

به نام خدا



دانشگاه صنعتی امیرکبیر  
( پلی تکنیک تهران )

دانشکده مهندسی کامپیوتر

مبانی و کاربردهای هوش مصنوعی ترم پاییز ۱۴۰۰

پروژه سوم

مهلت تحویل 1 بهمن ۱۴۰۰

---

### مقدمه

در این پروژه شما value iteration و Q-learning را پیاده سازی خواهید کرد. شما عوامل خود را ابتدا در Gridworld آزمایش خواهید کرد، سپس آن‌ها را روی یک کنترلر ربات شبیه سازی شده (Crawler) و Pacman اعمال خواهید کرد.

مانند پروژه های قبلی، برای دیباگ و تست درستی الگوریتم‌های خود میتوانید دستور زیر را اجرا کنید:

```
python autograder.py
```

برای استفاده از autograder.py تنها برای یک سوال به صورت انحصاری می توانید از دستور زیر استفاده کنید:

```
python autograder.py -q q2
```

همچنین، می توان آن را برای یک تست خاص، به فرم زیر اجرا کرد:

```
python autograder.py -t test_cases/q2/1-bridge-grid
```

ساختار پروژه بصورت زیر است و کلیه فایل‌های مورد نیاز در فایل زیپ موجود در سامانه کورسز خواهد بود:

فایل‌هایی که باید ویرایش کنید:	
valueIterationAgents.py	یک عامل value iteration برای حل MDP های شناخته شده.
qlearningAgents.py	عامل های Q-learning برای Crawler، Gridworld، Pacman و .
analysis.py	فایلی برای قرار دادن پاسخ های شما به سوالات داده شده در پروژه.
فایل‌هایی که شاید بخواهید آن‌ها را ببینید:	
mdp.py	متدهایی را بر روی MDP های متداول تعریف می کند ..
learningAgents.py	کلاس های پایه ValueEstimationAgent و QLearningAgent را تعریف می کند که عامل های شما آن ها را گسترش خواهند داد.
util.py	ابزارهای کمکی، از جمله util.Counter، که مخصوصاً برای Q-learner ها قابل استفاده است.
gridworld.py	پیاده سازی Gridworld.
featureExtractors.py	کلاس هایی برای استخراج ویژگی ها در pair های (state, action). برای تقریب عامل Q-learning استفاده می شود (در qlearningAgents.py).
فایل‌هایی که می‌توانید آن‌ها را رد کنید:	
enviroment.py	کلاس abstract برای محیط های یادگیری تقویتی کلی. استفاده شده توسط gridworld.py.
graphicsGridworldDisplay.py	صفحه نمایش گرافیکی Gridworld
graphicsUtils.py	ابزارهای گرافیکی
textGridworldDisplay.py	پلاگین برای رابط متنی Gridworld.
crawler.py	کد crawler و تست. شما این را اجرا می کنید اما آن را ویرایش نمی کنید.

graphicsCrawlerDisplay.py	GUI برای ربات crawler
autograder.py	تصحیح کننده خودکار پروژه
testParser.py	Parse کردن تست های مصحح خودکار و فایل های راه حل
testClasses.py	کلاس های کلی تست خودکار
test_cases/	پوشه دربردارنده تست های مختلف برای هر سوال
reinforcementTestClasses.py	کلاس های تست خودکار پروژه سوم

**آنچه باید انجام دهید:**

شما باید بخش هایی از سه فایل **valueIterationAgents.py** و **qlearningAgents.py** و **analysis.py** را تکمیل کنید.

**لطفا سایر فایل ها را تغییر ندهید.**

همچنین گزارشی مختصر از نحوه پیاده سازی های انجام شده تهیه کرده و در نهایت این دو فایل را در سامانه بارگذاری خواهید کرد.

## MDP ها!

پس از بارگیری کد پروژه از سامانه کورسز و خارج کردن آن از حالت فشرده، می توانید Gridworld را در حالت کنترل دستی، که از کلیدهای جهت دار استفاده می کند، اجرا کنید:

```
python gridworld.py -m
```

طرح two-exit که پیش از این در کلاس دیدید، را خواهید دید. نقطه آبی عامل است. **توجه داشته باشید** که وقتی کلید بالا را فشار می دهید، عامل فقط در 80 درصد مواقع به سمت شمال حرکت می کند. زندگی یک مامور Gridworld چنین است!

