مبانی و کاربردهای هوش مصنوعی

پروژه‌ی سوم

تانیا جواهرپور ۹۷۳۳۰۱۷

پروژه مطابق دستورکار ارائه شده پیاده سازی شده است و برای تمامی بخش‌ها در کد کامنت و توضیحات واضح گذاشته شده است.

۱. تکرار ارزش:

برای این بخش همانطور که در دستور پروژه ذکر شده است باید به تکمیل تابع computeActionFromValues و computeQValueFromValues در فایل valueIterationAgents.py و در کلاس valueIterationAgent بپردازیم تا با استفاده از تابع اول بهترین عمل را با توجه به تابع مقدارداده شده توسط self.values و با تابع دوم که ورودی آن state و action است Q-value را برای action در state با توجه به تابع مقدار ذخیره شده در self.values محاسبه شود. هم‌چنین نیاز است تا تابع runValueIteration که در ابتدای این کلاس صدا زده می‌شود نیز تکمیل گردد تا مقدار self.values محاسبه گردد.

در نهایت هدف اصلی پیاده‌سازی رابطه‌ی زیر است:

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

- **توضیح کد:**

تابع computeQValueFromValues:

در این تابع دقیقا مطابق فرمول محاسبه میکنیم که با انجام عمل a در موقعیت s بر اساس احتمال رخداد هر موقعیت ثانویه، پاداش به دست آمده، ارزش فعلی موقعیت ثانویه و گاما مقدار Q، که در واقع در فرمول بالا بخشی است که مجموع محاسبه میشود، به دست آید.

برای نگهداری مقادیر هر موقعیت ثانویه که نیاز به مجموع(total) آن داریم، از شمارنده‌ای که در خود برنامه پیاده شده است و موجودیت دیکشنری دارد استفاده میکنیم.

```
def computeQValueFromValues(self, state, action):
    """
```

```
      Compute the Q-value of action in state from the
      value function stored in self.values.
    """
    "*** YOUR CODE HERE ***"
    Qvalues = util.Counter()
    # nextState = next possible state by doing this action
    # T = probability of reaching nextState
    for nextState, T in self.mdp.getTransitionStatesAndProbs(state,
action):
        R = self.mdp.getReward(state, action, nextState)
        # self.discount = gama
        # self.value is being computed in runValueIteration
        Qvalues[nextState] = T * (R +
self.discount*self.values[nextState])
    return Qvalues.totalCount()
```

تابع computeActionFromValues:

این تابع دقیقا مانند فرمول از مقدار Q محاسبه شده استفاده می‌کند و با ماکسیمم گرفتن ارزش در $k+1$ را به دست می‌آورد.

برای نگهداری مقادیر ارزش هر عمل که نیاز به بیشینه(argMax) آن داریم، از شمارنده‌ای که در خود برنامه پیاده شده است و موجودیت دیکشنری دارد استفاده میکنیم.

```
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.
    """
    "*** YOUR CODE HERE ***"
    # Computing action(Vk+1(s)) by using Q values
    actionValues = util.Counter()
    for a in self.mdp.getPossibleActions(state):
        actionValues[a] = self.computeQValueFromValues(state, a)
    return actionValues.argMax()
```

تابع runValueIteration:

در تابع computeQValueFromValues هنگام محاسبه‌ی QValue نیاز به self.values است و همانطور که در ابتدا گفته شد در این تابع محاسبه می‌شود.

این تابع برای تمامی موقعیت‌ها و عمل‌هایی که در هر کدام از استیت‌ها می‌تواند انجام دهیم اجرا می‌شود.

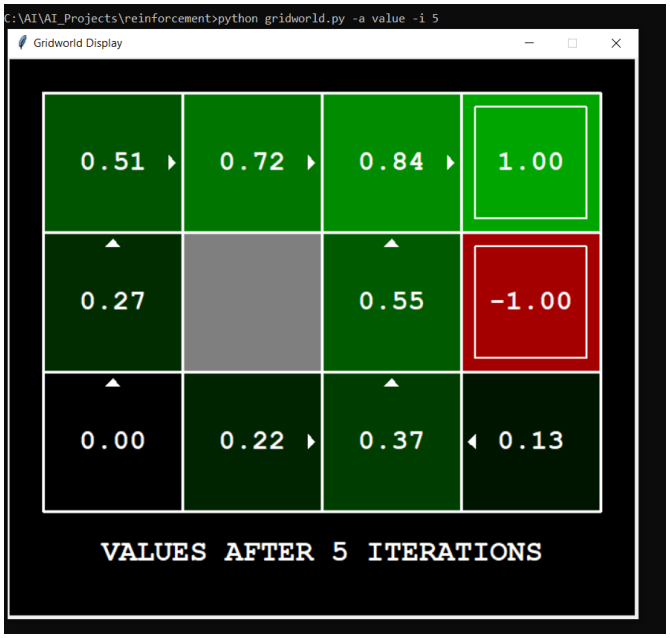
```
def runValueIteration(self):
    # Write value iteration code here
    "*** YOUR CODE HERE ***"
    # For each k, which means each iteration,
    # this loop computes all of the values for each state
    for each in range(self.iterations):

        updatedValues = util.Counter()

        for s in self.mdp.getStates():
            if not self.mdp.isTerminal(s):
                updatedValues[s] = -999
                updatedValues[s] = self.calculateBestQValue(s,
updatedValues[s])
        self.values = updatedValues
```

تابع calculateBestQvalue:

این تابع که در runValueIteration هم از آن استفاده شد با توجه به Qvalueی فعلی بر اساس موقعیت فعلی و اعمالی که در این اکشن می‌توانیم انجام دهیم بهترین Qvalueی ممکن را محاسبه میکند و بیشتر مقدار ممکن آن را برمیگرداند. این کار چون در سه کلاس بعدی که از این کلاس ارث‌بری میکنند هم نیاز بود، به همین دلیل به صورت تابع مجزا نوشته شد تا از تکرار در کد جلوگیری شود.

```
def calculateBestQValue(self, state, bestQValue):
    """
    Finds best value of Q based on possible actions of each state
    """
    for action in self.mdp.getPossibleActions(state):
        QValue = self.computeQValueFromValues(state, action)
        bestQValue = max(bestQValue, QValue)
    return bestQValue
```

- تصاویری از خروجی این بخش:



```
C:\AI\AI_Projects\reinforcement>python gridworld.py -a value -i 5

RUNNING 1 EPISODES

BEGINNING EPISODE: 1

Started in state: (0, 0)
Took action: north
Ended in state: (0, 1)
Got reward: 0.0

Started in state: (0, 1)
Took action: north
Ended in state: (0, 1)
Got reward: 0.0

Started in state: (0, 1)
Took action: north
Ended in state: (0, 1)
Got reward: 0.0

Started in state: (0, 1)
Took action: north
Ended in state: (0, 2)
Got reward: 0.0

Started in state: (0, 2)
Took action: east
Ended in state: (1, 2)
Got reward: 0.0

Started in state: (1, 2)
Took action: east
Ended in state: (1, 2)
Got reward: 0.0

Started in state: (1, 2)
Took action: east
Ended in state: (2, 2)
Got reward: 0.0

Started in state: (2, 2)
Took action: east
Ended in state: (3, 2)
Got reward: 0.0

Started in state: (3, 2)
Took action: exit
Ended in state: TERMINAL_STATE
Got reward: 1

EPISODE 1 COMPLETE: RETURN WAS 0.43046721000000016


AVERAGE RETURNS FROM START STATE: 0.43046721000000016
```

