숙제:

내용 :

텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

관련 코드 정리

1. Gridworld\_ernder.py

|  |
| --- |
| 코드 |
| import numpy as np import matplotlib import matplotlib.pyplot as plt   class Renderer:  def \_\_init\_\_(self, reward\_map, goal\_state, wall\_state):  self.reward\_map = reward\_map  self.goal\_state = goal\_state  self.wall\_state = wall\_state  self.ys = len(self.reward\_map)  self.xs = len(self.reward\_map[0])   self.ax = None  self.fig = None  self.first\_flg = True   def set\_figure(self, figsize=None):  fig = plt.figure(figsize=figsize)  self.ax = fig.add\_subplot(111)  ax = self.ax  ax.clear()  ax.tick\_params(labelbottom=False, labelleft=False, labelright=False, labeltop=False)  ax.set\_xticks(range(self.xs))  ax.set\_yticks(range(self.ys))  ax.set\_xlim(0, self.xs)  ax.set\_ylim(0, self.ys)  ax.grid(True)   def render\_v(self, v=None, policy=None, print\_value=True):  self.set\_figure()   ys, xs = self.ys, self.xs  ax = self.ax   if v is not None:  color\_list = ['red', 'white', 'green']  cmap = matplotlib.colors.LinearSegmentedColormap.from\_list(  'colormap\_name', color\_list)   # dict -> ndarray  v\_dict = v  v = np.zeros(self.reward\_map.shape)  for state, value in v\_dict.items():  v[state] = value   vmax, vmin = v.max(), v.min()  vmax = max(vmax, abs(vmin))  vmin = -1 \* vmax  vmax = 1 if vmax < 1 else vmax  vmin = -1 if vmin > -1 else vmin   ax.pcolormesh(np.flipud(v), cmap=cmap, vmin=vmin, vmax=vmax)   for y in range(ys):  for x in range(xs):  state = (y, x)  r = self.reward\_map[y, x]  if r != 0 and r is not None:  txt = 'R ' + str(r)  if state == self.goal\_state:  txt = txt + ' (GOAL)'  ax.text(x+.1, ys-y-0.9, txt)   if (v is not None) and state != self.wall\_state:  if print\_value:  offsets = [(0.4, -0.15), (-0.15, -0.3)]  key = 0  if v.shape[0] > 7: key = 1  offset = offsets[key]  ax.text(x+offset[0], ys-y+offset[1], "{:12.2f}".format(v[y, x]))   if policy is not None and state != self.wall\_state:  actions = policy[state]  max\_actions = [kv[0] for kv in actions.items() if kv[1] == max(actions.values())]   arrows = ["↑", "↓", "←", "→"]  offsets = [(0, 0.1), (0, -0.1), (-0.1, 0), (0.1, 0)]  for action in max\_actions:  arrow = arrows[action]  offset = offsets[action]  if state == self.goal\_state:  continue  ax.text(x+0.45+offset[0], ys-y-0.5+offset[1], arrow)   if state == self.wall\_state:  ax.add\_patch(plt.Rectangle((x,ys-y-1), 1, 1, fc=(0.4, 0.4, 0.4, 1.)))  plt.show()   def render\_q(self, q, show\_greedy\_policy=True):  self.set\_figure()   ys, xs = self.ys, self.xs  ax = self.ax  action\_space = [0, 1, 2, 3]   qmax, qmin = max(q.values()), min(q.values())  qmax = max(qmax, abs(qmin))  qmin = -1 \* qmax  qmax = 1 if qmax < 1 else qmax  qmin = -1 if qmin > -1 else qmin    color\_list = ['red', 'white', 'green']  cmap = matplotlib.colors.LinearSegmentedColormap.from\_list(  'colormap\_name', color\_list)   for y in range(ys):  for x in range(xs):  for action in action\_space:  state = (y, x)  r = self.reward\_map[y, x]  if r != 0 and r is not None:  txt = 'R ' + str(r)  if state == self.goal\_state:  txt = txt + ' (GOAL)'  ax.text(x+.05, ys-y-0.95, txt)   if state == self.goal\_state:  continue   tx, ty = x, ys-y-1   action\_map = {  0: ((0.5+tx, 0.5+ty), (tx+1, ty+1), (tx, ty+1)),  1: ((tx, ty), (tx+1, ty), (tx+0.5, ty+0.5)),  2: ((tx, ty), (tx+0.5, ty+0.5), (tx, ty+1)),  3: ((0.5+tx, 0.5+ty), (tx+1, ty), (tx+1, ty+1)),  }  offset\_map = {  0: (0.1, 0.8),  1: (0.1, 0.1),  2: (-0.2, 0.4),  3: (0.4, 0.4),  }  if state == self.wall\_state:  ax.add\_patch(plt.Rectangle((tx, ty), 1, 1, fc=(0.4, 0.4, 0.4, 1.)))  elif state in self.goal\_state:  ax.add\_patch(plt.Rectangle((tx, ty), 1, 1, fc=(0., 1., 0., 1.)))  else:   tq = q[(state, action)]  color\_scale = 0.5 + (tq / qmax) / 2 # normalize: 0.0-1.0   poly = plt.Polygon(action\_map[action],fc=cmap(color\_scale))  ax.add\_patch(poly)   offset= offset\_map[action]  ax.text(tx+offset[0], ty+offset[1], "{:12.2f}".format(tq))  plt.show()   if show\_greedy\_policy:  policy = {}  for y in range(self.ys):  for x in range(self.xs):  state = (y, x)  qs = [q[state, action] for action in range(4)] # action\_size  max\_action = np.argmax(qs)  probs = {0:0.0, 1:0.0, 2:0.0, 3:0.0}  probs[max\_action] = 1  policy[state] = probs  self.render\_v(None, policy) |
| 실행화면 |
|  |

