ایلیا نورانی 9931092

گزارش پروژه سوم مبانی هوش مصنوعی

: Q1

برای محاسبه ارزش ها باید از روابط زیر استفاده کنیم.:

$$V^*(s) = \max_a Q^*(s,a)$$
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

اما روابط فوق محاسبه ارزش را تا عمق بی نهایت ادامه می دهند. برای جلوگیری از این کار از رابطه زیر استفاده می کنیم که محاسبه را تا عمق مشخص انجام می دهد:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

توضیح کد :

در تابع computeQvalueFromValue مقادیر Q را با استفاده از فرمول فوق محاسبه می کند(در واقع تمام کد این قسمت فقط پیاده سازی فرمول محاسبه Q*(s,a) می باشد). تابع computeActionFromValue در واقع argmax رااز طریق مقادیر محاسبه شده در تابع قبلی حساب می کند و به عنوان خروجی بر می گرداند.. تابع runValueIteration , value iteration را پیاده سازی می کند(طبق فرمول بالا). کد این قسمت تماما فرمول مربوطه را نشان می دهد و تنها نکته آن این است که برای terminal state هیچ کاری انجام نمی دهد (داخل حلقه).

```python
    def computeQValueFromValues(self, state, action):
        """
          Compute the Q-value of action in state from the
          value function stored in self.values.
        """

        q_value = 0
        transitions = self.mdp.getTransitionStatesAndProbs(state, action)
        for next_state, probability in transitions:
            reward = self.mdp.getReward(state, action, next_state)
            q_value += probability * (reward + (self.discount * self.values[next_state]))
        return q_value
        "*** YOUR CODE HERE -----done----- ***"
        util.raiseNotDefined()

    def computeActionFromValues(self, state):
        """
          The policy is the best action in the given state
          according to the values currently stored in self.values.

          You may break ties any way you see fit.  Note that if
          there are no legal actions, which is the case at the
          terminal state, you should return None.
        """
        actions = self.mdp.getPossibleActions(state)
        q_values = util.Counter()
        if self.mdp.isTerminal(state):
            return
        for action in actions:
            q_values[action] = self.computeQValueFromValues(state, action)
        return q_values.argMax()
        "*** YOUR CODE HERE ------done------ ***"
        util.raiseNotDefined()

```

```python
    def runValueIteration(self):
        # Write value iteration code here

        states = self.mdp.getStates()
        iterations = self.iterations
        for it in range(iterations):
            values = util.Counter()  # before each iteration, copy values to not work with real one.
            for state in states:
                if self.mdp.isTerminal(state):
                    continue
                compute_value = util.Counter()  # make a dictionary for all possible values for each action
                for action in self.mdp.getPossibleActions(state):
                    each_action_value = self.computeQValueFromValues(state, action)
                    compute_value[action] = each_action_value
                values[state] = max(compute_value.values())
            self.values = values

        "*** YOUR CODE HERE  -----done-----***"

```

خروجی های این قسمت به شکل زیر است:

```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q1
Starting on 1-19 at 15:54:33

Question q1
===========

*** PASS: test_cases\q1\1-tinygrid.test
*** PASS: test_cases\q1\2-tinygrid-noisy.test
*** PASS: test_cases\q1\3-bridge.test
*** PASS: test_cases\q1\4-discountgrid.test

### Question q1: 4/4 ###


Finished at 15:54:33

Provisional grades
==================
Question q1: 4/4
------------------
Total: 4/4
```

:Q2

برای این که عامل بخواهد از پل عبور کند باید مقدار نزدیک به یک باشد تا موقع رسیدن به هدف های با ارزش تر امتیاز نهایی خیلی کم نشود . همچنین باید نویز طوری انتخاب شود که عامل تقریبا مطمِن باشد درون هدف های بد نمی افتد(امتیاز منفی).با توجه به این توضیحات ازا ونجایی که تخفیف مان 0.9 است فقط کافی است نویز را برابر 0.0 قرار دهیم .

