Inteligência Artificial

Prof. Denis Fantinato

3º Quadrimestre de 2018

Inteligência Artificial

Implementação

- MDP:
  - Algoritmo Valor-Iteração
  - Algoritmo Política-Iteração
- Aprendizado por Reforço:
  - Algoritmo de Estimação Direta
  - Algortimo TD (Diferença Temporal)
  - Algoritmo Q-learning
  - Estimando Funções
- Busca por Política
  - Algoritmo Hill Climbing

MDP

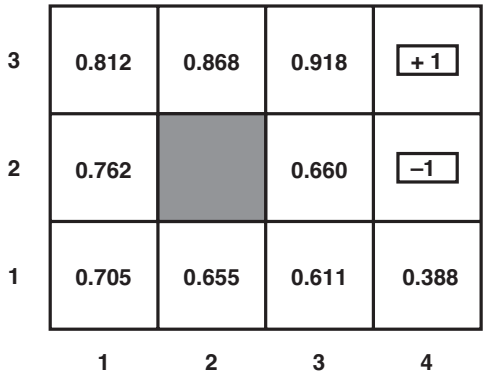
Formalização Matemática

$MDP = (S, A, R, \mathbb{P}, \gamma)$

com:

- S: conjunto de possíveis estados.
- A: conjunto de ações.
- $R : S \to \mathbb{R}$: mapa de recompensa para cada estado.
- $\mathbb{P}$: probabilidade de transição de um estado para outro dada uma ação.
- γ: fator de desconto. Um número entre 0 e 1.

Aplicação no Mundo 4×3:




```
def createMDP():

    '''
    definição do ambiente
    '''

    S       = [(i,j) for i in range(1,5)
                     for j in range(1,4) if (i,j) != (2,2)]

    goals   = [(4,3), (4,2)]
    actions = ["UP", "DOWN", "LEFT", "RIGHT"]
    A       = {s : actions if s not in goals else [None]
               for s in S }

    R        = {s : -0.04 for s in S}
    R[(4,3)] =  1
    R[(4,2)] = -1

    P        = { (s,a) : pvals(s, a, S) for s in S for a in A[s] }

    gamma    = .9

    return (S,A,R,P,gamma)
```

Responda:

- Como ficaria a lista S? Escreva os estados na ordem correta.

- O que há em:
  - 'A[(3,2)]'

  - 'A[(2,2)]'

  - 'A[(4,2)]'

```
def pvals(s, a, S):
    return [(0.8, move(s, a, S)), (0.1, move(s, succ(a), S)), (0.1, move(s, pred(a), S))]

def succ(a):
    return {"UP": "RIGHT", "DOWN": "LEFT", "RIGHT": "DOWN", "LEFT": "UP", None : None}[a]

def pred(a):
    return {"UP": "LEFT", "DOWN": "RIGHT", "RIGHT": "UP", "LEFT": "DOWN", None : None}[a]


def move(s, a, S):
    i, j = s
    if a == "UP":
        sp = (i, j+1)
    elif a == "DOWN":
        sp = (i, j-1)
    elif a == "LEFT":
        sp = (i-1, j)
    elif a == "RIGHT":
        sp = (i+1, j)
    elif a is None:
        return s

    if sp in S:
        return sp

    return s
```

Responda:
- Qual a saída para `P[((3,3),"RIGHT")] = pvals((3,3), "RIGHT", S)`?
- O que acontece se o resultado for um estado fora do grid?

