دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

گزارش تمرین شماره 3
گروه ...
درس یادگیری تعاملی
پاییز 1400

نام و نام خانوادگی	مهسا تاجیک
شماره دانشجویی	810198126

فهرست

چکیده

در این تمرین در بخش اول، به مدل کردن چند مسئله ی MDP بصورت تئوری پرداخته میشود و در بخش پیاده سازی یک مسئله ی ماز را بصورت MDP مدل کرده به پیاده سازی الگوریتم های policy iteration و value iteration میپردازیم.

سوال 1 - سوال تئوری

زیر بخش 1

در این مساله ما باید تصمیم بگیریم که چه مقداری از خرچنگ ها را در یک ماه در دهکده صید کنیم تا بازده طولانی مدت را به حداکثر برسانیم .هر خرچنگ مقدار ثابتی دلار تولید می کند .اما اگر بخش زیادی از خرچنگ ها صید شود، عملکرد ماه آینده کمتر خواهد بود. ما باید مقدار بهینه صید و فروش خرچنگ را برای بیابیم تا بازده آن را در یک دوره زمانی طولانی به حداکثر برسانیم.

States:

تعداد خرچنگ های موجود در آن دهکده در آن ماه. فرض میکنیم فقط چهار حالت وجود دارد. خالی، کم، متوسط، زیاد. چهار حالت به صورت زیر تعریف می شوند:

خالی ← خرچنگ موجود نیست.

کم ← تعداد موجود خرچنگ کمتر از یک ترشلد t1 است.

متوسط ← تعداد موجود خرچنگ بین t1 و t2 است.

بالا ← تعداد موجود خرچنگ بیش از t2 است.

Actions:

برای سادگی فرض میکنیم که فقط دو عمل وجود دارد. فروش مقدار خاصی از خرچنگ و نفروختن خرچنگ. برای حالت خالی تنها اقدام ممکن نفروختن است .

Rewards:

فرض میکنیم پاداش فروش خرچنگ در حالت کم، متوسط و بالا به ترتیب هزار دلار، 10 هزار دلار و 100 هزار دلار است. اگر اقدامی به حالت خالی برسد، پاداش آن بسیار کم است و برابر منفی 200 هزار دلار، زیرا نیاز به جبران جمعیت خرچنگ ها هستیم که به زمان و هزینه نیاز دارد.

State Transitions:

عمل نفروختن خرچنگ احتمال بالای دارد که به حالت هایی با تعداد خرچنگ بیشتر برود بجز زمانی که در حالتی با بیشترین خرچنگ یعنی حالت چهارم هستیم.

زیر بخش 2

States:

هر کدام از 7 مرحله در یک دوره یک حالت را برای ما ایجاد میکند. پس در مجموع 7 حالت داریم.

Actions:

در هر کدام از حالت ها می توانیم سهام را نگه داریم یا بفروشیم. پس دو عمل خواهیم داشت: نگه داشتن و فروختن سهام.

Rewards:

در هر مرحله ارزش سهام را نگاه می کنیم اگر نسبت به مرحله ی قبل ارزش سهام افزایش پیدا کرده بود و اکشن ما نگه داشتن سهام بود، یک پاداش مثبت درنظر میگیریم. اگر نسبت به مرحله ی قبل ارزش سهام افزایش داشت و ما آن را فروخته بودیم، مقداری پاداش منفی در نظر میگیریم. اگر ارزش سهام نسبت به مرحله ی قبل کاهش داشت و ما فروختیم مقداری پاداش مثبت و اگر ارزش سهام نسبت به مرحله قبل کاهش داشت و ما فروختیم مقداری پاداش مثبت در نظر میگیریم. اگر در پایان دوره در مجموع با 7 تصمیم قبلی که برای نگه داشتن یا فروش سهام گرفته بودیم در سود بودیم پاداش مثبتی 100 برابر پاداش های میان دوره ای دریافت می کنیم وگرنه پاداش منفی با همین اندازه دریافت خواهیم کرد.

State Transitions:

سهام را نگه داشتن در مرحله ی iام و نگه داشتن/ فروش سهام در مرحله ی jام یک ترنزیشن بین مرحله ی i و j با انجام عمل نگه داشتن/ فروش سهام می باشد.

زیر بخش 3

States:

تقاضای سال قبل استیت ما را میسازد.

Actions:

تصمیم گیری برای میزان تولید سال آینده

Rewards:

پاداش منفی ای درنظر گرفته میشود برای زمانیکه مشتری محصولی را میخواهد و موجود نیست، که این پاداش را میتوانیم مقدار زیادی درنظر بگیریم چون مشتری را هم از دست میدهیم و به ازای هر درخواست مشتری که پاسخ داده میشود پاداش مثبت خیلی کوچکی در نظر میگیریم.

State Transitions:

دوباره به همان استیت بر میگردیم که برای سال بعدی هم از روی تقاضای امسال، برای میزان تولید تصمیم گیری کنیم

زیر بخش 4

States:

می توانیم حالت ها را با سطح بندی صدای آژیر و همچنین جمعیت منطقه بسازیم:

حالت اول: صدای آژیری به گوش نرسد.

حالت دوم: صدای آژیر کم باشد و منطقه کم جمعیت باشد.

حالت سوم: صدای آژیر کم باشد و جمعیت منطقه متوسط باشد.

حالت چهارم: صدای آژیر کم باشد و منطقه پر جمعیت باشد.

حالت پنجم: صدای آژیر متوسط باشد و منطقه کم جمعیت باشد.

حالت ششم: صدای آژیر متوسط باشد و جمعیت منطقه متوسط باشد.

حالت هفتم: صدای آژیر متوسط باشد و منطقه پر جمعیت باشد.

حالت هشتم: صدای آژیر زیاد باشد و منطقه کم جمعیت باشد.

حالت نهم: صدای آژیر زیاد باشد و جمعیت منطقه متوسط باشد.

حالت دهم: صدای آژیر زیاد باشد و منطقه پر جمعیت باشد.

Actions:

اگر در حالت اول باشیم ماشینی فرستاده نمیشود. اگر درحالت دوم باشیم، تعداد ماشین هایی که فرستاده می شود کمتر از ترشلد t1 است. در حالت سوم تعداد ماشین هایی که فرستاده میشود بین t2,t1 ، در حالت های چهارم و پنجم تعداد ماشین ها بین t3,t2 ، در حالت ششم بین t4,t3 ، در حالت های هفتم و هشتم بین t5,t4 ، در حالت نهم بین t6,t5 و در حالت دهم بزرگتر از t6 است.

