

به نام خدا



دانشگاه صنعتی امیرکبیر

دانشکده‌ی مهندسی کامپیوتر

گزارش کار پروژه سوم

اروند درویش

9831137

1) تکرار ارزش

در این قسمت با پیاده سازی سه تابع `runValueIteration` ، `computeQValueFromValues` و `computeActionFromValues` سعی بر پیاده سازی معادله زیر داریم (برای بدست آوردن value در هر state نسبت به iteration قبل):

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

تابع `runValueIteration`:

```
def runValueIteration(self):
    # Write value iteration code here
    """*** YOUR CODE HERE ***"""
    for i in range(self.iterations):
        newVals = self.values.copy()
        for st in self.mdp.getStates():
            maxVal = -10000
            for act in self.mdp.getPossibleActions(st):
                tmp = self.getQValue(st, act)
                if tmp > maxVal:
                    maxVal = tmp
            if maxVal > -10000:
                newVals[st] = maxVal
        self.values = newVals
```

تابع `computeQValueFromValues`:

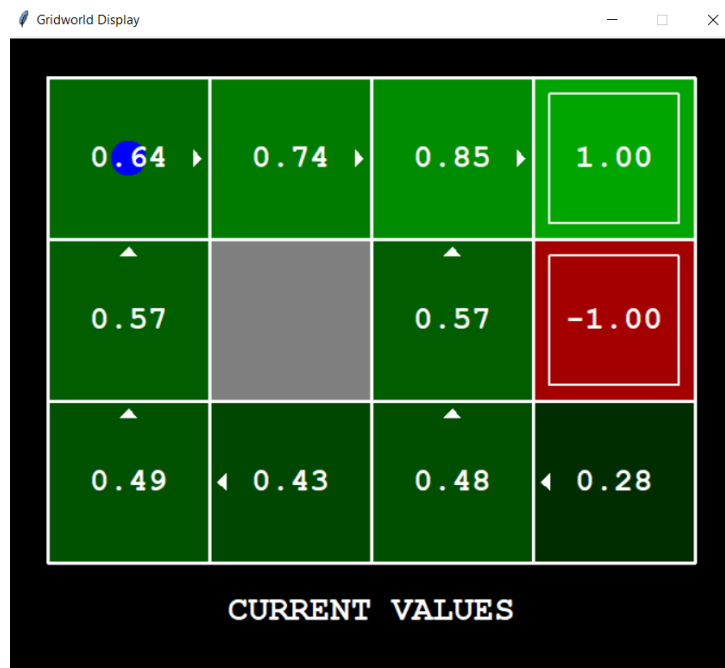
```
def computeQValueFromValues(self, state, action):
    """
    Compute the Q-value of action in state from the
    value function stored in self.values.
    """
    """*** YOUR CODE HERE ***"""
    nextStatesWithProbs = self.mdp.getTransitionStatesAndProbs(state, action)
    qVal = 0
    for nextSt, p in nextStatesWithProbs:
        qVal += p*(self.mdp.getReward(state, action, nextSt) + self.discount*self.getValue(nextSt))
    return qVal
```

تابع `computeActionFromValues`:

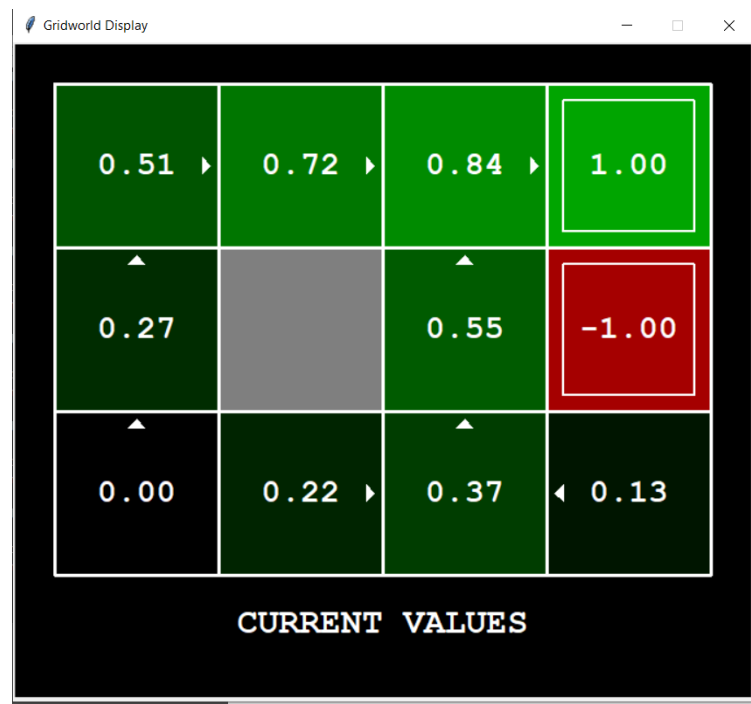
```
"""*** YOUR CODE HERE ***"""
actions = util.Counter()
for act in self.mdp.getPossibleActions(state):
    actions[act] = self.getQValue(state, act)
return actions.argmax()
```