شما می توانید بسیاری از جنبه های شبیه سازی را کنترل کنید. یک لیست کامل از گزینه های موجود با دستور زیر نمایش داده میشود:

```
python gridworld.py -h
```

عامل پیش فرض به صورت تصادفی حرکت می کند:

```
python gridworld.py -g MazeGrid
```

باید ببینید که عامل تصادفی در اطراف grid می چرخد تا زمانی که در یک خروجی بیفتد.

**نکته مهم:** Gridworld MDP به گونه ای است که ابتدا باید وارد یک حالت pre-terminal شوید (جعبه های دوگانه نشان داده شده در رابط کاربری گرافیکی) و سپس action ویژه «exit» را قبل از پایان اپیزود انجام دهید (در حالت ترمینال واقعی به نام **TERMINAL\_STATE**، که در رابط کاربری گرافیکی نشان داده نشده است). اگر یک اپیزود را به صورت دستی<sup>1</sup> اجرا کنید، به دلیل discount rate، ممکن است کل مقدار بازگشتی شما کمتر از حد انتظار شما باشد (d- برای تغییر دادن، ۰.۹ به طور پیش فرض).

---

<sup>1</sup> manually

به خروجی کنسول همراه با خروجی گرافیکی نگاه کنید (یا از t- برای همه متن استفاده کنید). در مورد هر انتقالی که عامل تجربه می کند به شما گفته می شود (برای خاموش کردن آن، از q- استفاده کنید).

مشابه با Pacman، موقعیت‌ها با مختصات دکارتی (x,y) نشان داده می‌شوند و هر آرایه‌ای با [x][y] نمایه‌سازی می‌شود، به طوری که «شمال» جهت افزایش y است. به‌طور پیش‌فرض، اکثر انتقال‌ها پاداش صفر دریافت می‌کنند. هر چند می‌توانید این را با گزینه پاداش زنده (-r) تغییر دهید.

## (۱) تکرار ارزش<sup>۲</sup> (۴ امتیاز)

معادله به روز رسانی حالت تکرار ارزش را به یاد بیاورید:

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

یک عامل تکرار ارزش در **ValueIterationAgent** بنویسید که تا حدی برای شما در **valueIterationAgents.py** مشخص شده است. عامل تکرار ارزش شما یک برنامه‌ریز آفلاین است و نه یک عامل یادگیری تقویتی. **training option** مربوط به تعداد تکرارهای تکرار ارزش است که باید در مرحله برنامه‌ریزی اولیه اجرا شود (**option -i**). **ValueIterationAgent** یک MDP در **constructor** می‌گیرد و تکرار ارزش را برای تعداد مشخصی از تکرارها قبل از اتمام اجرای **constructor**، اجرا می‌کند.

تکرار ارزش، تخمین‌های  $k$  مرحله‌ای مقادیر بهینه،  $V_k$  را محاسبه می‌کند. علاوه بر اجرای تکرار ارزش، متدهای زیر را برای **ValueIterationAgent** با استفاده از  $V_k$  پیاده‌سازی کنید:

- تابع **computeActionFromValues** که بهترین عمل را با توجه به تابع مقدار<sup>۳</sup> داده شده توسط **self.values** محاسبه می‌کند.

- تابع **computeQValueFromValues** که ورودی (state, action) را می‌گیرد و **Q-value** را برای action در state با توجه به تابع مقدار ذخیره شده در **self.values** محاسبه می‌کند.

<sup>۲</sup> Value Iteration

<sup>۳</sup> Value function

این اعداد همگی در رابط کاربری گرافیکی نمایش داده می شوند: مقادیر<sup>4</sup>، اعداد داخل مربع هستند. Q-value ها اعداد در ربع مربع و مقررات فلش های خارج از هر مربع هستند.

**مهم:** از نسخه "batch" تکرار ارزش استفاده کنید که در آن هر بردار  $V_k$  از یک بردار ثابت  $1 - V_k$  محاسبه می شود (مانند آنچه در اسلایدها و کلاس درس داشتید).

**راهنمایی:** شما می توانید به صورت اختیاری از کلاس **util.Counter** در **util.py** استفاده کنید که یک دیکشنری با مقدار پیش فرض صفر است. با این حال، مراقب **argMax** باشید: **argmax** واقعی که شما می خواهید ممکن است کلیدی باشد که در شمارنده نیست!

**توجه:** مطمئن شوید حالتی که یک state هیچ action ممکن در MDP ندارد را در پیاده سازی خود در نظر بگیرید. (به معنای این موضوع برای پاداش های آینده فکر کنید).