Algoritmo Iteração-Valor

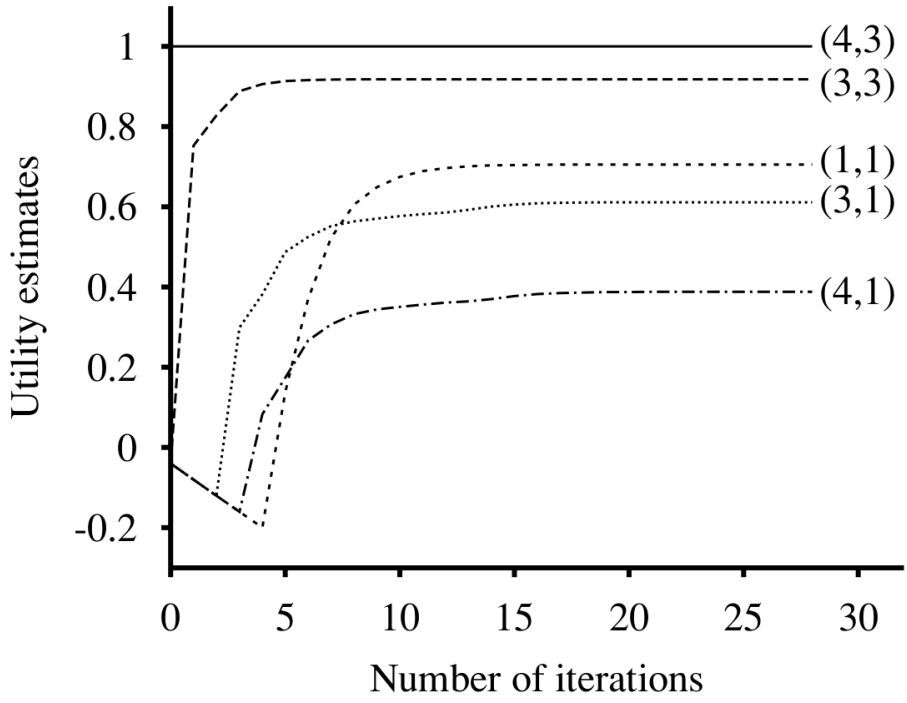
Processo iterativo para determinar os valores de utilidade de cada estado. Seja $U_i(s)$ a utilidade para o estado s na i-ésima iteração. Cada iteração atualiza os valores de U como:

$$U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

em que a atualização é feita simultaneamente para todos os estados.

```
def valueIteration(mdp, eps):

    S, A, R, P, gamma = mdp

    # Utilidade inicial nula
    Uprime = { s : 0.0 for s in S }

    # delta inicial é o limite + 1 para entrar no laço
    delta  = eps*(1 - gamma)/gamma + 1.

    while delta > eps*(1 - gamma)/gamma:
        U     = deepcopy(Uprime)
        delta = 0
        for s in S:
            if None in A[s]:
                Uprime[s] = R[s]
            else:
                Uprime[s] = R[s] + gamma * max(expVal(P[(s,a)],U) for a in A[s])
            delta     = max(delta, abs(Uprime[s] - U[s]))

    return U

def expVal(ps, U):
    '''
    retorna o valor esperado da utilidade dadas as probabilidades
    de possíveis estados consequentes armazenados em ps.
    '''
    return sum(p*U[s] for p, s in ps)
```

Responda:
- Qual a saída de `expVal(P[((3,3),"RIGHT")],U)`, assumindo `U[(3,2)]=1/2`, `U[(3,4)]=1/4` e `U[(3,3)]=1`?


1
0.8
0.6
0.4
0.2
0
-0.2
Utility estimates
0
5
10
15
20
25
30
Number of iterations
(4,3)
(3,3)
(1,1)
(3,1)
(4,1)

Algoritmo Iteração-Política

Uma outra abordagem é estimar a melhor política e determinar o valor de U a partir dela através de dois passos:

- **Avaliação de política:** dada uma política π, calcule U^{π}, assumindo que π é ótimo.
- **Melhoria da política:** atualize a política π baseado nos valores de U^{π}.

Na avaliação de política, relaciona a utilidade de s dado π_i com a utilidade dos vizinhos:

$$U_{i+1}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s')$$

```
def policyIteration(mdp):

    S, A, R, P, gamma = mdp

    U       = { s : 0.0 for s in S }
    # política inicial executa a primeira ação da lista
    pi      = { s : A[s][0] for s in S }
    changed = True

    while changed:
        U       = policyEvaluation(pi, U, mdp)
        changed = False
        for s in S:
            # maior valor esperado utilizando a matriz utilidade atual
            maxU = max(expVal(P[(s,a)],U) for a in A[s])
            # maior valor esperado utilizando a política atual
            piU  = expVal(P[(s,pi[s])],U)

            # Se a utilidade ganhar, atualiza a política
            if maxU > piU:
                idx     = np.argmax( [expVal(P[(s,a)],U) for a in A[s]] )
                pi[s]   = A[s][idx]
                changed = True
                #print(s, maxU, piU)
    return pi

def policyEvaluation(pi, U, mdp):
    S, A, R, P, gamma = mdp
    for s in S:
        U[s] = R[s] + gamma * expVal(P[(s, pi[s])],U)
    return U
```

Responda:

- O que há em pi[(3,3)]]?
- Faça a iteração inicial para s = (3,3) considerando:
- expVal(P[((3,3), "UP")],U) = 0.064
- expVal(P[((3,3), "DOWN")],U) = 0.064
- expVal(P[((3,3), "LEFT")],U) = -0.04
- expVal(P[((3,3), "RIGHT")],U) = 0.792

Função Principal:

```
def main():
    mdp = createMDP()
    U  = valueIteration(mdp, 1e-3)
    pi = policyIteration(mdp)

    print("Value Iteration: \n")
    for j in range(3,0,-1):
        for i in range(1,5):
            if (i,j) in U:
                print(f"|  {U[(i,j)]:.2f}  |", end="")
            else:
                print("|       |", end="")
        print("")

    print("\n\nPolicy Iteration: \n")
    for j in range(3,0,-1):
        for i in range(1,5):
            if (i,j) in U:
                print(f"|  {pi[(i,j)]}  |", end="")
            else:
                print("|   |", end="")
        print("")

main()
```

Resultado:

```
Value Iteration:

|  0.51  ||  0.65  ||  0.80  ||  1.00   |
|  0.40  ||        ||  0.49  ||  -1.00  |
|  0.30  ||  0.25  ||  0.34  ||  0.13   |


Policy Iteration:

|  RIGHT  ||  RIGHT  ||  RIGHT  ||  None  |
|  UP     ||         ||  UP     ||  None  |
|  UP     ||  RIGHT  ||  UP     ||  LEFT  |
```

Aprendizado por Reforço

Estimação Direta (Aprendizado Passivo)

Usa a seguinte relação para estimar a utilidade:

$$U^{\pi}(s) = E_s \left[\sum_{t=0}^{\infty} \gamma^t R(S_t) \right]$$

É necessário obter as sequências-amostras e depois estimar as utilidades.

```
def runDirectEst(mdp, pi, nTrials):
    S, A, R, P, gamma = mdp

    model = defaultdict(lambda: (0.0, 0))
    s0    = (1,1)
    goals = [(4,3), (4,2)]

    for trials in range(nTrials):
        model = directEst(model, s0, goals, R, gamma, performAction(pi, P))
        if model is None:
            break

    return model


def directEst(model, s, goals, R, gamma, nextState, maxLen=100):
    trial = [(s, R[s])]
    while s not in goals:
        s = nextState(s)
        trial.append( (s, R[s]) )
        if len(trial) > maxLen:
            return None
    for i, (s, r) in enumerate(trial):
        u        = sum(r*(gamma**j)
                       for j, (si, r) in enumerate(trial[i:]))
        model[s] = (model[s][0] + u, model[s][1] + 1)
    return model

# Escolhe próximo estado dado uma ação
def performAction(pi, P):
    def nextState(s):
        ps     = P[(s, pi[s])]
        probs  = list(map(lambda x: x[0], ps))
        states = list(map(lambda x: x[1], ps))
        idx    = np.random.choice(len(states), p=probs)
```

```
        return states[idx]
    return nextState
```

Exemplo:
- Uma sequência-amostra:

```
model = {(1, 1): (0.34401, 1), (1, 2): (0.42668, 1), (1, 3): (1.13914, 2),
(2, 3): (0.734, 1), (3, 3): (0.86, 1), (4, 3): (1.0, 1)})
```

- O que há na variável `model`?

Estado (1,3) acontece duas vezes.

```
Direct Estimation:

|  0.57  ||  0.73  ||  0.86  ||  1.00  |
|  0.43  ||        ||        ||        |
|  0.34  ||        ||        ||        |
```

- Por que o valor do estado (1,3) é 0.57?

TD (Aprendizado Passivo)

O algoritmo TD usa a informação da Equação de Bellman.

$$U^{\pi}(s) = U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

```
# Time difference
def td(percept, state, gamma, alpha):
    sn, rn          = percept
    s, a, r, pi, U = state

    if sn not in U:
        U[sn] = rn
    if s != (0,0):
        U[s]  = U[s] + alpha*(r + gamma*U[sn] - U[s])
    return pi[sn], (sn, pi[sn], rn, pi, U)

def runTD(mdp, pi, nTrials, alpha):
    S, A, R, P, gamma = mdp
    U         = {}
    nextState = performAction(pi, P)
    s0        = (1,1)
    goals     = [(4,3), (4,2)]

    for t in range(nTrials):
        # Estado inicial nulo
        state = (0,0), "", 0, pi, U
        s = s0
        percept = (s, R[s])
        g       = gamma
        while state[0] not in goals:
            a, state = td(percept, state, g, alpha)
            s        = nextState(s)
            percept  = (s, R[s])

    s, a, r, pi, U = state
    return U
```