1. Gridworld.py

|  |
| --- |
| 코드 |
| import numpy as np import gridworld\_render as render\_helper   class GridWorld:  def \_\_init\_\_(self):  self.action\_space = [0, 1, 2, 3] # 행동 공간(가능한 행동들)  self.action\_meaning = { # 행동의 의미  0: "UP",  1: "DOWN",  2: "LEFT",  3: "RIGHT",  }   self.reward\_map = np.array( # 보상 맵(각 좌표의 보상 값) 5\*5에 대한 정보  [[0, 0, 0, -1.0, 1.0],  [0, 0, 0, 0, 0],  [0, None, None, 0, 0],  [0, 0, 0, 0, -1.0],  [0, 0, 0, 0, 0]]  )  self.goal\_state = (0, 4) # 목표 상태(좌표)  self.wall\_state = {(2, 1), (2, 2)} # 벽 상태(좌표)  self.start\_state = (4, 0) # 시작 상태(좌표)  self.agent\_state = self.start\_state # 에이전트 초기 상태(좌표)   @property  def height(self):  return len(self.reward\_map)   @property  def width(self):  return len(self.reward\_map[0])   @property  def shape(self):  return self.reward\_map.shape   def actions(self):  return self.action\_space   def states(self):  for h in range(self.height):  for w in range(self.width):  yield (h, w)   def next\_state(self, state, action):  # 이동 위치 계산  action\_move\_map = [(-1, 0), (1, 0), (0, -1), (0, 1)]  move = action\_move\_map[action]  next\_state = (state[0] + move[0], state[1] + move[1])  ny, nx = next\_state   # 이동한 위치가 그리드 월드의 테두리 밖이나 벽인가?  if nx < 0 or nx >= self.width or ny < 0 or ny >= self.height:  next\_state = state  elif next\_state == self.wall\_state:  next\_state = state   return next\_state # 다음 상태 반환   def reward(self, state, action, next\_state):  r = self.reward\_map[next\_state]  return 0.0 if r is None else r   def reset(self):  self.agent\_state = self.start\_state  return self.agent\_state   def step(self, action):  state = self.agent\_state  next\_state = self.next\_state(state, action)  reward = self.reward(state, action, next\_state)  done = (next\_state == self.goal\_state)   self.agent\_state = next\_state  return next\_state, reward, done   def is\_wall(self, state):  return self.reward\_map[state] is None   def render\_v(self, v=None, policy=None, print\_value=True):  renderer = render\_helper.Renderer(self.reward\_map, self.goal\_state,  self.wall\_state)  renderer.render\_v(v, policy, print\_value)   def render\_q(self, q=None, print\_value=True):  renderer = render\_helper.Renderer(self.reward\_map, self.goal\_state,  self.wall\_state)  renderer.render\_q(q, print\_value) |
| 실행화면 |
|  |

