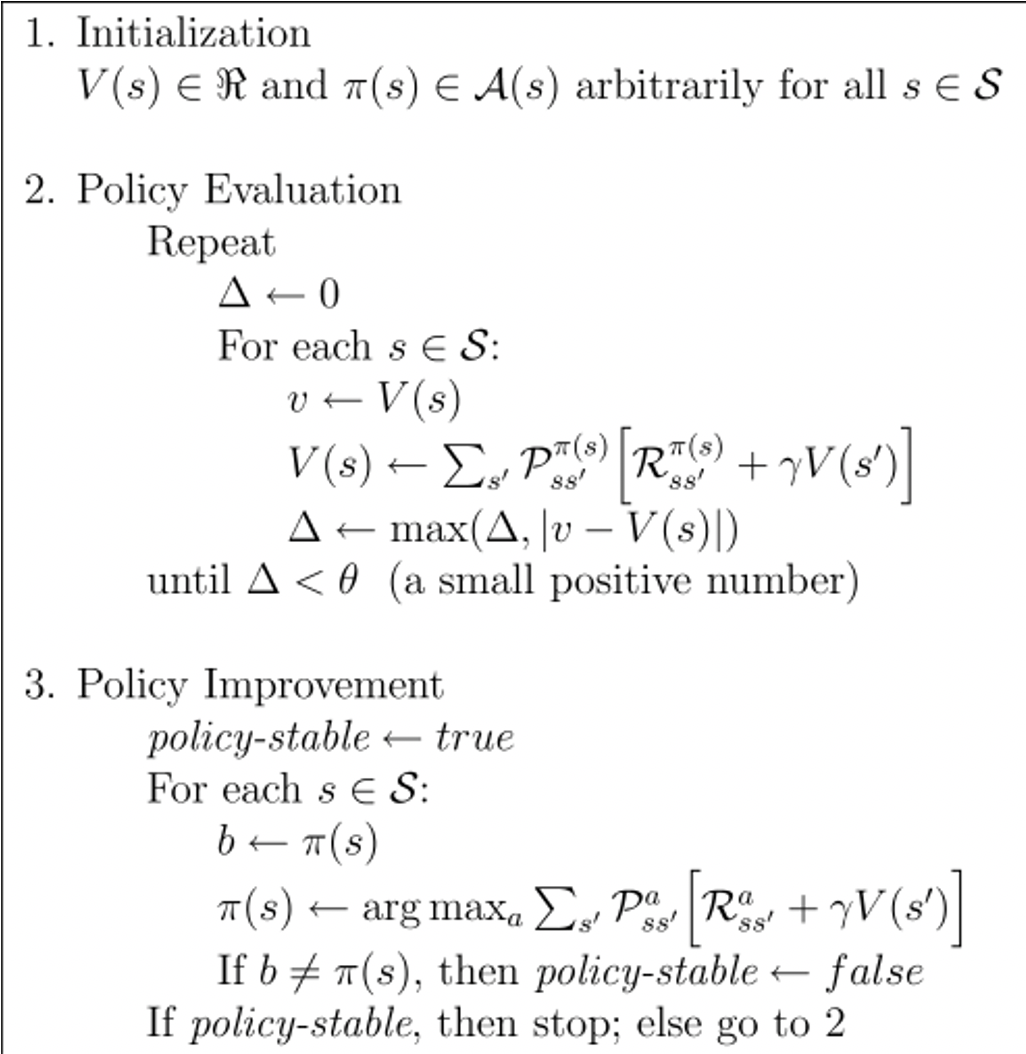
**

**RESPOSTA:**

O é o valor de um estado e é a expectativa do retorno começando nesse estado e depende da política do agente . A política de ações () modela o comportamento do agente, ou seja, mapeia estados em ações sendo que cada estado pode ter uma ou mais ações. A política ótima é aquela que maximiza .