Rewards:

در هر منطقه به نسبت جمعیت آن منطقه تعداد مجروحین و کشته شدگان را درنظر میگیریم اگر این میزان صفر بود پاداش زیادی مثلا 1500 واحد درنظر میگیریم. اگر از یک مقدارt1 کمتر بود، 100 پاداش منفی و اگر بیشتر از این مقدار بود 1200 امتیاز منفی دریافت می کند. اگر آژیر زده شود و ماشینی در آتش نشانی نباشد 3000 پاداش منفی دریافت میکند.

State Transitions:

ترنزیشن بین استیت ها نداریم.

سوال 2 – سوال پیاده‌سازی

هدف سوال

هدف سوال پیاده سازی یک مساله MDP با استفاده از الگوریتم های policy iteration و value iteration ، به منظور مسیریابی ربات از نقطه ی شروع به هدف است تا سیاست بهینه را پیدا کنیم.

محیط مساله یک گرید 15*15 است. ایجنت در خانه ی (0و0) در نظر گرفته شده است و باید به خان ی هدف که در (14و14) است ،برسد. از خانه ی فعلی میتواند 8 حرکت انجام دهد و به یکی از خانه های اطراف برود و یا در جای خود بماند. بنابراین در مجموع 9 اکشن برای ایجنت بصورت زیر تعریف شده است:

```
list_of_actions = [[0,0], [0,1], [0,-1], [1,0], [-1,0],[-1,1], [-1,-1], [1,-1], [1,1]]
```

هر یک از خانه های گرید یک استیت را برای ایجنت میسازد.

در هربار تغییر استیت ممکن است بعضی اکشن ها ما را به استیت ناممکن و غیرقابل دسترس یعنی مانع ها و بیرون از گرید ببرند بنابراین باید از رفتن به آن ها اجتناب شود.

در هر بار تصمیم گیری برای تغییر استیت ممکن است ایجنت با احتمال 0.8 به استیت مورد نظر خود برود و با احتمال 0.2 به خانه های اطراف لیز بخورد.

هر حرکت ایجنت و هر برخورد با مانع پاداش منفی خواهد داشت و رسیدن به هدف پاداش مثبت بزرگی خواهد داشت.

Policy iteration :

می دانیم که policy به این معناست که ما در هر state چه Action ای را انجام دهیم action های متفاوتی در هر استیت قابل انجام است و ما باید return term-long هر عمل را بررسی کنیم.

از انجایی که سیاست بهینه یک سیاست deterministic است ، ما هم در بین سیاست های deterministic به دنبال policy optimal میگردیم . یعنی در هر state فقط یک action را انجام خواهیم داد . در ابتدا عمل evaluation policy را انجام میدهیم . به این صورت که یک policy را در ابتدا میگیرد و با توجه به این policy خاص و return را برای همه ی تک state ها حساب می کند . با این پروسه ما ارزش هر state را تحت این سیاست پیدا میکنیم سپس عمل improvement policy را انجام میدهیم . یعنی تمام action هایی که در یک state را میتوانیم انجام دهیم بررسی کنیم و با توجه به value ما

greedyانتخاب کنیم تا بیشترین return را داشته باشیم . این عمل evaluation policy و improvement policyرا در یک حلقه آنقدر تکرار میکنیم تا improvementای دیگر نتوان انجام داد.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

اینکه ما تا چه میزان میتوانیم improvement انجام دهیم ، بستگی به مقدار تتا در policy evaluation دارد. در این مساله ما مقدار تتا را 0.001 در نظر گرفتیم.

$$V_{\pi^*}(s) = \max_a \sum_{s',r} P(r,s'|s,a)[r + \gamma\, V_{\pi^*}(s')]$$

ماتریس policy ما یک ماتریس 15*15 است که هر خانه ی آن یکی از استیت ها محیط است و مشخص میکند اکشن بهینه در آن استیت چه خواهد بود.

:Value iteration

در الگوریتم value iteration قسمتی که مقدار V(s) جدید را محاسبه می کنیم مقدار بیشینه را بین تمام اکشن ها بعنوان مقدار جدید در نظر میگیریم و policy هم ارگومان اکشنی است که بیشینه مقدار را دارد.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$

توضیح پیاده سازی

در کلاس **Environment** 5 تابع پیاده سازی شده اند که در ادامه در مورد آنها توضیح داده میشود:

```
def isStatePossible(self, state):
  for i in range(len(self.obstacle)):
    if state == self.obstacle[i]:
      return False
    elif not ((-1 < state[0] < 15) and (-1 < state[1] <15)):
      return False
    else:
      return True
```

تابع isStatePossible چک می کند که آیا یک استیت جز موانع هست یا نه و همچنین داخل نقشه 15*15 تایی که تعریف کردیم قرار میگیرد یا نه ، اگر استیتی بود که مانع بود یا خارج از این محدوده قرار داشت False وگرنه True برمیگرداند.

```
def isAccessible(self, state, state_p):

  for action in list_of_actions:
    if [sum(x) for x in zip(state, action)] == state_p:
      if state_p not in self.obstacle:
        return True
      else:
        return False
    else:
      return False
```

تابع isAccessible چک میکند که آیا از استیتی که در آن هستیم با انجام عمل action داده شده، امکان دسترسی به استیت state_p وجود دارد یا خیر یعنی جز همسایه های استیت فعلی باشد و مانع نباشد. اگر این شرایط برقرار بود True در غیراینصورت False برمیگرداند.

```
def getTransitionStatesAndProbs(self, state, action, state_p):

  prob = np.zeros(9)
  action_index = list_of_actions.index(action)
  if action_index in self.available_actions(state):
    n = len(self.available_actions(state))-1
    if n !=0:
      prob[action_index] = self.p
      for i in self.available_actions(state):
        if i != action_index:
      # print(self.available_actions(state))
          prob[i] = (1-self.p)/n
      prob_stay=(1-self.p)/n
    else:
      prob[action_index] = self.p
      prob[0] = 1-self.p
      prob_stay = 1-self.p
  else:
    prob[0] = self.p
    n = len(self.available_actions(state))
    if n !=0:
      prob[action_index] = 0
      for i in self.available_actions(state):
        if i != action_index:
      # print(self.available_actions(state))
          prob[i] = (1-self.p)/n
    prob_stay = (1-self.p)/n
  return prob,prob_stay
```

تابع getTransitionStateAndProbs احتمال اینکه در state عمل action را انجام داده و به state_p برویم را محاسبه میکند. دو حالت در نظر میگیریم:

اگر اکشنی که می خواهیم انجام دهیم ، دردسترس و ممکن باشد:

برای این حالت دو حالت ممکن است پیش بیاید: اکشن های مجاز ماندن در خانه ی فعلی و رفتن به استیت state_p باشد و یا بیشتر از این تعداد باشد :

حالت 1: در این حالت، با احتمال 0.8 به استیت موردنظر میرویم و به احتمال 0.2 در استیت فعلی میمانیم.