Responda:
- Qual o motivo do uso do estado nulo (`state = (0,0), "", 0, pi, U`)?
- O seguinte resultado foi obtido usando-se 1 trial e `alpha = 1`:

```
Time Difference:

|  -0.08  ||  -0.08  ||  0.86  ||  1.00  |
|  -0.08  ||         ||        ||        |
|  -0.08  ||         ||        ||        |
```

Algum estado repetiu? Por que foram observados esses valores?

Algoritmo Q-learning

Q-Learning é um algoritmo de diferença temporal que aprende uma função $Q(s,a)$ que indica a qualidade em escolher a ação a no estado s.

Similar a diferença temporal, no algoritmo Q-Learning atualizamos a função $Q(s,a)$ como:

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

```
def qlearn(percept, state, goals, A, gamma, alpha, f):
    sn, rn        = percept
    s, a, r, N, Q = state

    # se for estado final, usa a recompensa
    if sn in goals:
        Q[(sn, None)] = rn

    # se s nao for nulo, atualiza Q
    if s is not None:
        N[(s,a)] = N[(s,a)] + 1
        maxQ     = max(Q[(sn,ai)] for ai in A[sn])
        Q[(s,a)] = Q[(s,a)] + alpha * (r + gamma*maxQ - Q[(s,a)])

    # avalia o proximo estado e a acao que deve ser tomada
    if s in goals:
        state = (None, None, 0, N, Q)
    else:
        qvals    = [(f(Q[(sn,ai)], N[(sn,ai)]), ai) for ai in A[sn]]
        an       = max(qvals, key=lambda x: x[0])[1]
        state    = (sn, an, rn, N, Q)
    return an, state


def createF(nMax, rPlus):
    def f(q,n):
        if n < nMax:
            return rPlus
        return q
    return f
```

Responda:
- Se `nMax = 10`, `rPlus = 2`, o que a função `f(3.4,4)` e `f(3.8, 11)` retornam?

```
def runQ(mdp, nTrials, alpha, nMax, rPlus):
    S, A, R, P, gamma = mdp
```

```
    s0        = (1,1)
    goals     = [(4,3), (4,2)]

    Q = defaultdict(float)
    N = defaultdict(int)
    f = createF(nMax, rPlus)

    for t in range(nTrials):
        # Estado inicial nulo
        state = None, None, 0, N, Q
        s = s0
        percept = (s, R[s])
        while state[0] not in goals:
            a, state = qlearn(percept, state, goals, A, gamma, alpha, f)
            if s not in goals:
                s   = performQAction(s, a, P)
            percept = (s, R[s])

    # Transforma Q-values em politica
    s, a, r, N, Q = state
    pi            = {}
    for s in S:
        qvals    = [(Q[(s,a)], a) for a in A[s]]
        maxQ, an = max(qvals, key=lambda x: x[0])
        pi[s] = an
    return pi

def performQAction(s, a, P):
    ps     = P[(s, a)]
    probs  = list(map(lambda x: x[0], ps))
    states = list(map(lambda x: x[1], ps))
    idx    = np.random.choice(len(states), p=probs)

    return states[idx]
```