- **نتیجه‌ی خروجی autograder:**

```
C:\AI\AI_Projects\reinforcement>python autograder.py -q q1
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 1-23 at 22:36:03

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 22:36:03

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

۲. تجزیه و تحلیل عبور از پل

می‌دانیم هر چقدر نویز کمتر و گاما بیشتر باشد بهتر است؛ زیرا افزایش نویز موجب افزایش اعمال ناخواسته و هر چقدر گاما بیشتر باشد، وقتی طول مسیر برای رسیدن به هدف افزایش یابد ارزش به دست آمده دچار تخفیف کمتری میشود.

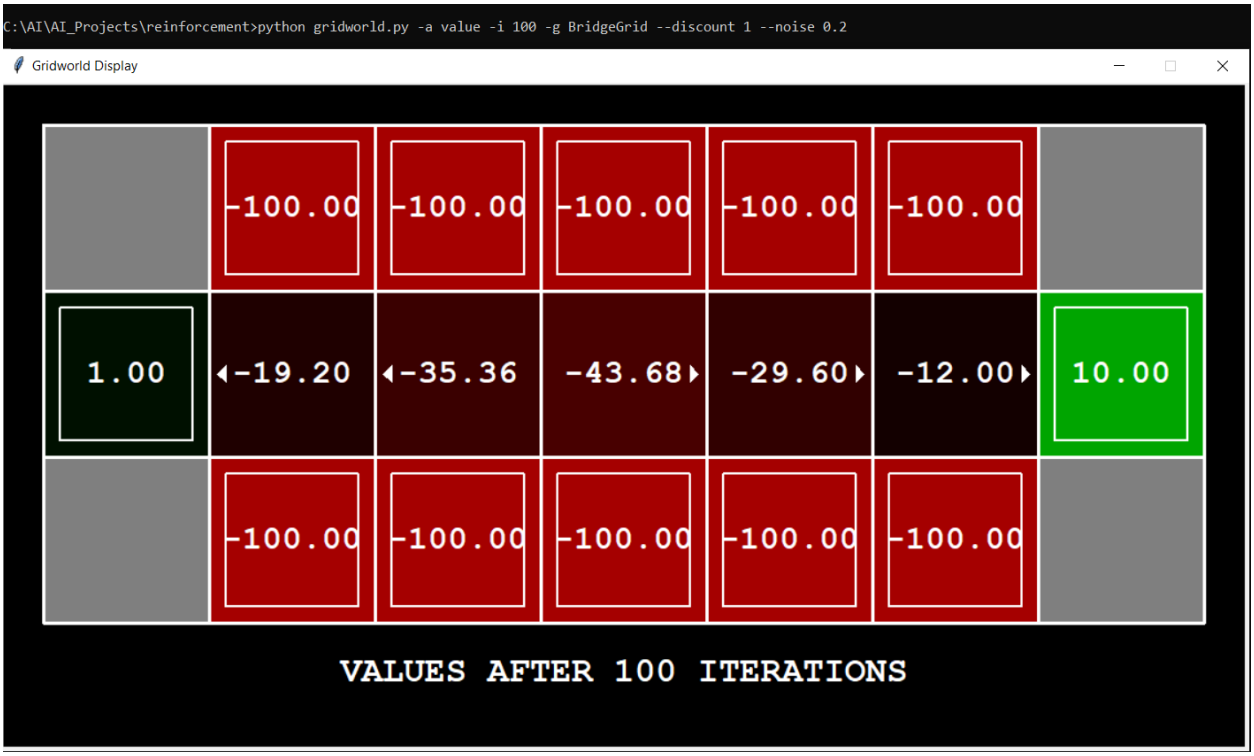
با این حال با توجه به این که تنها یکی از پارامترها را میشد تغییر داد ابتدا تغییر تخفیف را با کاهش آن و سپس با افزایش آن و نزدیک کردن آن به 1 بررسی کردیم. از آنجا که نتایج همچنان منفی بود سراغ تست کردن و کاهش نویز رفتیم که همانطور که مشاهده میکنید هر چقدر به 0 نزدیک‌تر شد مقادیر مثبت بیشتر و مثبتر شدند. به همین دلیل نویز را 0 قرار دادیم.

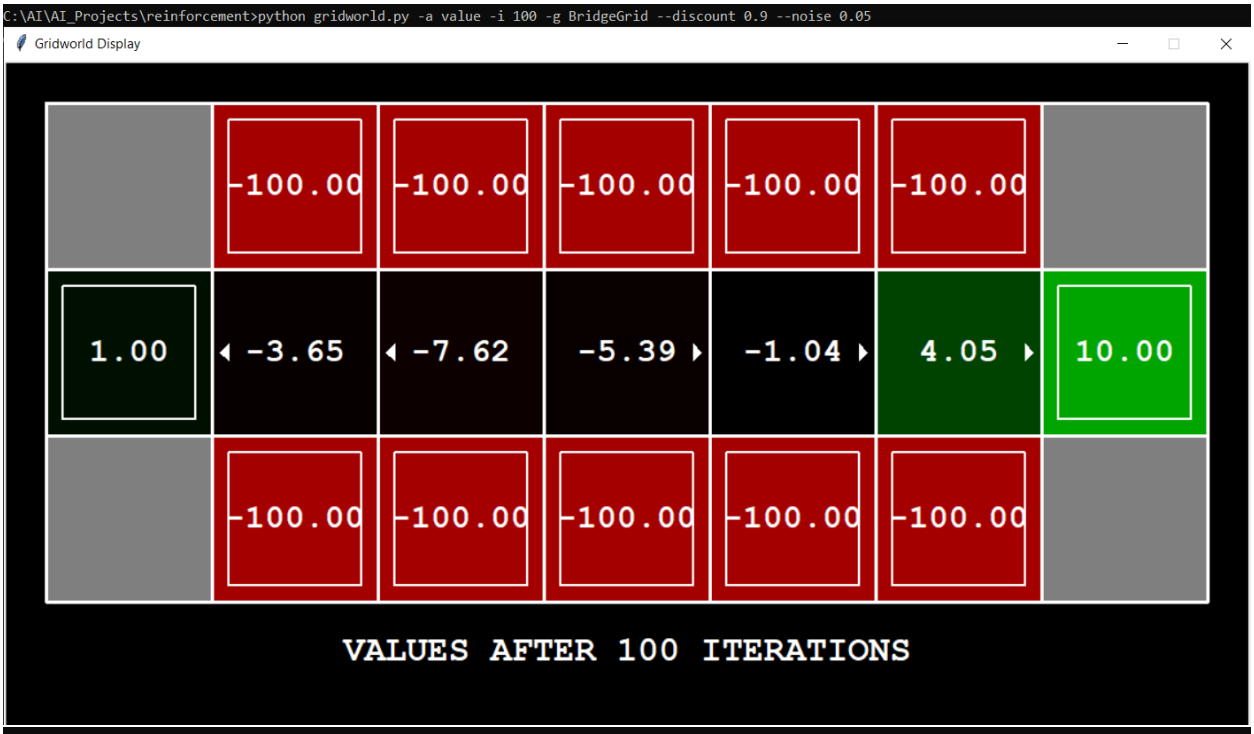
- **توضیح کد:**

```python
def question2():
    answerDiscount = 0.9
    answerNoise = 0
    return answerDiscount, answerNoise
```

- تصاویری از خروجی این بخش:

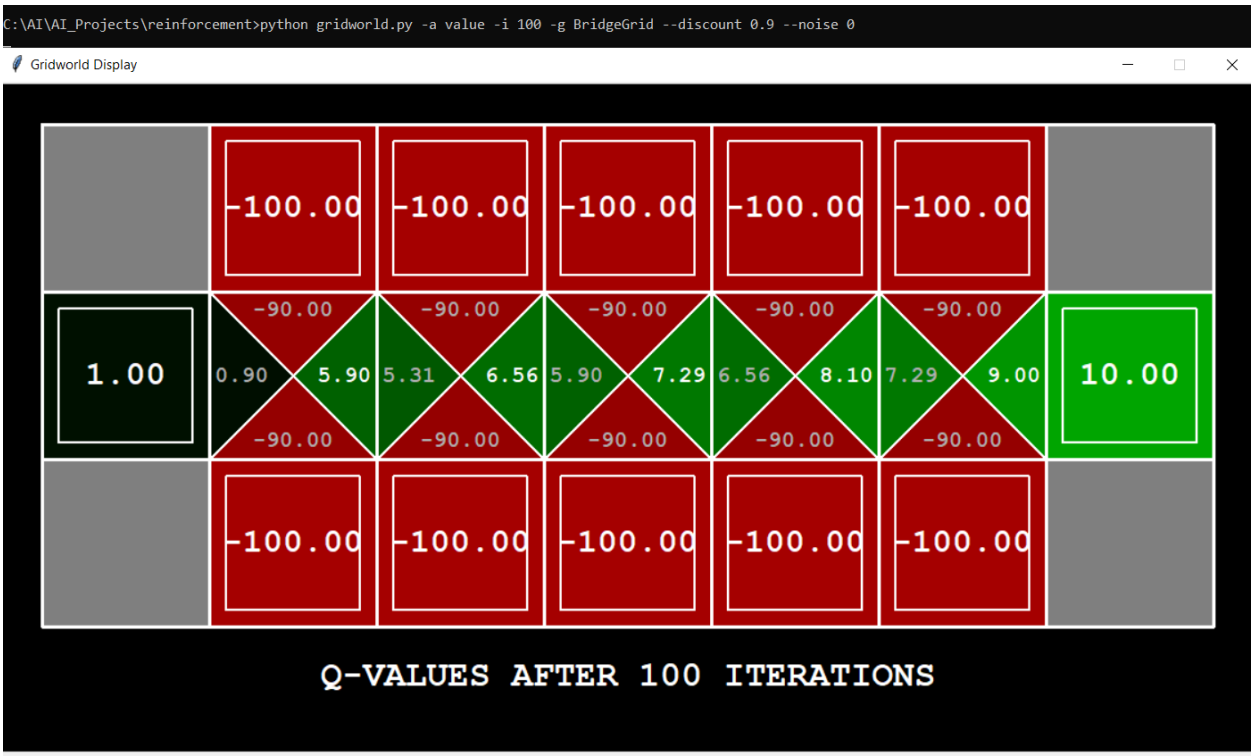

C:\AI\AI_Projects\reinforcement>python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.1 --noise 0.2
Gridworld Display
-100.00
-100.00
-100.00
-100.00
-100.00
1.00
-1.92
-2.15
-2.17
-2.10
-1.20
10.00
-100.00
-100.00
-100.00
-100.00
-100.00
VALUES AFTER 100 ITERATIONS


C:\AI\AI_Projects\reinforcement>python gridworld.py -a value -i 100 -g BridgeGrid --discount 1 --noise 0.2
Gridworld Display
-100.00
-100.00
-100.00
-100.00
-100.00
1.00
-19.20
-35.36
-43.68
-29.60
-12.00
10.00
-100.00
-100.00
-100.00
-100.00
-100.00
VALUES AFTER 100 ITERATIONS


C:\AI\AI_Projects\reinforcement>python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.05
Gridworld Display
-100.00
-100.00
-100.00
-100.00
-100.00
1.00
-3.65
-7.62
-5.39
-1.04
4.05
10.00
-100.00
-100.00
-100.00
-100.00
-100.00
VALUES AFTER 100 ITERATIONS


C:\AI\AI_Projects\reinforcement>python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.01
Gridworld Display
-100.00
-100.00
-100.00
-100.00
-100.00
1.00
2.00
3.25
4.66
6.24
8.01
10.00
-100.00
-100.00
-100.00
-100.00
-100.00
VALUES AFTER 100 ITERATIONS

خروجی ایده‌آل