حالت 2: به احتمال p = 0.8 به state_p منتقل میشویم و احتمال 1-p = 0.2 را باید به تعداد اکشن های ممکن و دردسترس دیگر از این استیت، تقسیم کنیم که این تعداد یکی کمتر از تعداد کل

اکشن های دردسترس و ممکن است(یکی همان اکشنی است که ما را به state_p میبرد که کم می کنیم) است.

اگر اکشنی که می خواهیم انجام دهیم ، دردسترس و ممکن نباشد:

احتمال رفتن به استیت مورد نظر صفر و ماندن در خانه ی فعلی 0.8 است و 0.2 بین اکشن های مجاز دیگر تقسیم میشود.

```
def getReward(self, state, action, state_p):

  if (state_p[0] == state[0]) and (state_p[1] == state[1]):
    return self.actionPrice
  elif [state_p[0],state_p[1]] in self.obstacle:
    return self.actionPrice + self.punish
  elif (not([state_p[0],state_p[1]] in self.obstacle)) and ((state_p[0] != goal[0]) or state_p[1] != goal[1]):
    return self.actionPrice
  elif (state_p[0] == goal[0]) and (state_p[1] == goal[1]):
    return self.goalReward + self.actionPrice
```

تابع getReward پاداش هر ترنزیشن را برمیگرداند. اگر در استیت فعلی بمانیم فقط actionPrice دریافت می کنیم، اگر استیت بعدی مانع باشد actionPrice + punish را دریافت میکنیم، اگر استیتی که میرویم مانع و goal نباشد، actionPrice را دریافت میکنیم و اگر به goal برویم، actionPrice + goalReward دریافت خواهیم کرد.

```
def available_actions(self, state):
  available__actions = []
  available__actions = list_of_actions.copy()
  for action in list_of_actions:
    if [[sum(x) for x in zip(state, action)][0],[sum(x) for x in zip(state, action)][1]] in self.obstacle:
      available__actions.remove(action)
    if ([sum(x) for x in zip(state, action)][0] < 0) or \
      ([sum(x) for x in zip(state, action)][1] < 0) or \
      ([sum(x) for x in zip(state, action)][0] > 14) or \
      ([sum(x) for x in zip(state, action)][1] > 14):
      available__actions.remove(action)
```

در تابع available_actions در لیست تمام اکشن ها بررسی میشود که از استیت فعلی چه اکشن هایی قابل انجام هستند و با انجام آنها به مانع نمیخوریم و خارج از نقشه نمی افتد و سپس اندیس اکشن های قابل انجام برگردانده میشود.

در کلاس **Agent** توابع زیر پیاده سازی شده اند که در ادامه در مورد آنها توضیح داده میشود:

```
def policy_evaluation(self):
  print('policy evaluation ...')
  while True:
    old_value = self.V.copy()
    for i in range(grid_size):
      for j in range(grid_size):
        new_state_value = self.bellman_equation_action2([i,j], self.policy[i,j])
        self.V[i, j] = new_state_value
    max_value_change = abs(old_value - self.V).max()
    if max_value_change < self.theta:
      break
```

در تابع policy_evaluation برای تمام استیت ها مقدار جدید با توجه به معادله بلمن محاسبه میشود و مقدار جدید با مقدار قبلی مقایسه میشود. اگر بیشترین مقدار تغییر در بین مقادیر استیت های مختلف، از یک مقدار theta که همان دقت ما را مشخص میکند،کمتر بود این تابع متوقف میشود.

```
def bellman_equation_action2(self, state, action):
  returns = 0
  if action in self.environment.available_actions(state):

      next_state = [sum(x) for x in zip(state, list_of_actions[action])]
      probs,_ = self.environment.getTransitionStatesAndProbs(state, list_of_actions[action], next_state)

      reward = self.environment.getReward(state, action, next_state)
      returns += 1 * probs[action] * (reward + self.discount * self.V[int(next_state[0]),int(next_state[1])])

  else:
      next_state = state
      probs,prob_stay= self.environment.getTransitionStatesAndProbs(state, list_of_actions[action], next_state)
      next_state = obstacle[action]
      reward = self.environment.getReward(state, action, next_state)
      next_state = state
      returns += 1 * prob_stay * (reward + self.discount * self.V[int(next_state[0]),int(next_state[1])])

  return returns
```

تابع بلمن استفاده شده در policy_evaluation در زیر آورده شده : دو حالت در نظر گرفته شده که استیت بعدی در دسترس وممکن باشد یا نباشد: برای هر دوحالت باید استیت بعدی را مشخص کنیم (که در حالت دوم ایجنت در خانه فعلی میماند). سپس با استفاده از تابع getTransitionStateAndProbs احتمال ترنزیشن را محاسبه میکنیم و با استفاده از تابع getReward پاداش این ترنزیشن را بدست آورده و در رابطه ی بلمن جایگذاری میکنیم تا مقدار return را بدست آوریم.

```
def policy_improvement(self):
  print('policy improvement ...')
  self.policy_stable = True
  for i in range(grid_size):
    for j in range(grid_size):
      old_action = self.policy[i, j]
      action_returns = []
      for action in list_of_actions:
        # if action in self.environment.available_actions([i,j]):
          action_returns.append(self.bellman_equation_action([i,j], list_of_actions.index(action)))
        # else:
          # action_returns.append(-np.inf)
      # print("action_returns",action_returns)
      new_action = list_of_actions[np.argmax(action_returns)]
      print("new_action",new_action)
      print("action_index",list_of_actions.index(new_action))

      for p in range(len(list_of_actions)):
        if list_of_actions[p] == new_action:
          new_action_index = p

      self.policy[i, j] = new_action_index
      print(self.policy)
      if self.policy_stable and old_action != new_action_index:
        self.policy_stable = False
  print(' *policy_stable: {}*'.format(self.policy_stable))
  return self.policy_stable,self.policy
```