برای آزمایش پیاده سازی خود، autograder را اجرا کنید:

```
python autograder.py -q q1
```

دستور زیر **ValueIterationAgent** شما را لود می کند، که یک سیاست<sup>5</sup> را محاسبه و ۱۰ بار اجرا می کند. دکمه ای را روی صفحه کلید فشار دهید تا مقادیر، Q-value ها و شبیه سازی را مشاهده کنید. باید متوجه شوید که مقدار حالت شروع ( $V(\text{start})$ ) که می توانید آن را از رابط کاربری گرافیکی بخوانید) و میانگین پاداش تجربی حاصل (چاپ شده پس از اتمام ۱۰ دور اجرا) کاملاً نزدیک هستند.

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

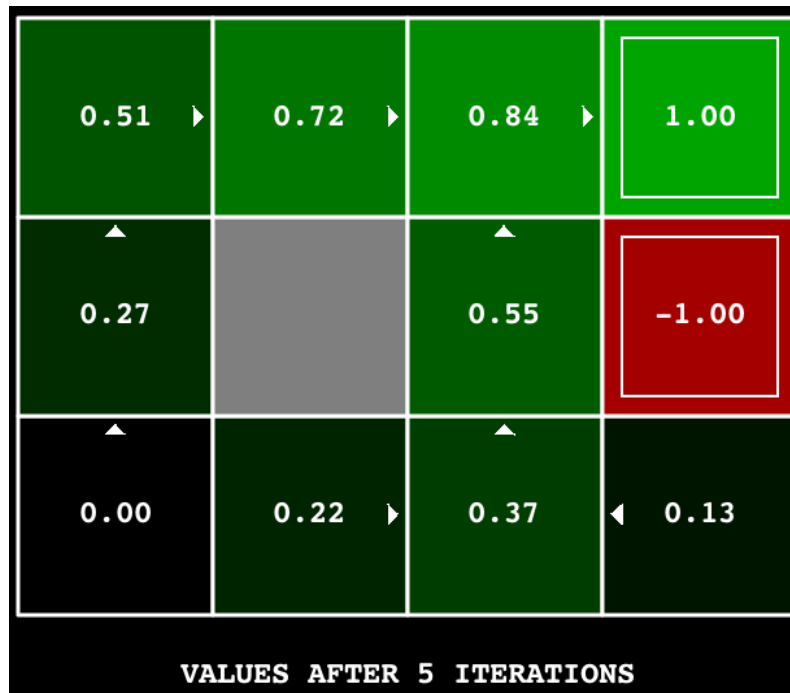
**راهنمایی:** در BookGrid پیش فرض، اجرای تکرار مقدار برای 5 تکرار باید این خروجی را به شما بدهد:

```
python gridworld.py -a value -i 5
```

---

<sup>4</sup> Values

<sup>5</sup> Policy



## ۲) تجزیه و تحلیل عبور از پل<sup>۶</sup> (۱ امتیاز)

**BridgeGrid** یک نقشه در grid world با حالت پایانه کم پاداش و حالت پایانه با پاداش بالا است که توسط یک «پل» باریک از هم جدا شده‌اند، که در دو طرف آن شکافی از پاداش منفی بالا وجود دارد. عامل نزدیک به حالت پاداش کم شروع می‌کند. با تخفیف پیش‌فرض ۰.۹ و نویز پیش‌فرض ۰.۲، سیاست بهینه<sup>۷</sup> از پل عبور نمی‌کند. **تنها یکی** از پارامترهای تخفیف و یا نویز را تغییر دهید تا سیاست بهینه باعث شود عامل برای عبور از پل تلاش کند. پاسخ خود را در قسمت **question2** در فایل **analysis.py** قرار دهید. (نویز به این اشاره دارد که هر چند وقت یک نماینده هنگام انجام یک عمل به یک حالت جانشین ناخواسته ختم می‌شود.) خروجی مقدارهای پیش‌فرض به صورت زیر است:

```
python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9
--noise 0.2
```

<sup>۶</sup> Bridge Crossing Analysis

<sup>۷</sup> optimal policy



### نحوه ارزیابی سوال دوم:

بررسی خواهد شد که شما فقط یکی از پارامترهای داده شده را تغییر داده اید و با این تغییر، یک عامل تکرار مقدار که به درستی پیاده سازی شده است، باید از پل عبور کند. برای بررسی پاسخ خود، autograder را اجرا کنید:

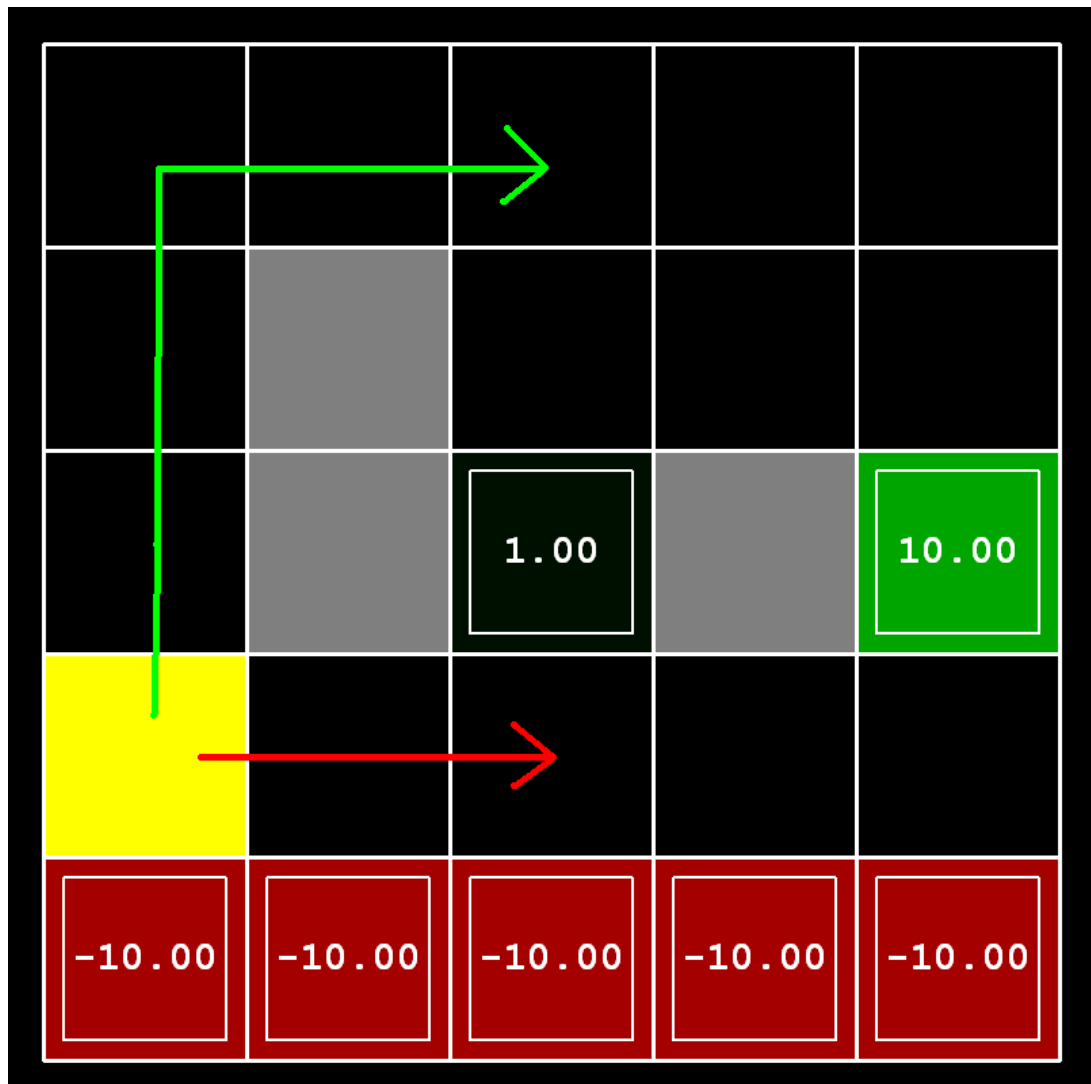
```
python autograder.py -q q2
```

### (۳) سیاست ها (۵ امتیاز)

نقشه ی DiscountGrid را که در زیر نشان داده شده است در نظر بگیرید. این دارای دو حالت پایانی با پاداش مثبت است. یک خروجی نزدیک با پاداش ۱+ و یک خروجی دور با پاداش ۱۰+. ردیف پایین grid شامل حالات پایانی با پاداش منفی است (نشان داده شده با رنگ قرمز) و به اصطلاح به این قسمت صخره می گوئیم. هر حالت در صخره دارای پاداش ۱۰- است. حالت شروع مربع زرد است. ما بین دو نوع مسیر تمایز قائل می شویم: (۱) مسیرهایی که "خطر صخره را به جان میخرند" و نزدیک ردیف پایین شبکه حرکت می کنند. این مسیرها کوتاهتر هستند، اما ریسک دریافت پاداش منفی زیادی دارند و با فلش قرمز در شکل زیر نشان داده شده اند. (۲) مسیرهایی که "از صخره اجتناب می کنند" و در امتداد



لبه بالایی grid حرکت می کنند. این مسیرها طولانی تر هستند، اما احتمال کمتری دارد که بازدهی منفی بزرگی را متحمل شوند. این مسیرها با فلش سبز رنگ در شکل زیر نشان داده شده اند.