```
C:\AI\AI_Projects\reinforcement>python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0

RUNNING 1 EPISODES

BEGINNING EPISODE: 1

Started in state: (1, 1)
Took action: east
Ended in state: (2, 1)
Got reward: 0.0

Started in state: (2, 1)
Took action: east
Ended in state: (3, 1)
Got reward: 0.0

Started in state: (3, 1)
Took action: east
Ended in state: (4, 1)
Got reward: 0.0

Started in state: (4, 1)
Took action: east
Ended in state: (5, 1)
Got reward: 0.0

Started in state: (5, 1)
Took action: east
Ended in state: (6, 1)
Got reward: 0.0

Started in state: (6, 1)
Took action: exit
Ended in state: TERMINAL_STATE
Got reward: 10

EPISODE 1 COMPLETE: RETURN WAS 5.904900000000001


AVERAGE RETURNS FROM START STATE: 5.904900000000001
```

- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\reinforcement>python autograder.py -q q2
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 1-24 at 21:12:48

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

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

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


Finished at 21:12:48

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

۳. سیاست‌ها

این بخش بسیار شبیه به بخش دوم می‌باشد، بنابراین توضیحاتی که برای نویز و تخفیف دادیم اینجا هم صدق میکند؛ اما اینجا چون نیاز به تغییر reward هم هست، با در نظر گرفتن آن نکات و این که هر چقدر پاداش زندگی منفی‌تر باشد تعداد حرکات برای رسیدن به حالت‌های پایانی کمتر میشود و اول به سمت پاداش کوچکتر می‌رویم این بخش را تکمیل کردیم و با کمک auto grader حدس‌ها و پیشنهادات خودمان را برای این مقادیر ارزیابی کردیم.

- **توضیح کد:**

در قسمت اول خروجی نزدیک ترجیح داده شده است و ریسک‌پذیری بالاست بنابراین باید پاداش مقدار زیادی منفی باشد.

در قسمت دوم هم‌چنان خروجی نزدیک مهم است پس پاداش هنوز مقدار منفی دارد اما نکته‌ی دارای اهمیت این است که دیگر ریسک‌پذیزی نداریم و باید از صخره اجتناب کنیم بنابراین نیاز به افزایش نویز و discount داریم.

در قسمت سوم نیاز به افزایش discount و منفی‌تر شدن پاداش بود.

در قسمت چهارم نیاز بود تا پاداش کمتر منفی باشد و در قسمت پنجم نیازنتد پاداش مثبت بودیم.

در تمامی قسمت‌ها با در نظر گرفتن نکات گفته شده موارد دستور کار را نیز در نظر گرفتیم و با auto grader از درستی اعداد در نظر گرفته شده اطمینان حاصل کردیم.

```
def question3a():
    answerDiscount = 0.02
    answerNoise = 0
    answerLivingReward = -4
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3b():
    answerDiscount = 0.5
    answerNoise = 0.3
    answerLivingReward = -0.5
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3c():
    answerDiscount = 0.9
    answerNoise = 0.2
    answerLivingReward = -1.5
    return answerDiscount, answerNoise, answerLivingReward
```

```
    # If not possible, return 'NOT POSSIBLE'

def question3d():
    answerDiscount = 0.8
    answerNoise = 0.2
    answerLivingReward = -0.5
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question3e():
    answerDiscount = 0.8
    answerNoise = 0.01
    answerLivingReward = 2
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\reinforcement>python autograder.py -q q3
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 1-24 at 21:23:10

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 21:23:11

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

۴. تکرار ارزش ناهم‌زمان

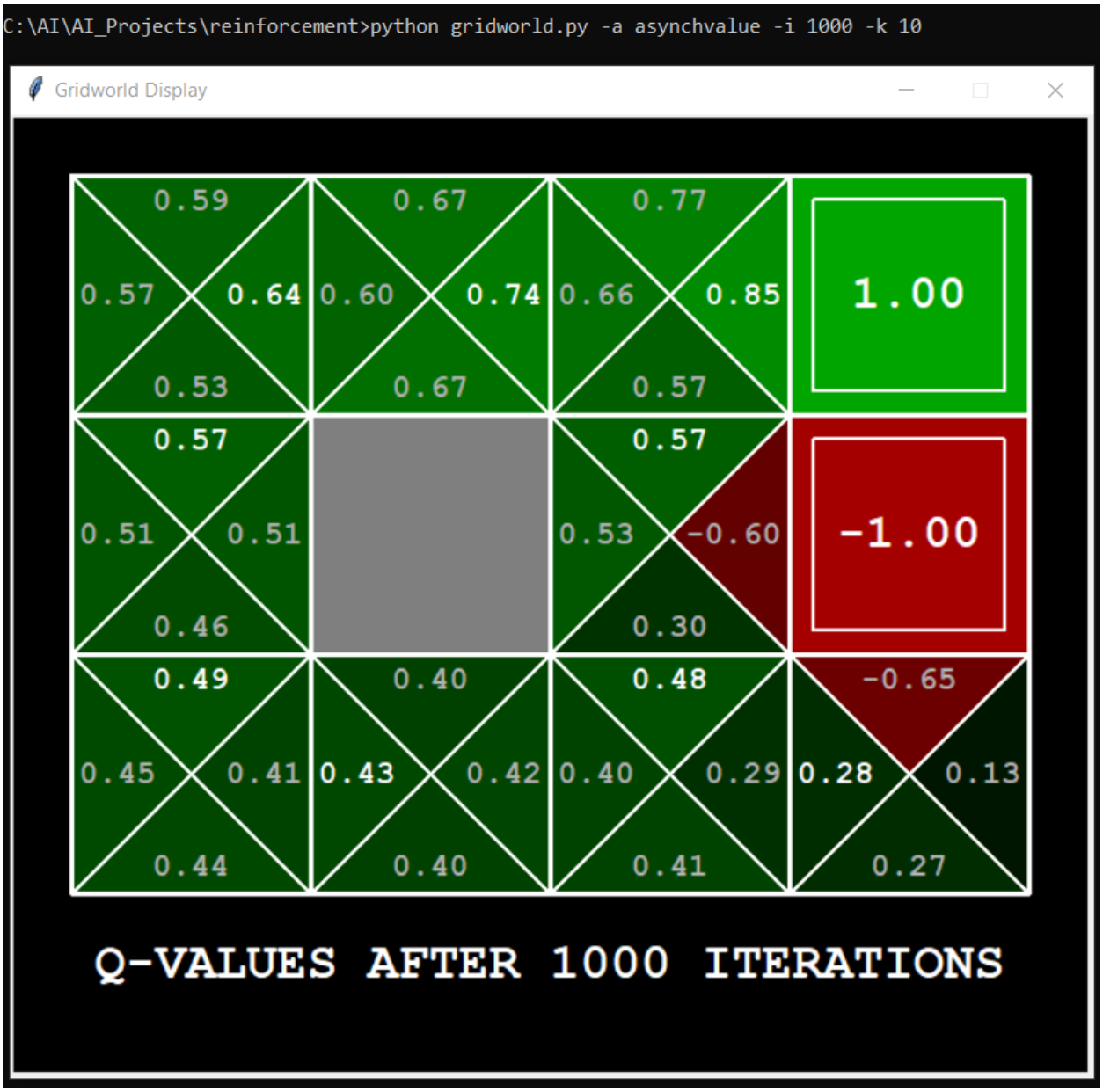
این بخش بسیار شبیه به بخش اول است اما همانطور که در صورت پروژه ذکر شد فقط یک حالت به روز رسانی می‌شود. برای این بخش باید تابع runValueIteration را در کلاس AsynchronousValueIterationAgent تغییر دهیم و تکمیل کنیم.