در تابع policy_improvement، همانطور که از اسم آن مشخص است پالیسی بهبود پیدا میکند. برای تمام استیت ها، مقادیری که با انجام هر اکشن بدست می آید با رابطه بلمن محاسبه کرده و ذخیره میکنیم و اکشنی را که بیشترین مقدار را برای ما دارد بعنوان پالیسی بعدی انتخاب می کنیم. اینکار را تا جایی ادامه می دهیم که پالیسی در دو ایتریشن متوالی دیگر تغییری نکند.

```
def policy_iteration(self):
  while True:
    # print(self.policy)
    self.policy_evaluation()
    a,b=self.policy_improvement()
    self.policy = b
    if self.policy_stable:
        break
        print(self.policy)
  print('Done!')
```

در تابع policy_iteration تا زمانیکه فلگ policy_stable مقدارش True نشده یعنی پالیسی قبلی و فعلی یکی نشده ادامه پیدا میکند و توابع policy_evaluation و policy_improvement برای مقداردهی و بهبود پالیسی فراخوانی میشوند.

```
def value_iteration(self):
  print('value iteration ...')
  while True:
    old_value = self.V.copy()
    for i in range(15):
      for j in range(15):
        new_state_value,p = self.bellman_equation2([i,j], self.policy[i,j])
        self.V[i, j] = new_state_value
        self.policy = p
    max_value_change = abs(old_value - self.V).max()
    if max_value_change < self.theta:
      break

  return self.V[i, j], self.policy
```

در تابع value_iteration نحوه محاسبه پالیسی و value در خط مربوط به معادله بلمن متفاوت است که برای آن تابع جداگانه ای نوشته شده است. مقدار value ، بیشینه ی مقادیر بین همه ی اکشن ها در یک استیت است و پالیسی ایندکس اکشنی با بیشترین مقدار است.

مقداردهی اولیه به پارامترها، تعریف اکشن ها ، موانع و goal state و همچنین ایجاد یک ابجکت از محیط و ایجنت و ساخت پالیسی رندوم اولیه :

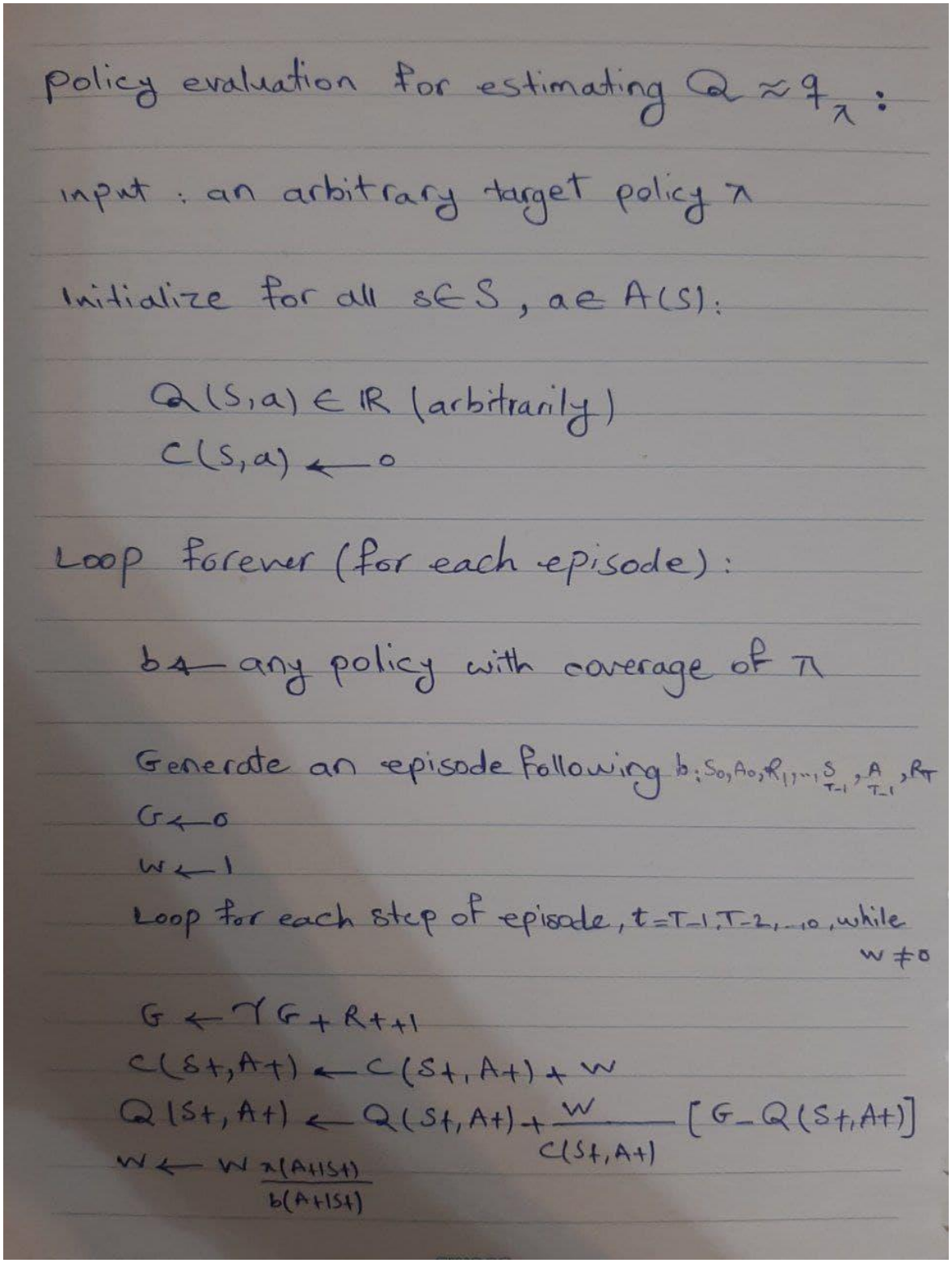
```
grid_size = 15
action_size = 9
list_of_actions = [[0,0], [0,1], [0,-1], [1,0], [-1,0],[-1,1], [-1,-1], [1,-1], [1,1]]
obstacle = [[0,6],[0,7],[1,6],[1,7],[2,6],[2,7],[3,6],[3,7],[7,12],[7,13],[7,14],[8,12] \
           ,[8,13],[8,14],[11,5],[11,6],[12,5],[12,6],[13,5],[13,6],[14,5],[14,6]]
goal = [14, 14]
```

```
env = Environment(obstacle = obstacle, id='1',action_count=9, actionPrice = -0.01, goalReward = 1000, punish=-1,
id = '1'
theta = 0.0001
discount = 0.9
value = np.zeros((grid_size,grid_size))
policy = np.zeros(value.shape, dtype=np.int)
# policy = np.eye(grid_size, dtype=np.int)

for r in range(grid_size):
  policy[r,:] = random.randint(0, 8)

agent1 = Agent(id, env, discount, theta, value, policy, map)
```