**Método da Iteração de Política**

O algoritmo de iteração de política é dividido em 3 partes: inicialização, avaliação e melhoria. A inicialização atribui valores aleatórios aos valores dos estados e a política . Durante a fase de avaliação os valores dos estados são atualizados utilizando a probabilidade de uma política evoluir de um estado para multiplicado pelo retorno dessa evolução somado a ao valor do estado futuro amortizado por uma taxa gama. A melhoria da política busca a política ótima dado o conjunto de ações e os valores dos estados. O pseudocódigo pode ser observado a seguir:



O código do programa está no Anexo I, na linha 99. A saída do programa para esse método foi:

Iteração de Política

Valor dos estados (V)

[[0.0, -1.0, -2.0, -3.0],

[-1.0, -2.0, -3.0, -2.0],

[-2.0, -3.0, -2.0, -1.0],

[-3.0, -2.0, -1.0, 0.0]]

Política Ótima (Pi\*)

[[[], ['left'], ['left'], ['down', 'left']],

[['up'], ['up', 'left'], ['up', 'down', 'left', 'right'], ['down']],

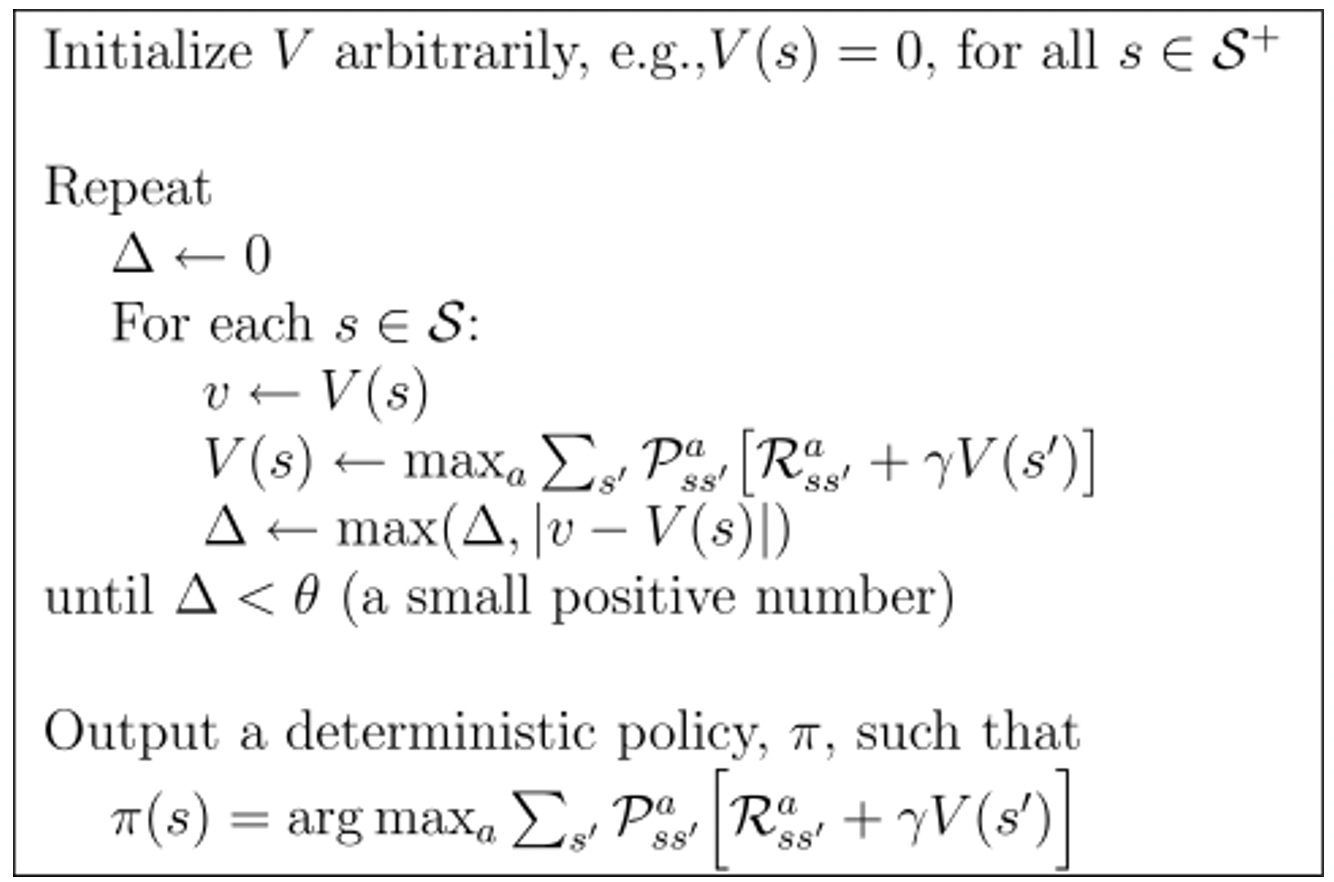
[['up'], ['up', 'down', 'left', 'right'], ['down', 'right'], ['down']],