- **توضیح کد:**

این تابع بسیار شبیه runValueIteration در کلاس پدرش است؛ اما همانطور که گفته شد تنها یک بار نیاز است موقعیت را بگیریم زیرا هر بار برای یک موقعیت اجرا می‌شود، پس ابتدا موقعیت‌ها را دریافت کرده و هر بار که حلقه تکرار میشود فقط یکی از آن‌ها را بررسی میکنیم که برای این کار نیاز است باقی‌مانده‌ی شماره‌ی پیمایش را بر تعداد کل موقعیت‌ها محاسبه کنیم تا شماره‌ی موقعیت به دست آمده و سپس اگر این موقعیت ترمینال نبود با استفاده از تابعی که برای بهترین مقدار Q در کلاس پدر پیاده‌سازی کردیم ارزش را برای این موقعیت به روز رسانی کنیم.

```
def runValueIteration(self):
    "*** YOUR CODE HERE ***"
    # The only difference between this part and runValueIteration in
ValueIterationAgent
    # is that in this part value should be compute for only one state,
    # so iteration should be on states(each time one state)
    s = self.mdp.getStates()
    for each in range(self.iterations):
        # state_num defines the value should be updated for which state
        state_num = each % len(s)
        state = s[state_num]
        if not self.mdp.isTerminal(state):
            updatedQValue = -999
            self.values[state] = self.calculateBestQValue(state,
updatedQValue)
```

- تصاویری از خروجی این بخش:



- **نتیجه‌ی خروجی autograder:**

```
C:\AI\AI_Projects\reinforcement>python autograder.py -q q4
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 1-24 at 21:53:53

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 21:53:53

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

۵. تکرار ارزش الویت‌بندی‌شده:

این قسمت دقیقا مطابق دستور پروژه پیاده شده است و توضیحات دستور پروژه و نکات ذکر شده را با تمامی جزئیات در پیاده‌سازی این بخش در نظر گرفتیم و تلاش بر این است که به روز رسانی‌های مقادیر حالت را به سمتی متمرکز کنیم که تحنمتات سیاست‌ها را تغییر دهیم. برای این بخش هم نیاز به تکمیل تابع runValueIteration است اما این بار در کلاس فرزند به نام PrioritizedSweepingValueIterationAgent.

- **توضیح کد:**

بخش اول تابع به یافتن اجداد اختصاص داه شده است و دقیقا همانطور که در کامنت توضیح داده شده است عمل کردیم؛ به این صورت که روی تمامی موقعیت‌ها پیمایش انجام دادیم و با بررسی تمامی اکشن‌های ممکن در آن موقعیت، احتمال رسیدن به تمامی موقعیت‌هایی که با انجام عمل مورد نظر در استیت فعلی می‌توانیم دست‌یابی کنیم را مورد بررسی قرار دادیم و با توجه به این نکته که اگر احتمال رسیدن به موقعیت 0 نباشد یعنی راهی بین موقعیت فعلی موقعیت ثانویه‌ی مورد بررسی هست و این موضوع این مفهوم را می‌رساند که در گراف مورد بررسی ما آنها رابطه‌ی والد فرزندی دارند اجداد هر موقعیتی را همانطور که در کد مشاهده میکنید تکمیل کردیم.

```
def runValueIteration(self):
    "*** YOUR CODE HERE ***"
    # predecessors is a dictionary where each key is a state and the value is
```

```
a set
    # containing all the predecessors of that state
    # Duplication is controlled here
    predecessors = collections.defaultdict(set)

for state in self.mdp.getStates():
    for action in self.mdp.getPossibleActions(state):
        for nextState, probability in
self.mdp.getTransitionStatesAndProbs(state, action):
            # Only the more than zero probability shows that
            # by this node reaching next considered node is reachable
            # So only nonzero probability of being reached should be added
            if probability > 0:
                predecessors[nextState].add(state)
```

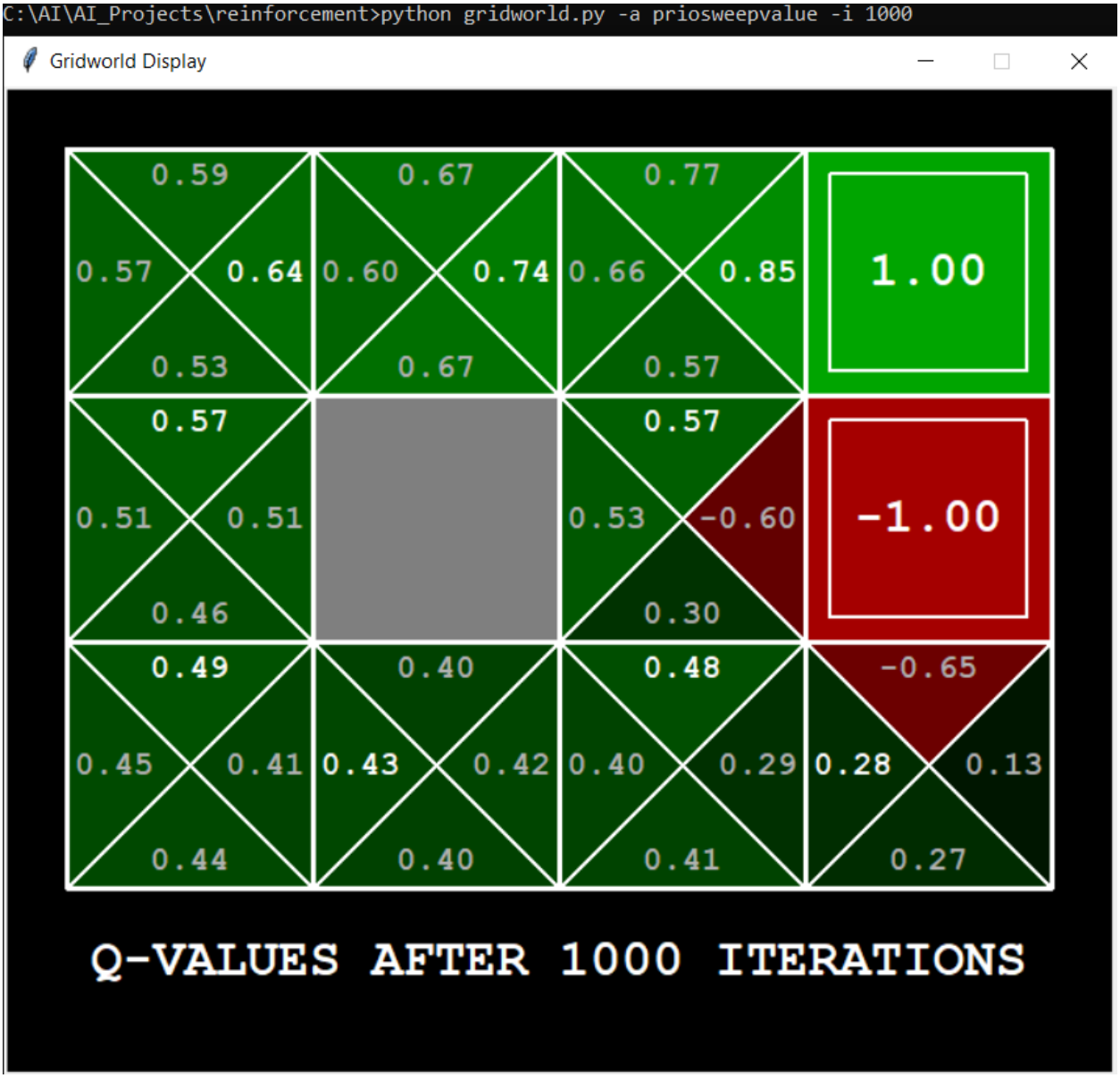
در بخش بعدی دقیقا مطابق توضیح دستور پروژه و با در نظر گرفتن این موضوع که صف اولیت ما از نوع min-heap است، با استفاده از تابع مجاسبه‌ی بهترین مقدار Q در کلاس پدر، به تکمیل و پر کردن این صف اولویت با استفاده از diff- به دست آمده برای هر موقعیتی که موقعیت ترمینال نیست پرداختیم.

```
minHeap = util.PriorityQueue()
for s in self.mdp.getStates():
    if not self.mdp.isTerminal(s):
        diff = abs(self.calculateBestQValue(s, -9999) - self.values[s])
        # Because the priority queue is a min-heap, so diff should be
negative
        minHeap.push(s, -diff)
```

در بخش آخر این تابع اگر هیپ ما خالی بود اجرا به پایان می‌رسد در غیر این صورت استیتی که با توجه به صف اواین باید بررسی کنیم را مورد بررسی قرار می‌دهیم و دقیقا مطابق دستور پروژه با محاسبه‌ی diff که قدر مطلق تفاضل بین مقدار فعلی حالت s که در self.values نگه داری می شود و بیشترین مقدارQ ممکن از حالت s است را برای تمامی اجداد این موقعیت محاسبه میکنیم و هر بار بررسی میکنیم تا اگر بزرگتر از تتا بود هیپ ما به روزرسانی شود.