نتایج

زیر بخش 1

شبه کد خواسته شده برای بدست آوردن ارزش استیت های محیط در روش مونت کارلو:



زیر بخش 2

در این قسمت سیاست بهینه برای حالت پایه خواسته شده است. پالیسی اولیه بصورت رندوم داده شده است:

```
 *policy_stable: True*
[[8 3 3 3 3 3 7 8 8 3 3 3 3 3 3]
 [1 8 3 3 3 3 7 8 3 3 3 3 3 3 3]
 [1 1 8 3 3 3 7 8 3 3 3 3 3 3 3]
 [1 1 8 8 3 3 3 3 3 3 3 3 3 3 3]
 [1 1 5 1 8 8 3 3 7 3 3 3 3 3 7]
 [1 1 5 1 1 8 3 3 3 3 3 3 3 7 7]
 [1 1 5 1 1 1 8 3 3 3 3 3 7 2 2]
 [1 1 5 1 1 1 1 8 3 3 3 3 7 6 6]
 [1 1 5 1 1 1 1 1 8 3 3 3 3 3 3]
 [1 1 5 1 1 1 1 1 1 8 3 3 7 8 7]
 [1 1 5 1 5 1 8 5 1 1 8 8 7 7 3]
 [1 1 5 1 5 5 1 5 1 1 8 8 3 3 3]
 [1 1 5 5 4 6 5 1 1 1 8 1 8 8 3]
 [1 1 5 4 4 6 5 1 1 1 8 1 8 8 3]
 [1 5 4 4 6 6 1 1 1 1 1 1 1 1 4]]
Done!
None
```

تعریف اکشن ها بصورت زیر بود:

```
list_of_actions = [[0,0], [0,1], [0,-1], [1,0], [-1,0],[-1,1], [-1,-1], [1,-1], [1,1]]
```

همانطور که انتظار میرود، پالیسی نشان میدهد که در مسیر مورب تماما اکشن 8 انتخاب شده که همان [1,1] است(یک سطر و یک ستون به جلو) و منطقی است که برای اینکه پاداش بیشتری بدست آورد این مسیر هزینه ی حرکتی کمتری برایش دارد. و همچنین مقدار value در خانه هایی که موانع هستند به نسبت خانه هایی که روی قطر ماتریس و اطراف آن قرار میگیرند کمتر است:

```
[[ 23.1841448   23.1841448   23.1841448   23.1841448   23.1841448
   23.1841448   23.1841448   23.1841448   23.1841448   23.1841448
   23.1841448   23.1841448   23.1841448   23.1841448   23.1841448 ]
 [ 23.1841448   32.21131223  32.21131223  32.21131223  32.21131223
   32.21131223  32.21131223  32.21131223  32.21131223  32.21131223
   32.21131223  32.21131223  32.21131223  32.21131223  32.21131223]
 [ 23.1841448   32.21131223  44.74904476  44.74904476  44.74904476
   44.74904476  44.74904476  44.74904476  44.74904476  44.74904476
   44.74904476  44.74904476  44.74904476  44.74904476  44.74904476]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  62.16256217
   62.16256217  62.16256217  62.16256217  62.16256217  62.16256217
   62.16256217  62.16256217  62.16256217  62.16256217  62.16256217]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  86.34800301
   86.34800301  86.34800301  86.34800301  86.34800301  86.34800301
   86.34800301  86.34800301  86.34800301  86.34800301  86.34800301]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  86.34800301
  119.93889307 119.93889307 119.93889307 119.93889307 119.93889307
  119.93889307 119.93889307 119.93889307 119.93889307  86.34800301]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  86.34800301
  119.93889307 166.59290705 166.59290705 166.59290705 166.59290705
  166.59290705 166.59290705 166.59290705 119.93889307  86.34800301]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  86.34800301
  119.93889307 166.59290705 231.39014868 231.39014868 231.39014868
  231.39014868 231.39014868 231.39014868 119.93889307  86.34800301]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  86.34800301
  119.93889307 166.59290705 231.39014868 321.38631761 321.38631761
  321.38631761 321.38631761 321.38631761 321.38631761 321.38631761]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  86.34800301
  119.93889307 166.59290705 231.39014868 321.38631761 446.38099668
  446.38099668 446.38099668 446.38099668 446.38099668 446.38099668]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  86.34800301
  119.93889307 166.59290705 231.39014868 321.38631761 446.38099668
  619.98471761 619.98471761 619.98471761 619.98471761 619.98471761]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  86.34800301
  119.93889307 166.59290705 231.39014868 321.38631761 446.38099668
  619.98471761 861.10099668 861.10099668 861.10099668 861.10099668]
 [ 23.1841448   32.21131223  44.74904476  62.16256217  62.16256217
   62.16256217 166.59290705 231.39014868 321.38631761 446.38099668
  619.98471761 861.10099668 1195.98471761 1195.98471761 1195.98471761]
 [ 23.1841448   32.21131223  44.74904476  44.74904476  44.74904476
   44.74904476 166.59290705 231.39014868 321.38631761 446.38099668
  619.98471761 861.10099668 1195.98471761 1661.10099668 1661.10099668]
 [ 23.1841448   32.21131223  32.21131223  32.21131223  32.21131223
   32.21131223 166.59290705 231.39014868 321.38631761 446.38099668
  619.98471761 861.10099668 1195.98471761 1661.10099668 1195.98471761]]
```