کد به همراه خروجی به شکل زیر است:

```
22    def question2():
23        answerDiscount = 0.9
24        answerNoise = 0.0
25        return answerDiscount, answerNoise
26
```

```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q2
Starting on 1-19 at 16:38:28

Question q2
===========

*** PASS: test_cases\q2\1-bridge-grid.test

### Question q2: 1/1 ###


Finished at 16:38:28

Provisional grades
==================
Question q2: 1/1
------------------
Total: 1/1
```

:Q3

a: تخفیف را کم می گذاریم تا خروجی نزدیک را ترجیح دهد.نویز را صفر می گذاریم تا عامل کنار صخره حرکت کند و به جانشین دلخواه برسیم(ریسک کند). همچنین پاداش را نزدیک مقدار نزدیکترین خروجی می گذاریم تا عامل حتی اگر در خروجی نامناسب رفت امتیازی قابل قبولی کسب کرده باشد.

B: تخفیف را کم می گذاریم تا خروجی نزدیک انتخاب شود. همچنین برای اینکه از کنار صخره عبور نکند مقدار امتیاز زنده بودن را کم می گذاریم و نویز را کمی بیشتر می کنیم تا قطعی پیش نرود.

C: تخفیف را زیاد می گذاریم تا عامل خروجی دور را ترجیح دهد . همچنین برای این که عامل ریسک کند و مسیر های کنار صخره را انتخاب کند باید مقدار امتیاز زنده بودن را زیاد کنیم و برای قطعی بودن حرکت ها نویز را کم می کنیم.

D:تخفیف را زیاد می گذاریم تا خروجی دور انتخاب شود . همچنین نویز را زیاد و امتیاز زنده بودن را کم می گذاریم تا عامل مسیر های صخره را انتخاب نکند.

E :در این حالت تخفیف را زیاد در نظر می گیریم تا عامل نگران رسیدن به هدف نباشد. همچنین امتیاز زنده ماندن را زیاد در نظر می گیریم تا عامل نخواهد به بازی پایان دهد . نویز را هم کمی زیاد می کنیم تا حرکات عامل قطعی نباشد.

کد به همراه خروجی در ادامه آمده است:

```python
def question3a():
    answerDiscount = 0.1
    answerNoise = 0.0
    answerLivingReward = 0.7
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3b():
    answerDiscount = 0.1
    answerNoise = 0.1
    answerLivingReward = 0.2
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3c():
    answerDiscount = 0.8
    answerNoise = 0.0
    answerLivingReward = 0.4
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3d():
    answerDiscount = 0.9
    answerNoise = 0.5
    answerLivingReward = 0.7
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3e():
    answerDiscount = 1.0
    answerNoise = 0.2
    answerLivingReward = 15
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

```

```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q3
Starting on 1-19 at 17:21:41

Question q3
===========

*** PASS: test_cases\q3\1-question-3.1.test
*** PASS: test_cases\q3\2-question-3.2.test
*** PASS: test_cases\q3\3-question-3.3.test
*** PASS: test_cases\q3\4-question-3.4.test
*** PASS: test_cases\q3\5-question-3.5.test

### Question q3: 5/5 ###


Finished at 17:21:41

Provisional grades
==================
Question q3: 5/5
------------------
Total: 5/5
```

جواب سوال مربوط به این بخش:

بله- value iteration همیشه به همگرایی می رسد. یعنی پیمایش را تا جایی ادامه می دهد که تغییرات بسیار ناچیز باشد(همگرایی).

:Q4

تنها تفاوت این قسمت با value iteration بخش قبل این است که برخلاف بخش قبل که در هر پیمایش تمام ادزش ها آپدیت می شدند در اینجا در هر پیمایش فقط یک ارزش تغییر می کند.یعنی در کد این بخش تنها یک حالت (state) انتخاب می شود و مقدار آن آپدیت می شود و برای حالت خروجی تغییری صورت نمی گیرد(خط 163 تا 170).