```
for each in range(self.iterations):
    # If the priority queue is empty, then terminate.
    if minHeap.isEmpty():
        break
    else:
        state = minHeap.pop()
        self.values[state] = self.calculateBestQValue(state, -9999)
        for predecessor in predecessors[state]:
            diff = abs(self.calculateBestQValue(predecessor, -9999) -
self.values[predecessor])
            if diff > self.theta:
                minHeap.update(predecessor, -diff)
```

- تصاویری از خروجی این بخش:



- **نتیجه‌ی خروجی autograder:**

```
C:\AI\AI_Projects\reinforcement>python autograder.py -q q5
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 1-24 at 22:12:54

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 22:12:54

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

۶. یادگیری Q:

در این بخش پروژه باید با تغییر ۴ تابع getQValue و computeValueFromQValues و update و computeActionFromQValues در کلاس QLearningAgent که در فایل qlearningAgents.py قرار دارد به پیاده‌سازی رابطه‌ی زیر بپردازیم.

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

- **توضیح کد:**

تابع update:

در این تابع ابتدا بررسی میشود که آیا این استیت قبلا بررسی شده است یا نه و اگر بررسی نشده بود برای آن نیاز به یک شمارنده از نوع دیکشنری داریم.

سپس دقیقا مانند فرمول عمل می‌کنیم و با توجه به گاما، پاداش ابتدا سمپل را حساب میکنیم. سپس خط دوم فرمول را اجرا کرده و مقدار کیو را برای این موقعیت با اجرای اکشنی که در ورودی تابع گرفته است به آپدیت می‌کنیم.

```
def update(self, state, action, nextState, reward):
    """
```

```
      The parent class calls this to observe a
      state = action => nextState and reward transition.
      You should do your Q-Value update here

      NOTE: You should never call this function,
      it will be called on your behalf
    """
    "*** YOUR CODE HERE ***"
    # This state has not been visited yet
    if state not in self.Qvalues:
        self.Qvalues[state] = util.Counter()

    # Update sample and Q based on formula
    newSample = reward + self.discount *
self.computeValueFromQValues(nextState)
    newQ = (1 - self.alpha) * self.Qvalues[state][action] + self.alpha *
newSample
    self.Qvalues[state][action] = newQ
```

تابع computeActionFromQValues:

در این تابع به دنبال بهترین عمل می‌گردیم و برای انتخاب آن از تابع رندوم استفاده می‌کنیم. همانطور که در کد مشاهده میکنید و کامنت‌ها گویا هستند برای موقعیتی که ورودی تابع است به این صورت به دنبال بهترین عمل می‌گردیم که با پیدایش بر روی اکشن‌هایی که در آن موقعیت می‌توانیم انجام دهیم بررسی می‌کنیم که بیشترین مقدار Q را دارد یا نه تا به لیستی که برای جمع‌آوری بهترین اعمال داریم اضافه کنیم و در آخر اگر این لیست خالی نبود، یکی از اعمال را به طور رندوم انتخاب می‌کنیم.

```
def computeActionFromQValues(self, state):
    """
      Compute the best action to take in a state.  Note that if there
      are no legal actions, which is the case at the terminal state,
      you should return None.
    """
    "*** YOUR CODE HERE ***"
    bestActions = []
    for action in self.getLegalActions(state):
        # check if this action has the max Q value
        if self.getQValue(state, action) ==
self.computeValueFromQValues(state):
            bestActions.append(action)
    if len(bestActions) > 0:
        return random.choice(bestActions)
    return None
```

تابع computeValueFromQValues:

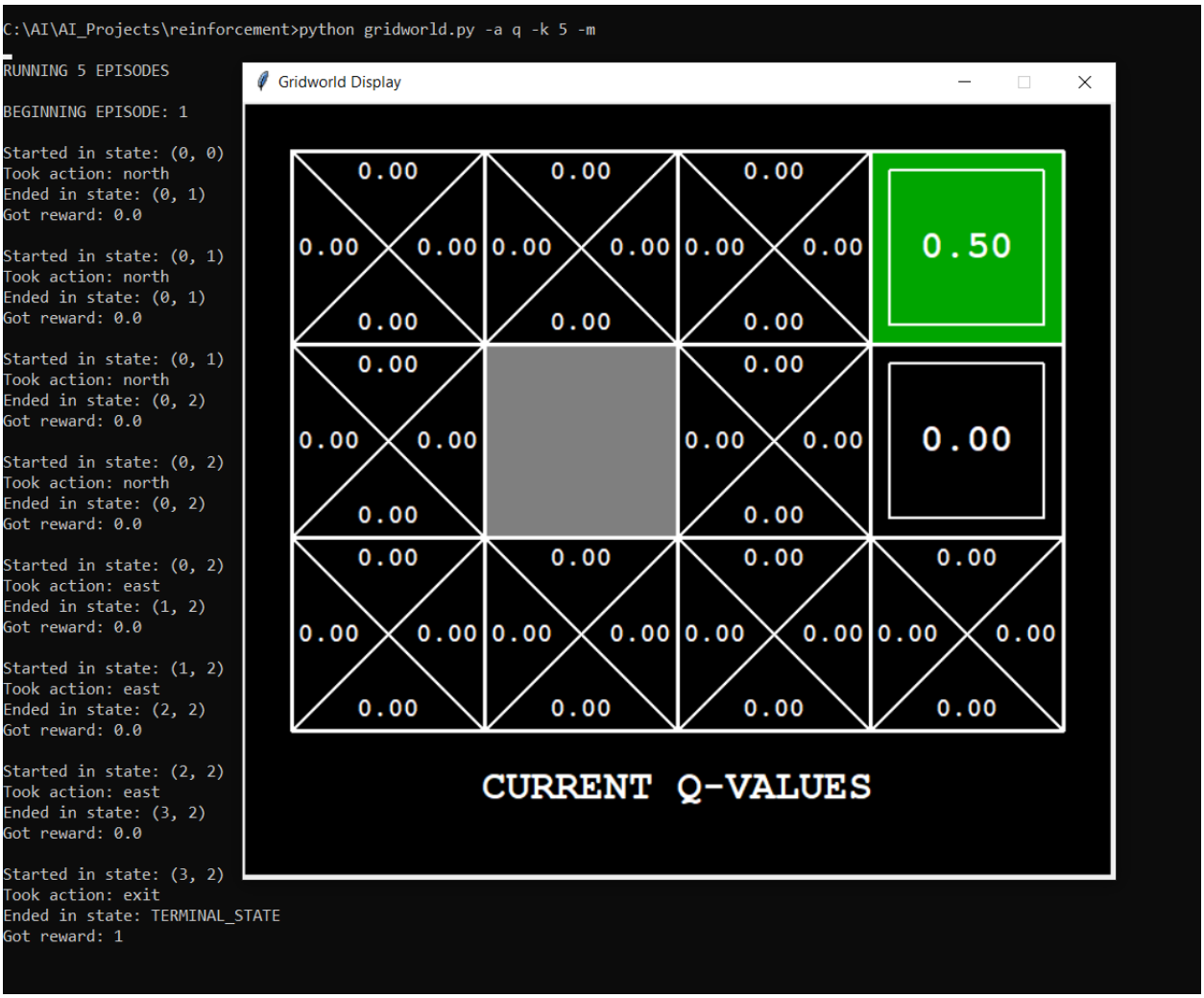
در این تابع به دنبال این هستیم تا عملی که بیشترین مقدار Q را دارد برای موقعیت فعلی که در آن هستیم پیدا کنیم و اگر این استست هیچ اکشنی نداشت مقدار 0 را بر می‌گردانیم.