[['up', 'right'], ['right'], ['right'], []]]

Indicando que o valor V é consistente, ou seja, representa o retorno esperado de um estado até o estado terminal. A política Pi é ótima, já que não existe nenhuma outra opção com retorno maior do que essa.

**Método da Iteração de Valor**

O algoritmo de iteração de valor é semelhante ao de iteração de política. Seu pseudocódigo é:



O código do programa está no Anexo I, na linha 163. A saída do programa para esse método foi:

Valor dos estados (V)

[[0.0, -0.25, -0.3125, -0.328125],

[-0.25, -0.3125, -0.328125, -0.3125],

[-0.3125, -0.328125, -0.3125, -0.25],

[-0.328125, -0.3125, -0.25, 0.0]]

Política Ótima (Pi\*)

[[[], ['left'], ['left'], ['down', 'left']],

[['up'], ['up', 'left'], ['up', 'down', 'left', 'right'], ['down']],

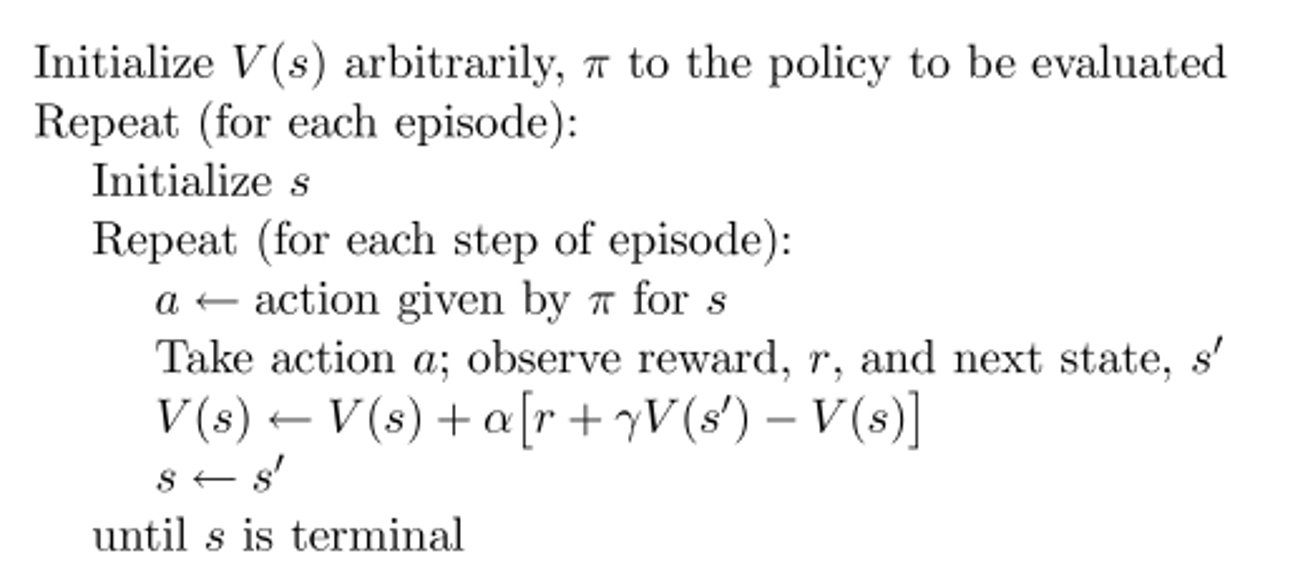
[['up'], ['up', 'down', 'left', 'right'], ['down', 'right'], ['down']],

[['up', 'right'], ['right'], ['right'], []]]

O valor dos estados já não indica o real resultado esperado para ir dele até um estado terminal, porém a política ótima apresenta todas as condições possíveis semelhante ao método de iteração de política.