خروجی تست ها :

python gridworld.py -a value -i 100 -k 10

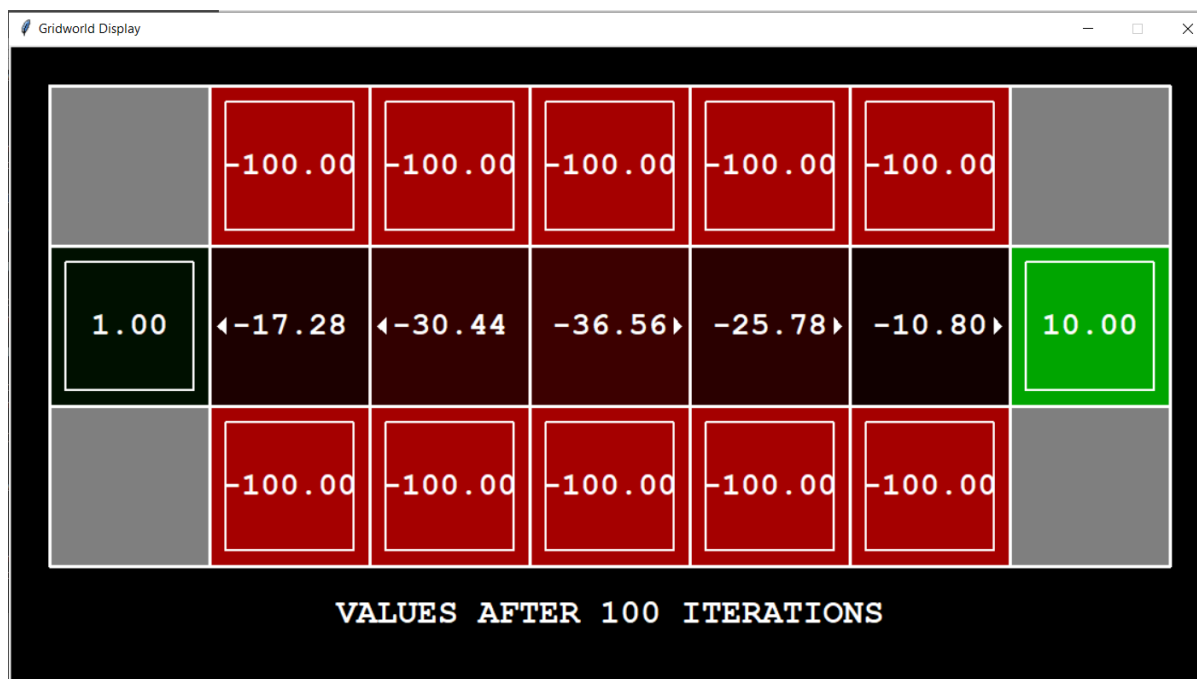


python gridworld.py -a value -i 5



2) تجزیه تحلیل عبور از پل

با قرار دادن مقدار صفر برای نویز در حقیقت ما محیط را قطعی (deterministic) کرده‌ایم و عامل ما میتواند از روی پل عبور کند. (به صورت قطعی)



3) سیاست ها

(a) اگر discount factor کوچک باشد، باعث می شود که عامل پاداش های نزدیک تر را ترجیح دهد و همچنین چون برای زندگی، یک پاداش منفی در نظر گرفته شده، عامل تلاش می کند که در کمترین زمان ممکن به حالت ترمینال برسد. همچنین چون نویز کم است عامل منطقی می داند که ریسک کند و از کنار صخره عبور کند چون احتمال خطا و نویز کم است.

(b) مانند بالا فقط با این تفاوت که اینبار نویز زیاد است و عامل دیگر ریسک نمی کند که از کنار صخره عبور کند.

(c) در این قسمت نیز، در صورتی که discount factor زیاد باشد، باعث می شود که پاداش های بزرگ تر ترجیح داده شوند. برای اینکه عامل از صخره حرکت کند و ریسک کند، با نویز کم و با پاداش زندگی منفی، عامل به این سمت حرکت خواهد کرد.

(d) همانند قسمت قبل، با این تفاوت که اینبار نویز زیاد است و عامل دیگر ریسک نمی کند که از کنار صخره عبور کند.