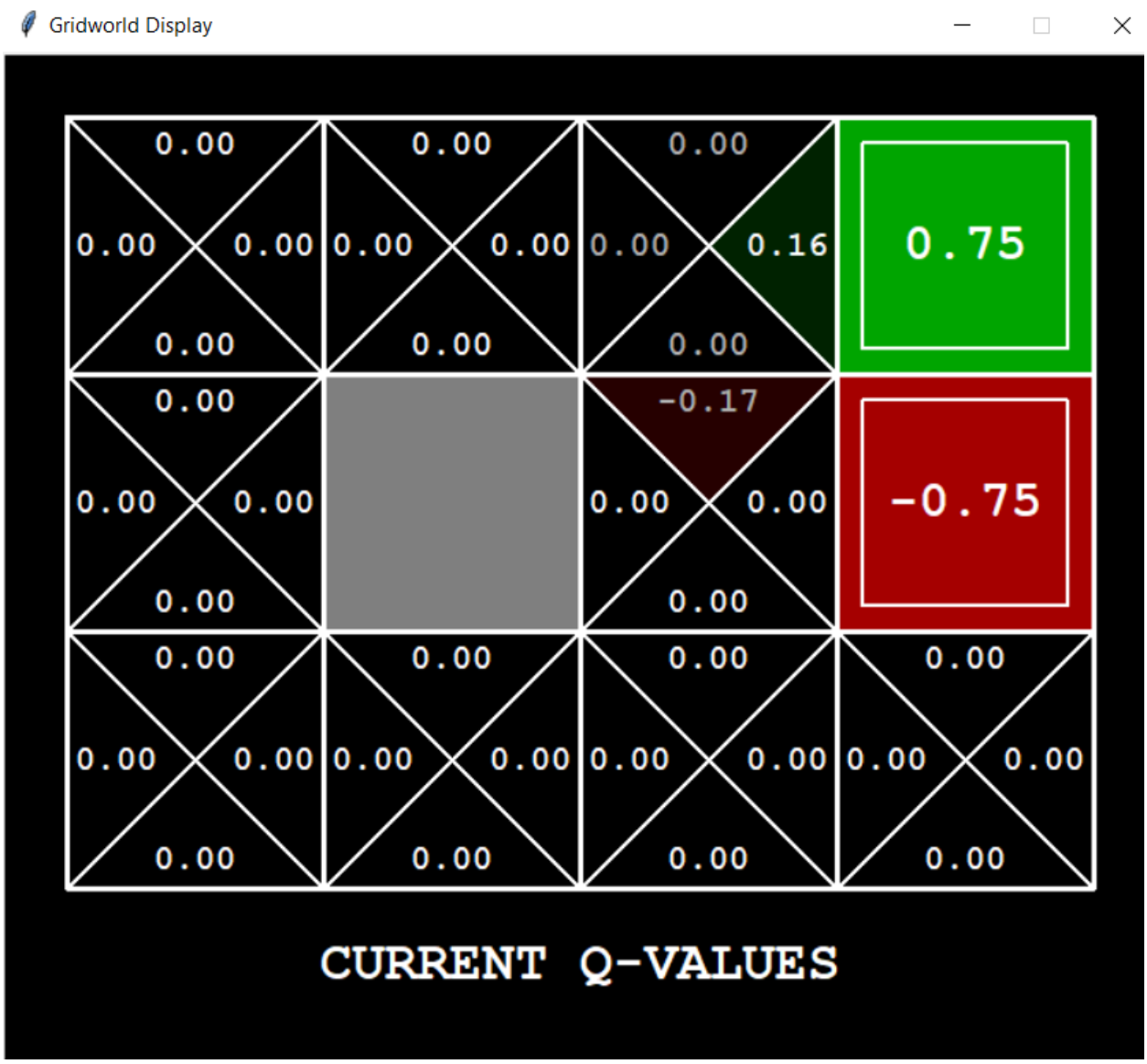
```
def computeValueFromQValues(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"
    # based on formula
    maxQ = -9999
    flag = 0
    for action in self.getLegalActions(state):
        maxQ = max(self.getQValue(state, action), maxQ)
        flag += 1
    if flag:
        return maxQ
    return 0
```

تابع getQValue:

یک تابع getter بسیار ساده است که با گرفتین موقعیت و عمل مورد نظر اگر این موقعیت در Qvalueها موجود بود مقدار QValue را برمیگرداند، در غیر این صورت خروجی آن 0 است.

- تصاویری از خروجی این بخش:




Gridworld Display
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.16
0.75
0.00
0.00
0.00
-0.17
0.00
0.00
0.00
0.00
0.00
-0.75
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
CURRENT Q-VALUES

- **نتیجه‌ی خروجی autograder:**

```
C:\AI\AI_Projects\reinforcement>python autograder.py -q q6
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 1-24 at 22:37:29

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 22:37:29

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

۷. اپسیلون حریصانه:

هدف اصلی اپسیلون این است که احتمال این که عامل عمل رندوم انجام دهد و به یادگیری بپردازد و یا این که از اعمال گذشته استفاده کند را مشخص کند. برای این بخش تابع getAction را تکمیل کردیم.

- **توضیح کد:**

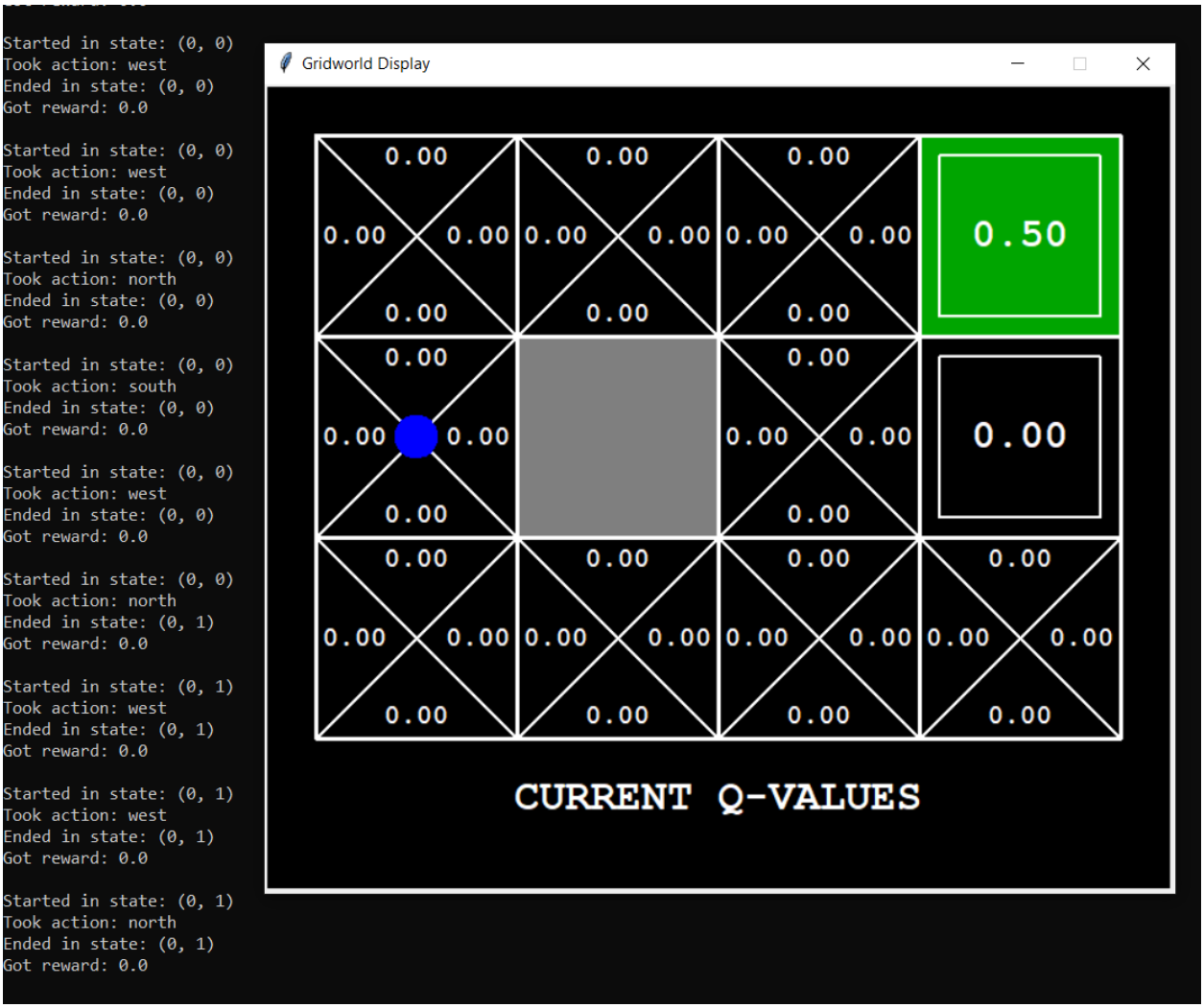
این تابع با گرفتن موقعیت و با استفاده اگر عمل‌های قانونی برای هر استیت به انتخاب اکشن با استفاده از نکاتی که گفته شد می‌پردازد.

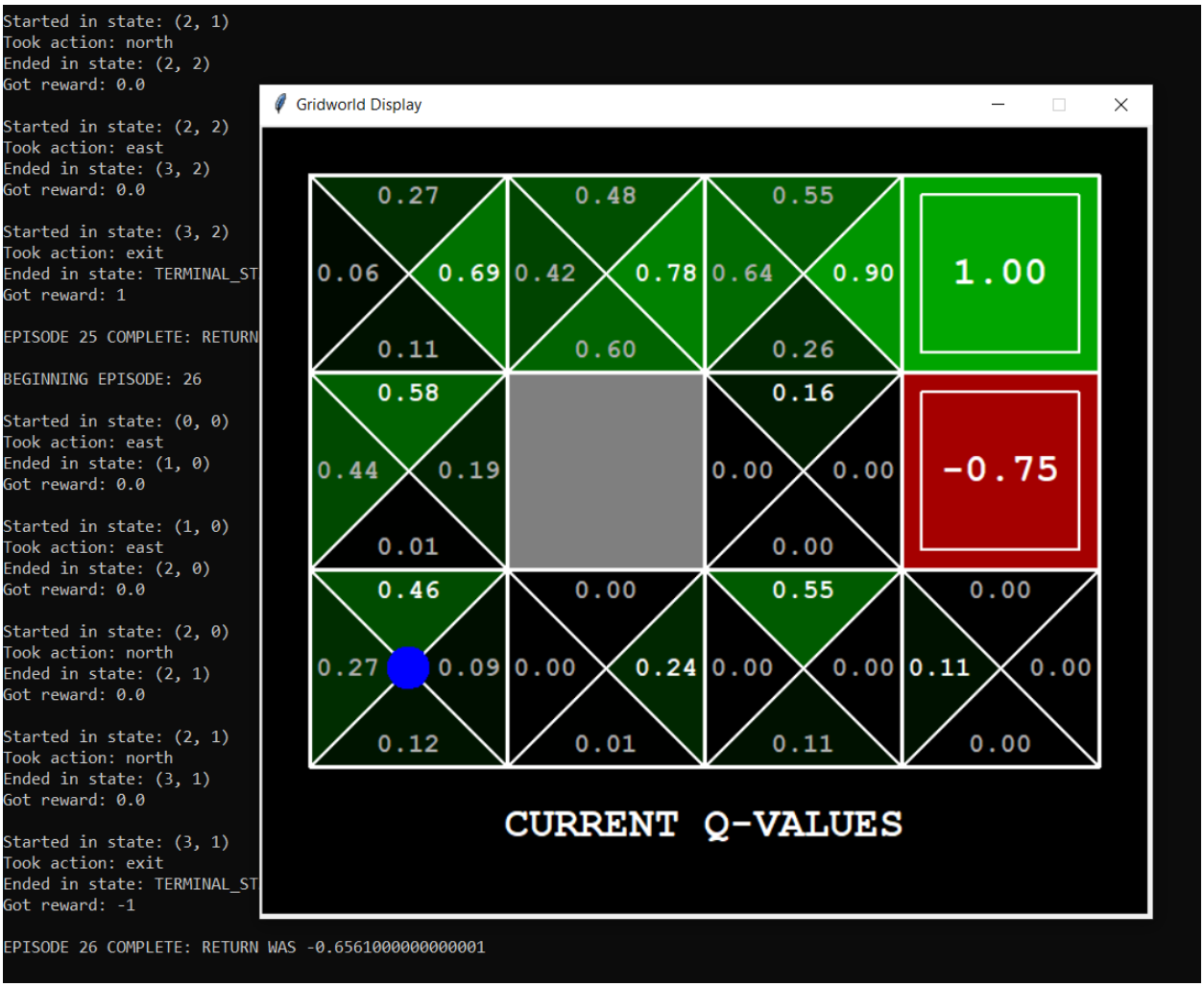
اگر عملی موجود بود با استفاده از flipCoin و احتمال اپسیلون به انتخاب عمل رندوم میپردازد تا یادگیری داشته باشد و با احتمال 1 منهای اپسیلون از computeActionFromQValues بهترین عمل را بر اساس QValue دریافت می‌کند و در نهایت عمل انتخاب شده را برمیگرداند. واضح است که اگر هیچ عمل قانونی‌ای وجود نداشته باشد خروجی این تابع None خواهد بود.