زیر بخش 3

در حالت بدون اصطکاک که حرکت عامل هزینه ای برایش ندارد و تنها برخورد با مانع برای او هزینه ی منفی دارد، پالیسی بصورت زیر است و مقدر value هم در ادامه آمده که تفاوت کمی با حالت قبل دارد ولی در خانه ی هدف و چندین خانه ی دیگر اندکی بیشتر است که منطقی است زیرا در این حالت ، حرکت بدون هزینه است.

```
 *policy_stable: True*
[[8 8 7 7 7 3 7 8 3 7 7 7 7 7 3]
 [8 8 3 3 3 3 7 8 8 3 3 3 3 3 3]
 [5 1 8 3 3 3 7 8 3 3 3 3 3 3 3]
 [5 1 1 8 3 3 3 3 3 3 3 3 3 3 3]
 [5 1 1 1 8 3 7 7 7 3 3 3 3 3 7]
 [5 1 1 1 1 8 3 3 3 3 3 3 3 7 2]
 [5 1 1 1 1 1 8 3 3 3 3 3 7 2 2]
 [5 1 1 1 1 1 1 8 3 3 3 3 7 6 6]
 [5 1 1 1 1 1 1 1 8 3 3 7 3 3 3]
 [5 1 1 1 1 1 1 1 1 8 3 8 3 7 3]
 [5 1 1 1 1 1 1 5 1 1 8 3 3 3 3]
 [5 1 1 1 5 5 8 1 1 1 1 8 3 3 3]
 [5 1 1 5 4 6 1 1 1 1 1 1 8 8 3]
 [5 1 5 4 4 6 1 1 1 1 1 1 8 8 3]
 [1 5 4 4 4 6 1 1 1 5 1 5 1 1 2]]
Done!
None
```

```
[[  23.21271623   23.21271623   23.21271623   23.21271623   23.21271623
    23.21271623   23.21271623   23.21271623   23.21271623   23.21271623
    23.21271623   23.21271623   23.21271623   23.21271623   23.21271623]
 [  23.21271623   32.23988366   32.23988366   32.23988366   32.23988366
    32.23988366   32.23988366   32.23988366   32.23988366   32.23988366
    32.23988366   32.23988366   32.23988366   32.23988366   32.23988366]
 [  23.21271623   32.23988366   44.77761619   44.77761619   44.77761619
    44.77761619   44.77761619   44.77761619   44.77761619   44.77761619
    44.77761619   44.77761619   44.77761619   44.77761619   44.77761619]
 [  23.21271623   32.23988366   44.77761619   62.1911336    62.1911336
    62.1911336    62.1911336    62.1911336    62.1911336    62.1911336
    62.1911336    62.1911336    62.1911336    62.1911336    62.1911336 ]
 [  23.21271623   32.23988366   44.77761619   62.1911336    86.37657444
    86.37657444   86.37657444   86.37657444   86.37657444   86.37657444
    86.37657444   86.37657444   86.37657444   86.37657444   86.37657444]
 [  23.21271623   32.23988366   44.77761619   62.1911336    86.37657444
   119.9674645   119.9674645   119.9674645   119.9674645   119.9674645
   119.9674645   119.9674645   119.9674645   119.9674645    86.37657444]
 [  23.21271623   32.23988366   44.77761619   62.1911336    86.37657444
   119.9674645   166.62147848  166.62147848  166.62147848  166.62147848
   166.62147848  166.62147848  166.62147848  119.9674645    86.37657444]
 [  23.21271623   32.23988366   44.77761619   62.1911336    86.37657444
   119.9674645   166.62147848  231.41872011  231.41872011  231.41872011
   231.41872011  231.41872011  231.41872011  119.9674645    86.37657444]
 [  23.21271623   32.23988366   44.77761619   62.1911336    86.37657444
   119.9674645   166.62147848  231.41872011  321.41488904  321.41488904
   321.41488904  321.41488904  321.41488904  321.41488904  321.41488904]
 [  23.21271623   32.23988366   44.77761619   62.1911336    86.37657444
   119.9674645   166.62147848  231.41872011  321.41488904  446.40956811
   446.40956811  446.40956811  446.40956811  446.40956811  446.40956811]
 [  23.21271623   32.23988366   44.77761619   62.1911336    86.37657444
   119.9674645   166.62147848  231.41872011  321.41488904  446.40956811
   620.01328904  620.01328904  620.01328904  620.01328904  620.01328904]
 [  23.21271623   32.23988366   44.77761619   62.1911336    86.37657444
   119.9674645   166.62147848  231.41872011  321.41488904  446.40956811
   620.01328904  861.12956811  861.12956811  861.12956811  861.12956811]
 [  23.21271623   32.23988366   44.77761619   62.1911336    62.1911336
    62.1911336   166.62147848  231.41872011  321.41488904  446.40956811
   620.01328904  861.12956811 1196.01328904 1196.01328904 1196.01328904]
 [  23.21271623   32.23988366   44.77761619   44.77761619   44.77761619
    44.77761619  166.62147848  231.41872011  321.41488904  446.40956811
   620.01328904  861.12956811 1196.01328904 1661.12956811 1661.12956811]
 [  23.21271623   32.23988366   32.23988366   32.23988366   32.23988366
    32.23988366  166.62147848  231.41872011  321.41488904  446.40956811
   620.01328904  861.12956811 1196.01328904 1661.12956811 1196.01328904]]
 *policy_stable: True*
```

زیر بخش 4

برای این حالت در تابع getReward حالت اولی که تعریف کرده بودیم که پاداش ماندن در خانه ی فعلی برابر با actionPrice بود ، صفر قرار میدهیم و هزینه ی برخورد با مانع برابر با منفی 10، هزینه ی