کد و خروجی در ادامه آمده است:

```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q4
Starting on 1-19 at 18:32:47

Question q4
==========

*** PASS: test_cases\q4\1-tinygrid.test
*** PASS: test_cases\q4\2-tinygrid-noisy.test
*** PASS: test_cases\q4\3-bridge.test
*** PASS: test_cases\q4\4-discountgrid.test

### Question q4: 1/1 ###


Finished at 18:32:47

Provisional grades
==================
Question q4: 1/1
------------------
Total: 1/1
```

```
160     def runValueIteration(self):
161
162         states = self.mdp.getStates()
163         for iteration in range(self.iterations):
164             state = states[iteration % len(states)]
165             if self.mdp.isTerminal(state):
166                 continue
167             specific_state = util.Counter()
168             for action in self.mdp.getPossibleActions(state):
169                 specific_state[action] = self.computeQValueFromValues(state, action)
170             self.values[state] = max(specific_state.values())
171         "*** YOUR CODE HERE ------done------ ***"
172
```

پاسخ سوال این بخش : در بخش قبل به ازای هر بروز رسانی همه حالت ها بررسی می شدند که این کار باعث می شود در نهایت ما iteration های کمتری داشته باشیم. اما مزیت پیاده سازی این قسمت این است که برای تغییر دادن یک ارزش لازم نیست همه ی ارزش ها را آپدیت کنیم.

:Q5

توضیح کد:

در خط 191 تا 194 یک صف اولویت و یک مجموعه (set) برای جانشین ها ساخته می شود . در خط 197 چک می شود اگر در حالت خروجی بود کاری انجام نشود. در خط 203 تا 206 همه جانشین ها بررسی می شود(همجنین یک مجموعه تعریف می شود تا از به وجود آمدن حالت های تکراری جلوگیری شود). سپس مقدار تفاوت را حساب می کند (diff). از آنجایی که صف به صورت min heap است تفوت ها به صورت منفی در صف اضافه می شوند(خط 209). پس از آن بررسی می کند اگر اگر صف خالی است کار ما تمام است . در غیر این صورت یک حالت را از صف خارج می کنیم(s) و اگر حالت خروجی نبود آن را آپدیت می کنیم . سپس مقدار diff را محاسبه می کنیم و اگر diff>theta آن را به صورت برعکس (منفی) به صف اضافه می کنیم.

کد و خروجی به شکل زیر است:

```python
    def runValueIteration(self):

        states = self.mdp.getStates()
        priority_queue = util.PriorityQueue()
        predecessor = {}

        for state in states:

            if self.mdp.isTerminal(state):
                continue
            q_values = {}

            for action in self.mdp.getPossibleActions(state):
                for next_state, prob in self.mdp.getTransitionStatesAndProbs(state, action):
                    if next_state not in predecessor:
                        predecessor[next_state] = set()
                    predecessor[next_state].add(state)
                q_values[action] = self.getQValue(state, action)
            diff = abs(self.values[state] - max(q_values.values()))
            priority_queue.update(state, -diff)

        for iteration in range(self.iterations):
            if priority_queue.isEmpty():
                break
            state = priority_queue.pop()
            if self.mdp.isTerminal(state):
                continue
            update_values = {}
            for action in self.mdp.getPossibleActions(state):
                update_values[action] = self.getQValue(state, action)
            self.values[state] = max(update_values.values())
            for p in predecessor[state]:
                q_values = {}
                for action in self.mdp.getPossibleActions(p):
                    q_values[action] = self.getQValue(p, action)
                diff = abs(self.values[p] - max(q_values.values()))
                if diff > self.theta:
                    priority_queue.update(p, -diff)

        "*** YOUR CODE HERE ----done----- ***"
```

```
D:\AUT\AI\p3\AI_P3\AI_P3>python gridworld.py -a priosweepvalue -i 1000
```

Gridworld Display


```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q5
Starting on 1-19 at 20:44:48

Question q5
===========

*** PASS: test_cases\q5\1-tinygrid.test
*** PASS: test_cases\q5\2-tinygrid-noisy.test
*** PASS: test_cases\q5\3-bridge.test
*** PASS: test_cases\q5\4-discountgrid.test

### Question q5: 3/3 ###


Finished at 20:44:49

Provisional grades
==================
Question q5: 3/3
------------------
Total: 3/3
```

:Q6

فرمول های مورد نیاز برای این بخش:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

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

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

توضیح کد:

getQvalue : مقدار Q(s,a) را با توجه به ورودی بر می گرداند.

computeValueFromQvalue : یک پیمایش بر روی Qvalues انجام می دهد و مقدار ماکسیمم را بر میگرداند(در حالت خروجی 0.0 برمی گرداند)

computeActionFromQValues روی QValues یک پیمایش انجام می دهد و بهترین حرکت را برمی گرداند.