**Método das Diferenças Temporais**

O método das diferenças temporais não necessita da probabilidade a priori de uma transição de estado dado uma ação. Para isso, ele utiliza um processo de amostragem, conhecido como Monte Carlo, para estimar as probabilidades. Seu pseudocódigo é:



O código do programa está no Anexo I, na linha 213. A saída do programa para esse método foi:

Diferença Temporal

Valor dos estados (V)

[[0.0, -13.688900197151094, -22.389018106044215, -23.27604781262058],

[-13.428248555433278,

-19.618414916329353,

-20.57174692816375,

-20.495187540523187],

[-19.252538739005487,

-20.25986410041081,

-17.04007564615278,

-12.474560142378678],

[-21.880419036211567, -20.571897094666344, -15.71981297766682, 0.0]]

Política Ótima (Pi\*)

[[[], ['left'], ['left'], ['down']],

[['up'], ['left'], ['down'], ['down']],

[['up'], ['right'], ['right'], ['down']],

[['up'], ['right'], ['right'], []]]

Devido ao processo e amostragem, a política ótima não possui todas as opções, porém apenas uma delas. Além disso, os valores dos estados não representam uma real estimativa de retorno.

**ANEXO I – CÓDIGO FONTE**

import numpy as np  
import random  
import pprint  
  
# Definitions  
DIM = 4  
  
# States  
S = [(i, j) for i in range(DIM) for j in range(DIM) if (i, j)]  
# Terminal States  
S\_terms = [(0, 0), (DIM - 1, DIM - 1)]  
# Actions  
A = {(-1, 0), (+1, 0), (0, +1), (0, -1)}  
# Action pretty names  
A\_name = {(+1, 0): "down", (-1, 0): "up", (0, +1): "right", (0, -1): "left"}  
  
  
def reward(s, a):  
 # Reward is constant  
 return -1  
  
  
def next\_state(S, s, a):  
 # Get the next state giving the current state and the action  
 min\_v = np.min(S)  
 max\_v = np.max(S)  
 return (  
 np.clip(s[0] + a[0], min\_v, max\_v), np.clip(s[1] + a[1], min\_v, max\_v))  
  
  
def pi\_namify(pi):  
 # Change the codes to names  
 namefied = [[[A\_name[v] for v in vals] for vals in row] for row in pi]  
 return namefied  
  
  
################################################################################  
# Policy Iteration  
################################################################################  
def policy\_iteration(S, S\_terms, V, A, Pi, gamma=1.0, eval\_err=1e-3):  
 policy\_stable = False  
 while not policy\_stable:  
 # Policy Evaluation  
 delta\_max = np.Inf  
 # Keep running until the maximu delta is small  
 while delta\_max > eval\_err:  
 delta\_max = 0  
 for s in S:  
 if s in S\_terms:  
 # Do not iterate terminal values  
 continue  
 # Backup the value  
 v = V[s]  
 partials = []  
 for a in Pi[s]:  
 # Apply the action in the state  
 s1 = next\_state(S, s, a)  
 # Reward function is constant  
 partial = reward(s, a) + gamma \* V[s1]  
 partials.append(partial)  
 # Calculate the new value.  
 V[s] = np.mean(partials)  
 # Calculate the delta  
 delta\_max = max(delta\_max, np.abs(v - V[s]))  
  
 # Policy Improvement  
 policy\_stable = True  
 for s in S:  
 if s in S\_terms:  
 # Do not iterate over terminal values  
 continue  
 b = Pi[s]  
 max\_val = -np.Inf  
 best\_actions = []  
 # Find the best actions  
 for a in A:  
 # Apply the action in the state  
 s1 = next\_state(S, s, a)  
 # Reward function is constant  
 val = reward(s, a) + gamma \* V[s1]  
 # print(val, max\_val, a)  
 if val == max\_val:  
 best\_actions.append(a)  
 elif val > max\_val:  
 best\_actions = [a]  
 max\_val = max(val, max\_val)  
 # print(best\_actions)  
 # Calculate the new value.  
 Pi[s] = best\_actions  
 if b != Pi[s]:  
 policy\_stable = False  
  