(e) در این حالت، در صورتی که پاداش زندگی بیشتر از پاداش های استیت های بازی باشد، عامل تلاش می کند که صرفاً زنده مانده و به استیت های نهایی داخل نشود.

4) تکرار ارزش ناهمزمان

در این قسمت همانند قسمت 1 باید value ها را آپدیت کنیم و دوباره تابع runValueIteration را بازنویسی کنیم با این تفاوت که به جای این که در هر iteration تمامی استیت ها آپدیت شوند، صرفاً یکی از استیت ها آپدیت می شود.

```
def runValueIteration(self):
    """*** YOUR CODE HERE ***"""
    statesNum = len(self.mdp.getStates())
    for i in range(self.iterations):
        state = self.mdp.getStates()[i % statesNum]
        if not self.mdp.isTerminal(state):
            maxVal = -10000
            for act in self.mdp.getPossibleActions(state):
                qValue = self.computeQValueFromValues(state, act)
                if qValue > maxVal:
                    maxVal = qValue
            self.values[state] = maxVal
```

خروجی تست ها :



```
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 22:23:58

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.
```

5) با اولویت

زمانی که value مربوط به یک استیت آپدیت می شود، این کار باعث می شود که value استیت هایی که از آن ها میتوان به این استیت رفت، تغییر کند. بنابراین برای اولویت دهی، باید به ازای هر استیت بدانیم که predecessors های آن استیت چیست. حال برای پیاده سازی runValueIteration در ابتدا یک priority queue تعریف می کنیم و سپس تمامی استیت های غیر نهایی را با توجه به اختلاف بین value فعلی و بیشترین مقداری که میتوانند به آن دست پیدا کنند که قرینه این مقدار اولویت را تشکیل می دهد به این صف اولویت اضافه می کنیم. در ادامه نیز با توجه به دفعات اجرا که iterations مشخص می کند، عنصر با اولویت بالاتر را از صف خارج می کنیم و value آن را آپدیت می کنیم و سپس predecessor های آن را با توجه به اختلاف بین مقدار فعلی values و بیشترین مقداری که میتواند به آن دست پیدا کند، مقدار اولویت آن استیت در صف را آپدیت می کنیم.

```

predecessors = {}
queue = util.PriorityQueue()
for state in self.mdp.getStates():
    if not self.mdp.isTerminal(state):
        for act in self.mdp.getPossibleActions(state):
            for nextStatesWithProbs in self.mdp.getTransitionStatesAndProbs(state, act):
                nextState = nextStatesWithProbs[0]
                if nextState not in predecessors:
                    predecessors[nextState] = set()
                predecessors[nextState].add(state)

for state in self.mdp.getStates():
    if not self.mdp.isTerminal(state):
        maxVal = -10000
        for act in self.mdp.getPossibleActions(state):
            tmp = self.computeQValueFromValues(state, act)
            if tmp > maxVal:
                maxVal = tmp
        diff = abs(maxVal - self.getValue(state))
        queue.update(state, -diff)

for i in range(self.iterations):
    if not queue.isEmpty():
        state = queue.pop()
        if not self.mdp.isTerminal(state):
            maxVal = -10000
            for act in self.mdp.getPossibleActions(state):
                tmp = self.computeQValueFromValues(state, act)
                if tmp > maxVal:
                    maxVal = tmp
            self.values[state] = maxVal

            for predecessor in predecessors[state]:
                maxVal = -10000
                for act in self.mdp.getPossibleActions(predecessor):
                    tmp = self.computeQValueFromValues(predecessor, act)
                    if tmp > maxVal:
                        maxVal = tmp
                diff = abs(maxVal - self.getValue(predecessor))
                if diff > self.theta:
                    queue.update(predecessor, -diff)

```


6) یادگیری Q

در متد `computeValueFromQValues` در ابتدا `action` های مجاز را در استیت مربوطه به دست می آوریم و سپس به بیشترین `QValue` را که می توان از این `state` به آن رسید، را به عنوان خروجی برمیگردانیم سپس برای پیاده سازی `computeActionFromQValues` همانند متد قبلی عمل می کنیم با این تفاوت که در اینجا `action` ای را که منجر به تولید مقدار ماکزیمم `QValue` شده است را به عنوان خروجی برمیگردانیم.