Exemplo:
- 1 trial:

```
{((1, 1), 'UP'): -0.04, ((1, 1), 'DOWN'): -0.04,
((1, 1), 'LEFT'): 0.0, ((1, 1), 'RIGHT'): 0.0,
((1, 2), 'UP'): -0.04, ((1, 2), 'DOWN'): -0.04,
((1, 2), 'LEFT'): 0.0, ((1, 2), 'RIGHT'): 0.0,
((1, 3), 'UP'): -0.04, ((1, 3), 'DOWN'): -0.04,
((1, 3), 'LEFT'): -0.04, ((1, 3), 'RIGHT'): 0.0,
((2, 3), 'UP'): -0.04, ((2, 3), 'DOWN'): -0.04,
((2, 3), 'LEFT'): -0.04, ((2, 3), 'RIGHT'): 0.0,
((3, 3), 'UP'): -0.04, ((3, 3), 'DOWN'): -0.04,
```

```
((3, 3), 'LEFT'): 0.0, ((3, 3), 'RIGHT'): 0.0,
((3, 2), 'UP'): -0.04, ((3, 2), 'DOWN'): 0.0,
((3, 2), 'LEFT'): 0.0, ((3, 2), 'RIGHT'): 0.0,
((2, 1), 'UP'): -0.04, ((2, 1), 'DOWN'): 0.0,
((2, 1), 'LEFT'): 0.0, ((2, 1), 'RIGHT'): 0.0,
((3, 1), 'UP'): -0.04, ((3, 1), 'DOWN'): 0.0,
((3, 1), 'LEFT'): 0.0, ((3, 1), 'RIGHT'): 0.0,
((4, 1), 'UP'): -0.94, ((4, 1), 'DOWN'): 0.0,
((4, 1), 'LEFT'): 0.0, ((4, 1), 'RIGHT'): 0.0,
((4, 2), None): -1, ((4, 3), None): 0.0}
```

- 1000 trials:

```
{((1, 1), 'UP'): 0.295, ((1, 1), 'DOWN'): -0.0751,
((1, 1), 'LEFT'): -0.0754, ((1, 1), 'RIGHT'): -0.0755,
((2, 1), 'UP'): -0.0585, ((2, 1), 'DOWN'): -0.0592,
((2, 1), 'LEFT'): 0.1895, ((2, 1), 'RIGHT'): -0.0576,
((1, 2), 'UP'): 0.3855, ((1, 2), 'DOWN'): -0.05272,
((1, 2), 'LEFT'): -0.05070, ((1, 2), 'RIGHT'): -0.052498, ...
```

Q-Learning:

```
|  UP     ||  UP    ||  UP  ||  None  |
|  RIGHT  ||        ||  UP  ||  None  |
|  DOWN   ||  DOWN  ||  UP  ||  UP    |
```

Estimando um modelo de função para a Utilidade

```
def utilityModel(nTrials, S, R, s, goals, gamma, nextState):
    dataX = []
    dataY = []
    for t in range(nTrials):
        xTrial, yTrial = makeTrial(s, goals, R, gamma, nextState)
        dataX = dataX + xTrial
        dataY = dataY + yTrial

    dataXn = np.zeros((len(dataX), 4))
    dataXn[:, :-1] = dataX
    dataXn[:, -1] = (dataXn[:,0] - 4)**2 + (dataXn[:,1] * 3)**2
    lr = LinearRegression()
    lr.fit(dataXn, dataY) # Obtem os coeficientes

    model = lambda s: lr.predict([[1, s[0], s[1], (s[0] - 4)**2 + (s[1] - 3)**2]])[0]

    print(lr.coef_, lr.intercept_)

    return { s : model(s) for s in S }

def makeTrial(s, goals, R, gamma, nextState):
    trial  = [(s, R[s])]
    xTrial = []
    yTrial = []
    while s not in goals:
        s = nextState(s)
        trial.append( (s, R[s]) )
    for i, (s,r) in enumerate(trial):
        x, y     = s
        u        = sum(r*(gamma**j)
                       for j, (si, r) in enumerate(trial[i:]))
        xTrial.append([1,x,y])
        yTrial.append(u)
    return xTrial, yTrial
```

Exemplo:

```
>>> print(dataX)
[[1 1 1]
 [1 1 2]
 [1 1 3]
 [1 2 3]
 [1 3 3]
 [1 4 3]]
```

```
>>> dataXn = np.zeros((len(dataX), 4))
    dataXn[:, :-1] = dataX
>>> print(dataXn)
[[1. 1. 1. 0.]
 [1. 1. 2. 0.]
 [1. 1. 3. 0.]
 [1. 2. 3. 0.]
 [1. 3. 3. 0.]
 [1. 4. 3. 0.]]


>>> dataXn[:, -1] = (dataXn[:,0] - 4)**2 + (dataXn[:,1] * 3)**2
>>> print(dataXn)
[[  1.   1.   1.  18.]
 [  1.   1.   2.  18.]
 [  1.   1.   3.  18.]
 [  1.   2.   3.  45.]
 [  1.   3.   3.  90.]
 [  1.   4.   3. 153.]]
```