  
# Initialization  
V = np.zeros((DIM, DIM))  
Pi = np.array([[A if (i, j) not in S\_terms else {} for i in range(DIM)] for j in  
 range(DIM)])  
  
policy\_iteration(S, S\_terms, V, A, Pi)  
  
print("-" \* 80)  
print("Iteração de Política")  
print("Valro dos estados (V)")  
pprint.pprint(V.tolist())  
print("Política Ótima (Pi\*)")  
pprint.pprint(pi\_namify(Pi))  
print("-" \* 80)  
  
  
################################################################################  
# Value Iteration  
################################################################################  
def value\_iteraction(S, S\_terms, V, A, Pi, gamma=1.0, eval\_err=1e-3):  
 # Policy Evaluation  
 delta\_max = np.Inf  
 # Keep running until the maximum delta is small  
 while delta\_max > eval\_err:  
 delta\_max = 0  
 for s in S:  
 if s in S\_terms:  
 # Do not iterate terminal values  
 continue  
 # Backup the value  
 v = V[s]  
 partials = []  
 for a in A:  
 # Apply the action in the state  
 s1 = next\_state(S, s, a)  
 # Reward function is constant  
 partial = 1 / len(A) \* (reward(s, a) + gamma \* V[s1])  
 partials.append(partial)  
 # Calculate the new value. Completely different than the policy  
 # iteration  
 V[s] = np.max(partials)  
 # Calculate the delta  
 delta\_max = max(delta\_max, np.abs(v - V[s]))  
  
 # Calculate the Pi  
 for s in S:  
 if s in S\_terms:  
 # Do not iterate terminal values  
 continue  
 max\_val = -np.Inf  
 best\_actions = []  
 for a in A:  
 s1 = next\_state(S, s, a)  
 val = V[s1]  
 if val == max\_val:  
 best\_actions.append(a)  
 elif val > max\_val:  
 best\_actions = [a]  
 max\_val = max(val, max\_val)  
 # Find the greatest Value  
 Pi[s] = best\_actions  
  
  
# Initialization  
V = np.zeros((DIM, DIM))  
Pi = np.array([[A if (i, j) not in S\_terms else {} for i in range(DIM)] for j in  
 range(DIM)])  
  
# Run the Policy Iteraction  
value\_iteraction(S, S\_terms, V, A, Pi)  
  
print("-" \* 80)  
print("Iteração de Valor")  
print("Valor dos estados (V)")  
pprint.pprint(V.tolist())  
print("Política Ótima (Pi\*)")  
pprint.pprint(pi\_namify(Pi))  
  
  
################################################################################  
# Temporal difference  
################################################################################  
def temporal\_difference(S, S\_terms, V, A, Pi, alpha=0.1, gamma=1.0,  
 n\_episodes=10000):  
 for \_ in range(n\_episodes):  
 # Randomly select an action  
 s = random.choice(S)  
 while s not in S\_terms:  
 # Randomly select an action  
 a = random.choice(list(A))  
 # Apply the action in the state  
 s1 = next\_state(S, s, a)  
 V[s] = V[s] + alpha \* (reward(s, a) + gamma \* V[s1] - V[s])  
 s = s1  
  
 # Calculate the Pi  
 for s in S:  
 if s in S\_terms:  
 # Do not iterate terminal values  
 continue  
 max\_val = -np.Inf  
 best\_actions = []  
 for a in A:  
 s1 = next\_state(S, s, a)  
 val = V[s1]  
 if val == max\_val:  
 best\_actions.append(a)  
 elif val > max\_val:  
 best\_actions = [a]  
 max\_val = max(val, max\_val)  
 # Find the greatest Value  
 Pi[s] = best\_actions  
  
  
V = np.zeros((DIM, DIM))  
Pi = np.array([[A if (i, j) not in S\_terms else {} for i in range(DIM)] for j in  
 range(DIM)])  
  
# Run the Policy Iteraction  
temporal\_difference(S, S\_terms, V, A, Pi)  
  
  
print("-" \* 80)  
print("Diferença Temporal")  
print("Valor dos estados (V)")  
pprint.pprint(V.tolist())  
print("Política Ótima (Pi\*)")  
pprint.pprint(pi\_namify(Pi))