Update : در این تابع فرمول آخر (در بالا آمده است) محاسبه می شود و qValue را آپدیت می کند.

در حرکاتی که عامل تا به حال مشاهده نکرده مقدار q برابر صفر است و بعد از آن با کسب تجربه مقادیر تغییر می کنند.

کد به همراه خروجی در ادامه آمده است:

```
118     def update(self, state, action, nextState, reward):
119         """
120           The parent class calls this to observe a
121           state = action => nextState and reward transition.
122           You should do your Q-Value update here
123
124           NOTE: You should never call this function,
125           it will be called on your behalf
126         """
127
128         self.q_values[(state, action)] = (1 - self.alpha) * self.getQValue(state, action) + self.alpha * (reward + self.discount * self.getValue(nextState))
129
130         "*** YOUR CODE HERE --------done------ ***"
131         #util.raiseNotDefined()
132
```

```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q6
Starting on 1-19 at 23:23:28

Question q6
===========

*** PASS: test_cases\q6\1-tinygrid.test
*** PASS: test_cases\q6\2-tinygrid-noisy.test
*** PASS: test_cases\q6\3-bridge.test
*** PASS: test_cases\q6\4-discountgrid.test

### Question q6: 4/4 ###


Finished at 23:23:28

Provisional grades
==================
Question q6: 4/4
------------------
Total: 4/4
```

```
47      def getQValue(self, state, action):
48          """
49            Returns Q(state,action)
50            Should return 0.0 if we have never seen a state
51            or the Q node value otherwise
52          """
53          return self.q_values[(state, action)]
54          "*** YOUR CODE HERE  ---done----***"
55          util.raiseNotDefined()
56
57
58      def computeValueFromQValues(self, state):
59          """
60            Returns max_action Q(state,action)
61            where the max is over legal actions.  Note that if
62            there are no legal actions, which is the case at the
63            terminal state, you should return a value of 0.0.
64          """
65          if self.getLegalActions(state):
66              q_values = util.Counter()
67              for action in self.getLegalActions(state):
68                  q_values[action] = self.getQValue(state, action)
69              return max(q_values.values())
70          return 0.0
71          "*** YOUR CODE HERE  -----done----- ***"
72          util.raiseNotDefined()
73
74      def computeActionFromQValues(self, state):
75          """
76            Compute the best action to take in a state.  Note that if there
77            are no legal actions, which is the case at the terminal state,
78            you should return None.
79          """
80
81          if self.getLegalActions(state):
82              best_val = self.computeValueFromQValues(state)
83              max_q_val = []
84              for action in self.getLegalActions(state):
85                  if best_val == self.getQValue(state, action):
86                      max_q_val.append(action)
87              return random.choice(max_q_val)
88          return None
89
90          "*** YOUR CODE HERE ------done------ ***"
91          util.raiseNotDefined()
92
```

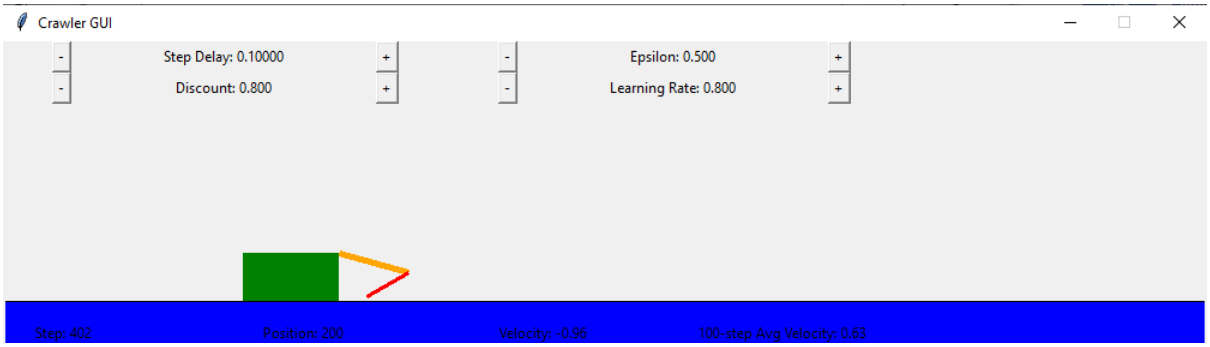
:Q7

توضیح کد:اگر حرکت مجاز وجود داشته باشد با استفاده از اپسیلون یک سکه پرتاب می کنیم. و با توجه به آن مشخص می شود که حرکت رندوم انجام دهیم یا بر اساس بهترین مقدار Q عمل کنیم(خط 108 تا 112).