```
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
    "*** YOUR CODE HERE ***"
    # Check if there is any legal action
    if legalActions:
        # Use flipCoin to manage probability epsilon
        # A random legal action is chosen with probability epsilon
        if util.flipCoin(self.epsilon):
            return random.choice(legalActions)
        # the current best Q-Value is chosen with probability 1 - epsilon
        else:
            return self.computeActionFromQValues(state)
    # If there is not any legal action returns None,
    # otherwise returns chosen action
    return None
```

- تصاویری از خروجی این بخش:





- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\reinforcement>python autograder.py -q q7
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 1-24 at 22:58:49

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 22:58:52

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

۸. بررسی دوباره‌ی عبور از پل:

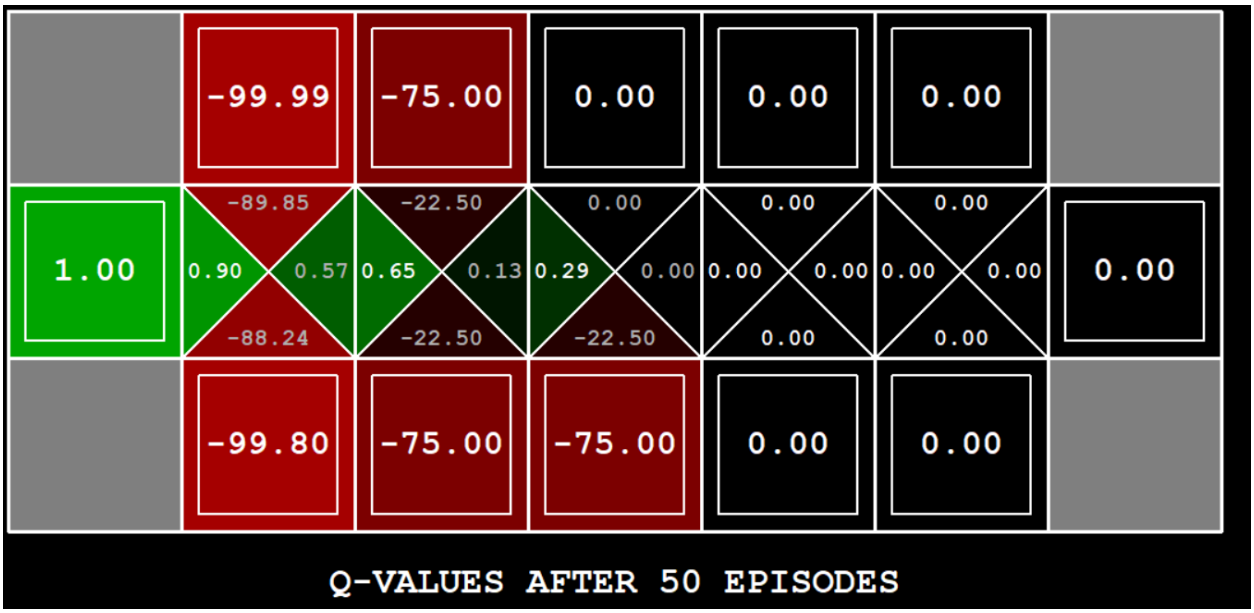
همانطور که در قسمت قبل هم توضیح داده شد اپسیلن احتمال یادگیری و رندوم عمل کردن یا استفاده از تجربه‌های پیشین را مشخص میکند. هر چقدر نزدیک به ۱ باشد یادگیری بیشتر و هر چقدر نزدیک به ۰ باشد احتمال استفاده از تجربیات گذشته بیشتر است. همچنین خانه‌ی اولیه پاداشی برابر 1+ دارد بنابراین در اپیزودهای اولیه به سمت خانه‌ی شروع تمایل پیدا میشود و سیاست بهینه شکل میگیرد و حتی اگر اپسیلون برابر ۱ هم باشد احتمالا رسیدن به پاداش بزرگتر بسیار بسیار کم است. اگر نرخ یادگیری را هم تغییر دهیم فایده‌ای ندارد و باز هم نمیتوانیم خانه‌های نهایی را بررسی و کاوش کنیم. بنابراین نتیجه‌ی کلی به این شکل است که پاسخ این بخش not possible است.

این موارد در عکس‌های خروجی در بخش بعد مشخص است.

- **کد:**

```
    def question8():
    return "NOT POSSIBLE"
    # If not possible, return 'NOT POSSIBLE'
```

تصاویر خروجی:




-100.00
-98.44
-50.00
0.00
0.00
1.00
-90.00
0.90
0.71
-90.00
-80.16
0.79
0.11
-86.84
0.00
0.14
0.00
-22.50
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
-100.00
-99.61
-75.00
-50.00
0.00
Q-VALUES AFTER 100 EPISODES


-100.00
-98.44
-50.00
0.00
0.00
1.00
0.90
0.79
0.14
0.00
0.00
0.00
-100.00
-99.61
-75.00
-50.00
0.00
VALUES AFTER 100 EPISODES

۹. اپسیلون حریصانه:

همانطور که در دستور پروژه هم گفته شده با تکمیل بخش هفتم، در واقع این بخش هم انجام شده است.

- **تصاویری از خروجی این بخش:**

```
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Average Score: 500.6
Scores:        495.0, 503.0, 503.0, 503.0, 503.0, 503.0, 495.0, 499.0, 503.0, 499.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