هر حرکت منفی 1 و پاداش رسیده به خانه ی هدف 100 درنظر گرفته میشود و پالیسی بصورت زیر خواهد:

```
  *policy_stable: True*
 [[2 4 4 4 4 1 0 0 2 4 4 4 4 4 1]
  [2 8 8 7 3 3 0 0 3 3 7 7 7 3 3]
  [2 8 8 3 3 7 0 0 8 3 3 3 3 3 7]
  [2 5 1 8 3 3 3 3 3 3 3 3 3 3 3]
  [2 5 1 1 8 3 3 3 7 3 3 3 3 3 7]
  [2 5 1 1 1 8 3 3 3 3 3 3 3 7 2]
  [2 5 1 1 1 1 8 3 3 3 3 3 7 2 2]
  [2 5 1 1 1 1 1 8 3 3 3 3 7 6 0]
  [2 5 1 1 1 1 1 1 8 8 3 3 8 3 7]
  [2 5 1 1 1 1 1 1 1 8 8 7 3 7 3]
  [2 5 1 1 1 1 1 5 1 1 8 8 7 7 7]
  [2 5 1 1 5 5 1 8 1 1 8 8 3 3 3]
  [2 5 1 5 4 6 5 8 1 1 8 1 8 3 3]
  [2 5 5 4 4 0 5 8 1 1 8 1 1 8 3]
  [2 5 4 4 1 0 1 5 1 1 1 1 1 1 2]]
 Done!
 None
```

```
[[-5.75916230e-01 -3.78006873e-01 -3.78006873e-01 -3.78006873e-01
  -3.78006873e-01 -5.75916230e-01 -1.13682109e-92 -1.20879121e+00
  -5.75916230e-01 -3.78006873e-01 -3.78006873e-01 -3.78006873e-01
  -3.78006873e-01 -3.78006873e-01 -5.75916230e-01]
 [-3.78006873e-01  3.66845509e-01  3.66845509e-01  3.66845509e-01
   3.66845509e-01  3.66845509e-01 -7.80141844e-01 -7.80141844e-01
   3.66845509e-01  3.66845509e-01  3.66845509e-01  3.66845509e-01
   3.66845509e-01  3.66845509e-01  3.66845509e-01]
 [-3.78006873e-01  3.66845509e-01  1.62061876e+00  1.62061876e+00
   1.62061876e+00  1.62061876e+00 -7.80141844e-01 -7.80141844e-01
   1.62061876e+00  1.62061876e+00  1.62061876e+00  1.62061876e+00
   1.62061876e+00  1.62061876e+00  1.62061876e+00]
 [-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
   3.36197050e+00  3.36197050e+00  3.36197050e+00  3.36197050e+00
   3.36197050e+00  3.36197050e+00  3.36197050e+00  3.36197050e+00
   3.36197050e+00  3.36197050e+00  3.36197050e+00]
 [-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
   5.78051459e+00  5.78051459e+00  5.78051459e+00  5.78051459e+00
   5.78051459e+00  5.78051459e+00  5.78051459e+00  5.78051459e+00
   5.78051459e+00  5.78051459e+00  5.78051459e+00]
 [-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
   5.78051459e+00  9.13960359e+00  9.13960359e+00  9.13960359e+00
   9.13960359e+00  9.13960359e+00  9.13960359e+00  9.13960359e+00
   9.13960359e+00  9.13960359e+00  5.78051459e+00]
 [-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
   5.78051459e+00  9.13960359e+00  1.38050050e+01  1.38050050e+01
   1.38050050e+01  1.38050050e+01  1.38050050e+01  1.38050050e+01
   1.38050050e+01  9.13960359e+00  5.78051459e+00]
 [-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
   5.78051459e+00  9.13960359e+00  1.38050050e+01  2.02847292e+01
   2.02847292e+01  2.02847292e+01  2.02847292e+01  2.02847292e+01
   2.02847292e+01  9.13960359e+00 -1.20879121e+00]
 [-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
   5.78051459e+00  9.13960359e+00  1.38050050e+01  2.02847292e+01
   2.92843460e+01  2.92843460e+01  2.92843460e+01  2.92843460e+01
   2.92843460e+01  2.92843460e+01  2.92843460e+01]
 [-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
   5.78051459e+00  9.13960359e+00  1.38050050e+01  2.02847292e+01
   2.92843460e+01  4.17838140e+01  4.17838140e+01  4.17838140e+01
   4.17838140e+01  4.17838140e+01  4.17838140e+01]]
```

```
[-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
  5.78051459e+00  9.13960359e+00  1.38050050e+01  2.02847292e+01
  2.92843460e+01  4.17838140e+01  5.91441860e+01  5.91441860e+01
  5.91441860e+01  5.91441860e+01  5.91441860e+01]
[-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
  5.78051459e+00  9.13960359e+00  1.38050050e+01  2.02847292e+01
  2.92843460e+01  4.17838140e+01  5.91441860e+01  8.32558140e+01
  8.32558140e+01  8.32558140e+01  8.32558140e+01]
[-3.78006873e-01  3.66845509e-01  1.62061876e+00  3.36197050e+00
  3.36197050e+00  3.36197050e+00  1.38050050e+01  2.02847292e+01
  2.92843460e+01  4.17838140e+01  5.91441860e+01  8.32558140e+01
  1.16744186e+02  1.16744186e+02  1.16744186e+02]
[-3.78006873e-01  3.66845509e-01  1.62061876e+00  1.62061876e+00
  1.62061876e+00 -7.80141844e-01  1.38050050e+01  2.02847292e+01
  2.92843460e+01  4.17838140e+01  5.91441860e+01  8.32558140e+01
  1.16744186e+02  1.63255814e+02  1.63255814e+02]
[-5.75916230e-01  3.66845509e-01  3.66845509e-01  3.66845509e-01
 -5.75916230e-01 -1.20879121e+00  1.38050050e+01  2.02847292e+01
  2.92843460e+01  4.17838140e+01  5.91441860e+01  8.32558140e+01
  1.16744186e+02  1.63255814e+02  1.16744186e+02]]
```

در این حالت مقادیر value ها بسیار کوچک میشوند چون پاداش برخورد با مانع و هزینه ی حرکت را بالا بردیم و پاداش هدف را هم کم کرده ایم و همچنان مقدار در خانه های اطراف مانع کمتر است از بقیه خانه ها بخصوص قطر اصلی.

زیر بخش 5

حالت دوم را درنظر میگیریم یعنی حرکت بدون اصطکاک و برای 4 مقدار مختلف discount factor مسئله را بررسی میکنیم.