کد و خروجی به شکل زیر است:

```python
    def getAction(self, state):
        """
          Compute the action to take in the current state.  With
          probability self.epsilon, we should take a random action and
          take the best policy action otherwise.  Note that if there are
          no legal actions, which is the case at the terminal state, you
          should choose None as the action.

          HINT: You might want to use util.flipCoin(prob)
          HINT: To pick randomly from a list, use random.choice(list)
        """
        # Pick Action
        legalActions = self.getLegalActions(state)
        action = None

        if legalActions:
            if util.flipCoin(self.epsilon):
                return random.choice(legalActions)
            return self.computeActionFromQValues(state)
        return action
        "*** YOUR CODE HERE ***"
        util.raiseNotDefined()

        return action
```

```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q7
Starting on 1-19 at 23:48:22

Question q7
===========

*** PASS: test_cases\q7\1-tinygrid.test
*** PASS: test_cases\q7\2-tinygrid-noisy.test
*** PASS: test_cases\q7\3-bridge.test
*** PASS: test_cases\q7\4-discountgrid.test

### Question q7: 2/2 ###


Finished at 23:48:24

Provisional grades
==================
Question q7: 2/2
------------------
Total: 2/2
```



: Q8

با اپسیلون یک عامل در تمام مراحل در حال یادگیری است که باعث می شود به سیسات بهینه دست پیدا نکند.

با اپسیلون صفر نیز عامل یادگیری امجام نمی دهد و از معلومات خود استفاده می کند که باعث می شود عامل به سیاست بهینه نرسد.

در نتیجه به ازای هیچ اپسیلونی به سیسات بهینه نمی رسیم.

```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q8
Starting on 1-19 at 23:51:32

Question q8
===========

*** PASS: test_cases\q8\grade-agent.test

### Question q8: 1/1 ###


Finished at 23:51:32

Provisional grades
==================
Question q8: 1/1
------------------
Total: 1/1
```


```python
def question8():
    answerEpsilon = None
    answerLearningRate = None

    #return answerEpsilon, answerLearningRate
    # If not possible, return 'NOT POSSIBLE'

    return 'NOT POSSIBLE'

```

: Q9

```
        Completed 1500 out of 2000 training episodes
        Average Rewards over all training: -181.59
        Average Rewards for last 100 episodes: 256.73
        Episode took 1.34 seconds
Reinforcement Learning Status:
        Completed 1600 out of 2000 training episodes
        Average Rewards over all training: -156.10
        Average Rewards for last 100 episodes: 226.23
        Episode took 1.36 seconds
Reinforcement Learning Status:
        Completed 1700 out of 2000 training episodes
        Average Rewards over all training: -136.58
        Average Rewards for last 100 episodes: 175.74
        Episode took 1.35 seconds
Reinforcement Learning Status:
        Completed 1800 out of 2000 training episodes
        Average Rewards over all training: -113.08
        Average Rewards for last 100 episodes: 286.46
        Episode took 1.38 seconds
Reinforcement Learning Status:
        Completed 1900 out of 2000 training episodes
        Average Rewards over all training: -94.18
        Average Rewards for last 100 episodes: 245.93
        Episode took 1.36 seconds
Reinforcement Learning Status:
        Completed 2000 out of 2000 training episodes
        Average Rewards over all training: -77.69
        Average Rewards for last 100 episodes: 235.71
        Episode took 1.38 seconds
Training Done (turning off epsilon and alpha)
```

```
----------------------------------------------
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 498
Pacman emerges victorious! Score: 495
Average Score: 498.1
Scores:        495.0, 499.0, 499.0, 499.0, 499.0, 499.0, 495.0, 503.0, 498.0, 495.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