Responda:
- Quais são as funções f_0, f_1, f_2 e f_3?

```
def runUtilityModel(mdp, pi, nTrials):
    S, A, R, P, gamma = mdp

    model = defaultdict(lambda: (0.0, 0))
    s0    = (1,1)
    goals = [(4,3), (4,2)]

    model = utilityModel(nTrials, S, R, s0, goals, gamma, performAction(pi, P))

    return model
```

Exemplo:
A variável `model` irá conter o valor de utilidade esperada para cada estado a partir da função obtida:

```
model = {(1, 1): 0.217, (1, 2): 0.373, (1, 3): 0.535, (2, 1): 0.189, (2, 3): 0.507,
(3, 1): 0.168, (3, 2): 0.324, (3, 3): 0.486, (4, 1): 0.152, (4, 2): 0.309, (4, 3): 0.471}
```

Responda:
- Como você alteraria o código para incluir mais uma característica f_4?

Hill-Climbing:

```
def hillClimbing(mdp, s0, goals, nTrials):
    S, A, R, P, gamma = mdp

    piCurrent = { s : np.random.choice(A[s]) for s in S }

    Ucurrent  = evaluate(mdp, piCurrent, nTrials)
    improved  = True

    while improved:
        improved             = False
        bestNeighbor, Uneigh = genNeighbors(mdp, piCurrent, s0, goals, nTrials)

        if Uneigh[s0] > Ucurrent[s0]:
            improved  = True
            Ucurrent  = deepcopy(Uneigh)
            piCurrent = deepcopy(bestNeighbor)
    return piCurrent
```

Responda:
- Como a política inicial é ajustada?

```
def evaluate(mdp, pi, nTrials):
    model = runDirectEst(mdp, pi, nTrials)
    if model is None:
        U = { s:-np.inf for s in mdp[0] }
    else:
        U     = {}
        for s, (v, n) in model.items():
            U[s] = v/n
    return U

def genNeighbors(mdp, pi, s0, goals, nTrials):
    S, A, R, P, gamma = mdp

    candidates = [(s,a) for s in S for a in A[s]
                  if s not in goals and a != pi[s]]

    bestNeighbor = deepcopy(pi)
    Uneigh       = evaluate(mdp, pi, nTrials)
    for s, a in candidates:
       oldAction = pi[s]
       pi[s]     = a

       Ucand     = evaluate(mdp, pi, nTrials)
       if Ucand[s0] > Uneigh[s0]:
```

```
            Uneigh       = deepcopy(Ucand)
            bestNeighbor = deepcopy(pi)

        pi[s]     = oldAction

    return bestNeighbor, Uneigh
```

Responda:
- Qual estimativa é feita para avaliar cada candidato à solução?
- Qual a política é mantida?

Chamada Principal:

Para executar os programas:

```
def main():
    mdp = createMDP()
    pi  = policyIteration(mdp)

    model = runDirectEst(mdp, pi, 1000)
    print(model)


    print("Direct Estimation: \n")
    for j in range(3,0,-1):
        for i in range(1,5):
            if (i,j) in model:
                soma, n = model[(i,j)]
                v       = soma / n
                print(f"|  {v:.2f}  |", end="")
            else:
                print("|       |", end="")
        print("")

    U = runTD(mdp, pi, 1000, 0.1)

    print("Time Difference: \n")
    for j in range(3,0,-1):
        for i in range(1,5):
            if (i,j) in U:
                v = U[(i,j)]
                print(f"|  {v:.2f}  |", end="")
            else:
                print("|       |", end="")
        print("")

    piQ = runQ(mdp, 1000, 0.1, 10, 2)

    print("Q-Learning: \n")
    for j in range(3,0,-1):
        for i in range(1,5):
            if (i,j) in piQ:
                v = piQ[(i,j)]
                print(f"|  {v}  |", end="")
            else:
                print("|       |", end="")
        print("")
```

```
    U = runUtilityModel(mdp, pi, 1000)

    print("Utility Model: \n")
    for j in range(3,0,-1):
        for i in range(1,5):
            if (i,j) in U:
                v = U[(i,j)]
                print(f"|  {v:.2f}  |", end="")
            else:
                print("|       |", end="")
        print("")

    piHC = hillClimbing(mdp, (1,1), [(4,2), (4,3)], 10)

    print("Hill Climbing: \n")
    for j in range(3,0,-1):
        for i in range(1,5):
            if (i,j) in piHC:
                v = piHC[(i,j)]
                print(f"|  {v}  |", end="")
            else:
                print("|       |", end="")
        print("")

main()
```