- **نتیجه‌ی autograder:**

خروجی

```
Starting on 1-24 at 23:15:50

Question q9
==========

Beginning 2000 episodes of Training
Reinforcement Learning Status:
        Completed 100 out of 2000 training episodes
        Average Rewards over all training: -510.09
        Average Rewards for last 100 episodes: -510.09
        Episode took 0.75 seconds
Reinforcement Learning Status:
        Completed 200 out of 2000 training episodes
        Average Rewards over all training: -511.44
        Average Rewards for last 100 episodes: -512.78
        Episode took 1.29 seconds
Reinforcement Learning Status:
        Completed 300 out of 2000 training episodes
        Average Rewards over all training: -474.83
        Average Rewards for last 100 episodes: -401.61
        Episode took 1.36 seconds
Reinforcement Learning Status:
        Completed 400 out of 2000 training episodes
        Average Rewards over all training: -443.94
        Average Rewards for last 100 episodes: -351.28
        Episode took 1.41 seconds
Reinforcement Learning Status:
        Completed 500 out of 2000 training episodes
        Average Rewards over all training: -425.37
        Average Rewards for last 100 episodes: -351.08
        Episode took 1.49 seconds
Reinforcement Learning Status:
        Completed 600 out of 2000 training episodes
        Average Rewards over all training: -392.55
        Average Rewards for last 100 episodes: -228.44
        Episode took 1.58 seconds
Reinforcement Learning Status:
        Completed 700 out of 2000 training episodes
        Average Rewards over all training: -361.91
        Average Rewards for last 100 episodes: -178.11
        Episode took 1.48 seconds
Reinforcement Learning Status:
        Completed 800 out of 2000 training episodes
        Average Rewards over all training: -345.18
        Average Rewards for last 100 episodes: -228.07
        Episode took 1.55 seconds
Reinforcement Learning Status:
        Completed 900 out of 2000 training episodes
        Average Rewards over all training: -331.22
        Average Rewards for last 100 episodes: -219.49
```

```
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Reinforcement Learning Status:
        Completed 100 test episodes
        Average Rewards over testing: 500.68
        Average Rewards for last 100 episodes: 500.68
        Episode took 1.67 seconds
Average Score: 500.68
Scores:        503.0, 503.0, 499.0, 499.0, 503.0, 503.0, 503.0, 503.0, 495.0, 503.0, 499.0, 503.0, 503.0, 503.0, 503.0, 503.0, 495.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 503.0, 499.0, 495.0, 503.0, 503.0,
495.0, 495.0, 495.0, 499.0, 499.0, 503.0, 503.0, 503.0, 503.0, 495.0, 499.0, 503.0, 495.0, 499.0, 503.0, 503.0, 503.0, 503.0, 499.0, 503.0, 495.0, 503.0, 503.0, 503.0, 503.0, 499.0, 503.0, 499.0, 503.0, 503.0,
99.0, 503.0, 503.0, 503.0, 503.0, 503.0, 495.0, 499.0, 503.0, 495.0, 499.0, 495.0, 495.0, 503.0, 495.0, 503.0, 499.0, 503.0, 503.0, 503.0, 495.0, 495.0, 503.0, 503.0, 499.0, 503.0, 503.0, 495.0, 499.0, 503.0, 5
3.0, 503.0, 499.0, 503.0, 503.0, 503.0, 499.0, 499.0, 495.0, 499.0, 499.0, 503.0
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,
in, 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, W
n, 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 --fixRandomSeed
***     100 wins (1 of 1 points)
***         Grading scheme:
***          < 70:  0 points
***         >= 70:  1 points

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


Finished at 23:16:23

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

۱۰. Q-learning تقریبی:

در این بخش همانطور که در دستور کار ذکر شده است یک عامل Q-learning پیاده شده است که وزن ویژگی‌های حالت‌ها را یاد میگرید. برای این بخش به تکمیل تابع getValue و update پرداختیم و روابط زیر را پیاده‌سازی کردیم.

Q-learning تقریبی:

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

و

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

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

- **توضیح کد:**

تابع getQValue:

این تابع دقیقا باید همان موردی که در توضیح آن نوشته شده است یعنی حاصل ضرب featureهای این موقعیت با انجام عمل مورد نظر در وزن را برگرداند. که در روابط بالا اولین رابطه حاصل جمع خروجی‌های همین تابع است.

تابع update:

همانطور که در دستور پروژه و توضیح خود کد هم آمده است، این تابع برای به روزرسانی وزن‌ها بر اساس transition میباشد. برای محاسبه‌ی difference که در به روزرسانی وزن‌ها هم کاربرد دارد، همانطور که در روابط نشان داده شد نیاز به ماکسیمم مقدار Q برای استیت بعدی داریم تا در گاما ضرب شود و سپس مقدار Q برا این موقعیت و اکشن از آن کم شود و با پاداش جمع شود. این کار در دو خط بالای حلقه انجام شده است.

سپس برای به روز رسانی تمامی وزن‌ها وارد حلقه می‌شویم و معادل فرمول عمل میکنیم.

```
•       def getQValue(self, state, action):
    """
      Should return Q(state,action) = w * featureVector
      where * is the dotProduct operator
    """
    "*** YOUR CODE HERE ***"
    return self.featExtractor.getFeatures(state, action)*self.weights

def update(self, state, action, nextState, reward):
    """
       Should update your weights based on transition
    """
    "*** YOUR CODE HERE ***"
    # every thing in this part is based on formula
    extractedFeatures = self.featExtractor.getFeatures(state, action)
    maxQValue = self.discount * self.computeValueFromQValues(nextState)
    difference = maxQValue - self.getQValue(state, action) + reward

    for feature in extractedFeatures:
        self.weights[feature] += self.alpha * difference *
extractedFeatures[feature]
```

- تصاویری از خروجی این بخش:

```
C:\AI\AI_Projects\reinforcement>python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid
Beginning 2000 episodes of Training
Reinforcement Learning Status:
        Completed 100 out of 2000 training episodes
        Average Rewards over all training: -510.28
        Average Rewards for last 100 episodes: -510.28
        Episode took 1.31 seconds
Reinforcement Learning Status:
        Completed 200 out of 2000 training episodes
        Average Rewards over all training: -510.92
        Average Rewards for last 100 episodes: -511.56
        Episode took 1.99 seconds
Reinforcement Learning Status:
        Completed 300 out of 2000 training episodes
        Average Rewards over all training: -494.84
        Average Rewards for last 100 episodes: -462.67
        Episode took 2.31 seconds
Reinforcement Learning Status:
        Completed 400 out of 2000 training episodes
        Average Rewards over all training: -466.92
        Average Rewards for last 100 episodes: -383.17
        Episode took 2.85 seconds
Reinforcement Learning Status:
        Completed 500 out of 2000 training episodes
        Average Rewards over all training: -411.59
        Average Rewards for last 100 episodes: -190.26
        Episode took 2.88 seconds
Reinforcement Learning Status:
        Completed 600 out of 2000 training episodes
        Average Rewards over all training: -386.30
        Average Rewards for last 100 episodes: -259.88
        Episode took 2.91 seconds
Reinforcement Learning Status:
        Completed 700 out of 2000 training episodes
        Average Rewards over all training: -363.79
        Average Rewards for last 100 episodes: -228.73
        Episode took 2.99 seconds
Reinforcement Learning Status:
        Completed 800 out of 2000 training episodes
        Average Rewards over all training: -335.74
        Average Rewards for last 100 episodes: -139.33
        Episode took 3.16 seconds
```

```
Reinforcement Learning Status:
        Completed 1600 out of 2000 training episodes
        Average Rewards over all training: -142.64
        Average Rewards for last 100 episodes: 227.29
        Episode took 2.91 seconds
Reinforcement Learning Status:
        Completed 1700 out of 2000 training episodes
        Average Rewards over all training: -117.92
        Average Rewards for last 100 episodes: 277.58
        Episode took 2.96 seconds
Reinforcement Learning Status:
        Completed 1800 out of 2000 training episodes
        Average Rewards over all training: -99.31
        Average Rewards for last 100 episodes: 217.21
        Episode took 2.98 seconds
Reinforcement Learning Status:
        Completed 1900 out of 2000 training episodes
        Average Rewards over all training: -82.10
        Average Rewards for last 100 episodes: 227.66
        Episode took 2.83 seconds
Reinforcement Learning Status:
        Completed 2000 out of 2000 training episodes
        Average Rewards over all training: -63.14
        Average Rewards for last 100 episodes: 297.09
        Episode took 3.02 seconds
Training Done (turning off epsilon and alpha)
-----------------------------------------
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Average Score: 499.8
Scores:        499.0, 495.0, 499.0, 503.0, 499.0, 503.0, 499.0, 495.0, 503.0, 503.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\reinforcement>python autograder.py -q q10
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 1-25 at 0:07:08

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 0:07:09

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

- نتیجه‌ی نهایی autograder برای کل پروژه:

```
Finished at 0:08:26

Provisional grades
==================
Question q1: 4/4
Question q2: 1/1
Question q3: 5/5
Question q4: 1/1
Question q5: 3/3
Question q6: 4/4
Question q7: 2/2
Question q8: 1/1
Question q9: 1/1
Question q10: 3/3
------------------
Total: 25/25
```