برای پیاده سازی متد `update` در ابتدا `sample` ورودی را به واسطه رابطه زیر محاسبه می کنیم :

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

سپس با توجه به مفهوم `Moving average` مقدار مربوط به `QValue` را آپدیت می کنیم :

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

7) اپسیلون حریصانه

```
# Pick Action
legalActions = self.getLegalActions(state)
action = None
"*** YOUR CODE HERE ***"
if util.flipCoin(self.epsilon):
    action = random.choice(legalActions)
else:
    action = self.getPolicy(state)
return action
```

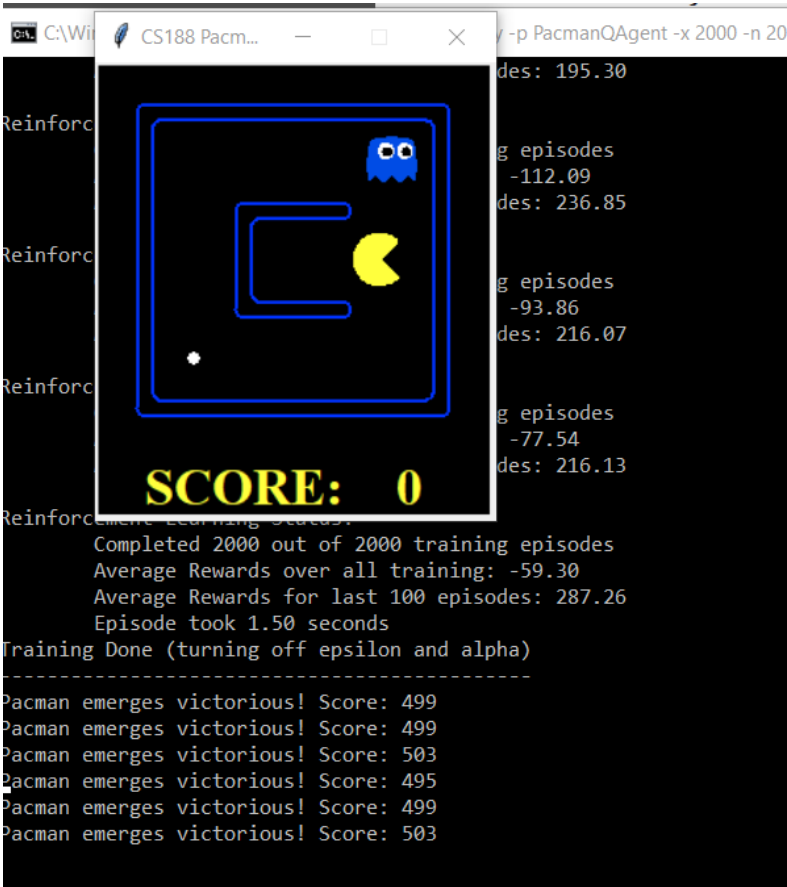
در اینجا در صورتی که action مجازی وجود داشته باشد، با استفاده از تابع flipcoin با احتمال epsilon مقدار true تولید می شود که در این حالت حرکت بعدی به صورت رندوم انتخاب می شود. در غیر اینصورت با توجه به سیاست موجود حرکت بعدی انتخاب می شود.

8) بررسی دوباره عبور از پل

در اینجا چون مقدار اپسیلون در طول یادگیری ثابت است اگر زیاد باشد باید استیت های جدیدتر و بیشتری را از نقشه کاوش کند و هرچه کمتر باشد خیلی کاوش نمیکند و به همان استیت های ابتدایی اکتفا میکند پس ما برای اینکه بخواهیم کل نقشه کاوش شود نیاز است تا اپسیلون را بیشترین مقدار ممکن بگذاریم ولی از آنجا که تکرار ما (iteration) کم است و فقط 50 بار است حتی با اپسیلون 1 نیز فرصت کاوش تمامی حالت ها پیدا نمی شود بنابراین پاسخ not possible می شود.

9) پکمن و یادگیری Q

در این قسمت تغییری در کد و تابع ها ایجاد داده نشده است و خروجی تست های این بخش به صورت زیر است :



The image shows a screenshot of a Pacman game window titled "CS188 Pacm..." and a terminal window titled "y-p PacmanQAgent -x 2000 -n 20". The game window displays a Pacman character (yellow) and a ghost (blue) on a black background with a blue maze. The score is 0. The terminal window shows the results of 2000 training episodes, including average rewards and the final score of 499.

```
C:\Win... CS188 Pacm... y-p PacmanQAgent -x 2000 -n 20
des: 195.30
g episodes
-112.09
des: 236.85
g episodes
-93.86
des: 216.07
g episodes
-77.54
des: 216.13
Reinforcement Learning Results:
Completed 2000 out of 2000 training episodes
Average Rewards over all training: -59.30
Average Rewards for last 100 episodes: 287.26
Episode took 1.50 seconds
Training Done (turning off epsilon and alpha)
-----
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
```