Discount factor = 0

```
*policy_stable: True*
[[2 4 4 4 4 1 0 0 2 4 4 4 4 4 1]
 [2 0 0 0 0 1 0 0 2 0 0 0 0 0 1]
 [2 0 0 0 0 1 0 0 2 0 0 0 0 0 1]
 [2 0 0 0 0 1 0 0 2 0 0 0 0 0 1]
 [2 0 0 0 0 5 4 4 6 0 0 0 0 0 1]
 [2 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [2 0 0 0 0 0 0 0 0 0 0 8 3 3 1]
 [2 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [2 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [2 0 0 0 0 0 0 0 0 0 0 5 4 4 1]
 [2 0 0 0 8 3 3 7 0 0 0 0 0 0 1]
 [2 0 0 0 1 0 0 2 0 0 0 0 0 0 1]
 [2 0 0 0 1 0 0 2 0 0 0 0 0 0 1]
 [2 0 0 0 1 0 0 2 0 0 0 0 0 8 3]
 [2 3 3 3 1 0 0 2 3 3 3 3 3 1 1]]
Done!
None
```

Discount factor = 0.3

```
 *policy_stable: True*
[[2 4 4 4 4 1 0 0 2 4 4 4 4 4 1]
 [2 8 3 3 3 1 0 0 2 3 3 3 3 3 1]
 [2 1 8 3 3 1 0 0 2 3 3 3 3 3 1]
 [2 1 8 8 3 3 3 3 3 3 3 3 3 3 3]
 [2 1 5 1 8 3 3 3 8 7 7 7 7 7 7]
 [2 1 5 1 8 8 8 7 7 7 7 7 7 7 2]
 [2 1 5 1 5 8 8 3 3 3 3 3 7 2 2]
 [2 1 5 1 5 8 1 8 3 3 3 3 7 6 0]
 [2 1 5 1 5 8 1 1 8 8 7 3 3 3 7]
 [2 1 5 1 5 8 1 1 1 8 8 3 7 3 7]
 [2 1 5 1 1 1 5 5 1 8 8 3 3 3 3]
 [2 1 5 1 5 5 1 1 1 8 1 8 8 7 3]
 [2 1 5 5 4 0 5 1 1 8 1 8 8 8 7]
 [2 1 5 4 1 0 5 1 1 8 1 8 1 8 3]
 [2 3 3 3 1 0 5 1 1 1 1 5 5 1 2]]
Done!
None
```

Discount factor = 0.6

```
 *policy_stable: True*
[[8 8 7 7 7 3 7 8 3 7 7 7 7 7 3]
 [8 8 3 3 3 7 7 8 8 3 3 3 3 3 7]
 [5 1 8 3 3 3 7 8 3 3 3 3 3 3 3]
 [5 1 1 8 8 3 8 7 3 7 7 7 7 7 3]
 [5 1 1 8 8 3 3 3 3 7 7 7 7 7 7]
 [5 1 1 8 1 8 3 3 3 3 3 3 3 7 7]
 [5 1 1 8 1 1 8 3 3 3 3 3 7 2 2]
 [5 1 1 8 1 1 8 8 3 3 3 3 7 6 6]
 [5 1 1 8 1 1 1 1 8 8 7 3 3 7 3]
 [5 1 1 8 1 1 1 1 8 8 3 7 7 7 7]
 [5 1 1 8 1 5 8 5 8 1 8 8 3 3 3]
 [5 1 1 1 5 5 1 1 8 1 8 8 3 3 3]
 [5 1 1 5 4 6 1 5 8 1 5 1 8 3 3]
 [5 1 5 4 4 6 1 5 8 1 5 5 1 8 3]
 [1 5 4 4 4 6 5 1 1 1 1 1 1 1 2]]
Done!
None
```

Discount factor = 0.9

```
 *policy_stable: True*
[[8 8 7 7 7 3 7 8 3 7 7 7 7 7 3]
 [8 8 3 3 3 3 7 8 8 3 3 3 3 3 3]
 [5 1 8 3 3 3 7 8 3 3 3 3 3 3 3]
 [5 1 1 8 3 3 3 3 3 3 3 3 3 3 3]
 [5 1 1 1 8 3 7 7 7 3 3 3 3 3 7]
 [5 1 1 1 1 8 3 3 3 3 3 3 3 7 2]
 [5 1 1 1 1 1 8 3 3 3 3 3 7 2 2]
 [5 1 1 1 1 1 1 8 3 3 3 3 7 6 6]
 [5 1 1 1 1 1 1 1 8 3 3 7 3 3 3]
 [5 1 1 1 1 1 1 1 1 8 3 8 3 7 3]
 [5 1 1 1 1 1 1 5 1 1 8 3 3 3 3]
 [5 1 1 1 5 5 8 1 1 1 1 8 3 3 3]
 [5 1 1 5 4 6 1 1 1 1 1 1 8 8 3]
 [5 1 5 4 4 6 1 1 1 1 1 1 8 8 3]
 [1 5 4 4 4 6 1 1 1 5 1 5 1 1 2]]
Done!
None
```

همانطور که میدانیم دو نوع reward در مسائل وجود دارد . یکی reward instant و یکی هم delayed reward. اگر ما با نگاه عمق کوتاه تصمیم گیری کنیم ،یعنی reward expected را فقط به پاداش لحظه ای منحصر کرده باشیم . اما اگر ما farsighted باشیم ، میتوانیم تاثیرات پاداش هایی که در آینده خواهیم گرفت را در return expected خود لحاظ کنیم.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

کمکی که گاما میکند به این صورت که اگر همه ی reward ها bounded باشند ، G_t هم bounded میشود

$$G_t = R_{t+1} + \gamma\, G_{t+1}$$

و چون G_t خودش یک variable random است ، خواهیم داشت:

$$V_\pi(s) = E_\pi[G_t | s_t = s]$$

ما factor discount را بین 0 تا 1 باید انتخاب کنیم.

اگر گاما را برابر 0 در نظر بگیریم همان حالت myopic میشود و هر چه که گاما را به سمت 1 ببریم ، به این معناست که reward هایی که در آینده میگیریم ، اهمیت بیشتری برای ما پیدا میکنند . و اگر گاما را هم برابر با 1 انتخاب کنیم ، یعنی ارزش همه ی reward هایی که قرار است بگیریم برای ما به یک اندازه است که این حالت تضمینی به ما برای همگرایی نمی دهد .

زیر بخش 6

روند اجرای کد پیاده‌سازی

تمام کدهای این تمرین در فایل نوت بوک ضمیمه شده قابل اجراست.

منابع

[1]
A. Choudhary, “Dynamic Programming in reinforcement learning,” *Analyticsvidhya.com*, 17-Sep-2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/. [Accessed: 11-Dec-2021].

[2]
R. Bhandarkar, “Policy Iteration in RL: A step by step Illustration,” *Towards Data Science*, 25-Mar-2020. [Online]. Available: https://towardsdatascience.com/policy-iteration-in-rl-an-illustration-6d58bdcb87a7. [Accessed: 11-Dec-2021].