```
Win Rate:      100/100 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q9\grade-agent.test (1 of 1 points)
***     Grading agent using command:  python pacman.py -p PacmanQAgent -x 2000 -n 2100 -l smallGrid -q -f --fixRandomSee
d
***     100 wins (1 of 1 points)
***         Grading scheme:
***          < 70:  0 points
***         >= 70:  1 points

### Question q9: 1/1 ###


Finished at 4:15:34

Provisional grades
==================
Question q9: 1/1
------------------
Total: 1/1
```

:Q10

در پیاده سازی این قسمت از فرمول های زیر استفاده می کنیم :

$$Q(s,a) = \sum_{i=1}^{n} f_i(s,a) w_i$$

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$

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

از آنجایی که محاسبه Q ها در واقعیت غیرممکن است از Approximate Q-learning استفاده می کنیم و مقدار را به صورت تقریبی محاسبه می کنیم. یعنی مقدار Q را ابتدا با استفاده از فرمول بالا بدست می آوریم و مقدار تقریبی آن را از مقدار واقعی کم می کنیمو سپس مقدار تفاوت (difference) را محاسبه می کنیم و بر اساس آن مقادیر وزن ها را آپدیت می کنیم.

کد و خروجی به صورت زیر است :

```
188     def getQValue(self, state, action):
189         """
190           Should return Q(state,action) = w * featureVector
191           where * is the dotProduct operator
192         """
193 
194         feats = self.featExtractor.getFeatures(state, action)
195         return sum([feats[feature] * self.weights[feature] for feature in feats])
196 
197         "*** YOUR CODE HERE -----done----- ***"
198         util.raiseNotDefined()
199 
200     def update(self, state, action, nextState, reward):
201         """
202           Should update your weights based on transition
203         """
204 
205         difference = (reward + (self.discount * self.getValue(nextState))) - self.getQValue(state, action)
206         feats = self.featExtractor.getFeatures(state, action)
207         for feature in feats:
208             self.weights[feature] = self.weights[feature] + (self.alpha * difference * feats[feature])
209 
210         "*** YOUR CODE HERE -----done----- ***"
211         #util.raiseNotDefined()
```

getQValue : مقدار Q را طبق فرمول بالا محاسبه می کند(تقریبی).

Update : مقدار تفاوت (difference) را حساب می کند و با استفاده از آن طبق فرمول مقادری وزن ها را آپدیت می کند.

```
D:\AUT\AI\p3\AI_P3\AI_P3>python autograder.py -q q10
Starting on 1-20 at 4:37:13

Question q10
============

*** PASS: test_cases\q10\1-tinygrid.test
*** PASS: test_cases\q10\2-tinygrid-noisy.test
*** PASS: test_cases\q10\3-bridge.test
*** PASS: test_cases\q10\4-discountgrid.test
*** PASS: test_cases\q10\5-coord-extractor.test

### Question q10: 3/3 ###


Finished at 4:37:14

Provisional grades
==================
Question q10: 3/3
------------------
Total: 3/3
```

```
Reinforcement Learning Status:
        Completed 1600 out of 2000 training episodes
        Average Rewards over all training: -157.91
        Average Rewards for last 100 episodes: 176.12
        Episode took 1.92 seconds
Reinforcement Learning Status:
        Completed 1700 out of 2000 training episodes
        Average Rewards over all training: -135.24
        Average Rewards for last 100 episodes: 227.38
        Episode took 1.99 seconds
Reinforcement Learning Status:
        Completed 1800 out of 2000 training episodes
        Average Rewards over all training: -116.26
        Average Rewards for last 100 episodes: 206.40
        Episode took 2.07 seconds
Reinforcement Learning Status:
        Completed 1900 out of 2000 training episodes
        Average Rewards over all training: -100.87
        Average Rewards for last 100 episodes: 176.24
        Episode took 2.00 seconds
Reinforcement Learning Status:
        Completed 2000 out of 2000 training episodes
        Average Rewards over all training: -84.47
        Average Rewards for last 100 episodes: 227.01
        Episode took 2.00 seconds
Training Done (turning off epsilon and alpha)
-----------------------------------------
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 501
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 501
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Average Score: 501.0
Scores:        503.0, 501.0, 499.0, 503.0, 501.0, 503.0, 499.0, 499.0, 503.0, 499.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```