1. policeyEval.py

|  |
| --- |
| 코드 |
| from collections import defaultdict from gridworld import GridWorld #from homework.gridworld import GridWorld  def eval\_onestep(pi, V, env, gamma = 0.9):  for state in env.states():  if state == env.goal\_state:  V[state] = 0  continue   if env.reward\_map[state] is None:  continue   action\_probs = pi[state]  new\_V = 0   for action,action\_prob in action\_probs.items():  next\_state = env.next\_state(state, action)  r = env.reward(state, action, next\_state)   if r is None:  r = 0.0   new\_V += action\_prob \* ( r+ gamma \* V[next\_state])    V[state] = new\_V  return V   def policyEval(pi, V, env, gamma, threshold = 0.001):  while True:  old\_V = V.copy()  V = eval\_onestep(pi, V, env, gamma)    delta = 0  for state in V.keys():  t = abs(V[state] - old\_V[state])  if delta < t:  delta = t    if delta < threshold:  break  return V   if \_\_name\_\_ == '\_\_main\_\_':  env = GridWorld()  gamma = 0.9    pi = defaultdict(lambda: {0: 0.25, 1:0.25, 2:0.25, 3:0.25})  V = defaultdict(lambda: 0)    V = policyEval(pi, V, env, gamma)    env.render\_v(V, pi) |
| 실행화면 |
|  |

1. policyIter.py

|  |
| --- |
| 코드 |
| from collections import defaultdict from gridworld import GridWorld from policyEval import policyEval   def argmax(d):  *"""d (dict)"""* max\_value = max(d.values())  max\_key = -1  for key, value in d.items():  if value == max\_value:  max\_key= key      return max\_key   def greedyPolicy(V, env, gamma):  pi = {}    for state in env.states():  if env.is\_wall(state):  continue   action\_values = {}    for action in env.actions():  next\_state = env.next\_state(state, action)  r = env.reward(state, action, next\_state)  value = r + gamma \* V[next\_state]  action\_values[action] = value    max\_action = argmax(action\_values)  action\_probs = {0: 0, 1: 0, 2: 0, 3:0}  action\_probs[max\_action] = 1.0  pi[state] = action\_probs  return pi   def policyIter(env, gamma, threshold=0.001, is\_render = True):  pi = defaultdict(lambda: {0: 0.25, 1: 0.25, 2: 0.25, 3: 0.25})  V = defaultdict(lambda: 0)    while True:  V = policyEval(pi, V, env, gamma, threshold)  new\_pi = greedyPolicy(V, env, gamma)    if is\_render:  env.render\_v(V, pi)    if new\_pi == pi:  break  pi = new\_pi  return pi      if \_\_name\_\_ == '\_\_main\_\_':  env = GridWorld()  gamma = 0.9    pi = policyIter(env, gamma) |
| 실행화면 |
|  |

1. vaslueIter.py

|  |
| --- |
| 코드 |
| from collections import defaultdict from gridworld import GridWorld from policyIter import greedyPolicy, policyIter   def value\_iter\_onestep(V, env, gamma):  for state in env.states():  if state == env.goal\_state:  V[state] = 0  continue    action\_values = []  for action in env.actions():  next\_state = env.next\_state(state, action)  r = env.reward(state, action, next\_state)  value = r + gamma \* V[next\_state]  action\_values.append(value)    V[state] = max(action\_values)  return V   def value\_iter(V, env, gamma, threshold = 0.001, is\_render = True):  while True:  if is\_render:  env.render\_v(V)  old\_V = V.copy()  V = value\_iter\_onestep(V, env, gamma)    delta = 0    for state in V.keys():  t = abs(V[state] - old\_V[state])  if delta < t:  delta = t    if delta < threshold:  break  return V      if \_\_name\_\_ == '\_\_main\_\_':  V = defaultdict(lambda: 0)  env = GridWorld()  gamma = 0.9    V = value\_iter(V, env, gamma)    pi = policyIter(env, gamma)  env.render\_v(V, pi) |
| 실행화면 |
|  |