در این سوال، تنظیمات پارامترهای تخفیف، نويز و پاداش زندگی<sup>8</sup> را برای این MDP انتخاب می کنید تا چند نوع مختلف از سیاست های بهینه را بدست آورید. تنظیم مقادیر پارامتر برای هر بخش باید این ویژگی را داشته باشد که اگر عامل شما سیاست بهینه خود را بدون ایجاد نويز دنبال کند، رفتار داده شده را نشان می دهد. اگر رفتار خاصی با هیچ تنظیمی از پارامترها به دست نیامد، با بازگرداندن رشته "NOT POSSIBLE"، ادعا کنید که این سیاست غیرممکن است.

در اینجا انواع سیاست های بهینه ای وجود دارد که باید سعی کنید تولید کنید:

- خروجی نزدیک را ترجیح دهید (+۱)، خطر صخره را ریسک کنید (-۱۰)

<sup>8</sup> living reward

- خروجی نزدیک را ترجیح دهید (+۱)، اما از صخره را اجتناب کنید (-۱۰)
- خروجی دور را ترجیح دهید (+۱۰)، خطر صخره را ریسک کنید (-۱۰)
- خروجی دور را ترجیح دهید (+۱۰)، خطر صخره را اجتناب کنید (-۱۰)
- از هر دو خروجی و صخره اجتناب کنید (بنابراین اجرای یک قسمت هرگز نباید پایان یابد)

برای بررسی پاسخ های خود، autograder را اجرا کنید:

```
python autograder.py -q q3
```

سوال های question3a تا question3c باید هر کدام یک سه تایی از (تخفیف، نويز، پاداش زندگی) را در analysis.py برگردانند.

**توجه:** می توانید سیاست های خود را در رابط کاربری گرافیکی بررسی کنید. به عنوان مثال، با استفاده از پاسخ صحیح به 3a، فلش در (0,1) باید به سمت شرق، فلش در (1,1) نیز باید به سمت شرق، و فلش در (2,1) باید به سمت شمال باشد. **توجه:** در برخی از سیستم ها ممکن است فلش ها برایتان نشان داده نشوند. در این حالت، دکمه ای را روی صفحه کلیک فشار دهید تا صفحه نمایش qValue نمایش داده شود، و به طور ذهنی خط مشی را با گرفتن  $\arg \max$ ، برای هر حالت qValue محاسبه کنید.

### نحوه ارزیابی سوال سوم:

ما بررسی خواهیم کرد که در هر مورد سیاست مورد نظر برگردانده شود.

## ۴) تکرار ارزش ناهمزمان<sup>۹</sup> (۱ امتیاز)

یک عامل تکرار مقدار در **AsynchronousValueIterationAgent** بنویسید که تا حدی برای شما در **valueIterationAgents.py** مشخص شده است. عامل تکرار ارزش شما یک برنامه ریز آفلاین است و نه یک عامل یادگیری تقویتی. **training option** مربوط به تعداد تکرارهای تکرار ارزش است که باید در مرحله برنامه ریزی اولیه اجرا شود (option -i).

<sup>۹</sup> Asynchronous Value Iteration

AsynchronousValueIterationAgent یک MDP در constructor می گیرد و تکرار مقدار چرخه ای<sup>10</sup> (شرح شده در پاراگراف بعدی) را برای تعداد مشخصی از تکرارها قبل از اتمام اجرای constructor، اجرا می کند. توجه داشته باشید که تمام این کد تکرار مقدار باید در داخل constructor (متد \_\_init\_\_) قرار گیرد.

دلیل اینکه این کلاس **AsynchronousValueIterationAgent** نامیده می شود این است که ما در هر تکرار<sup>11</sup>، فقط یک حالت را به روزرسانی میکنیم، برخلاف انجام یک به روز رسانی به صورت batch. در اینجا نحوه عملکرد تکرار ارزش چرخه ای آمده است. در اولین تکرار، فقط مقدار حالت اول را در لیست حالت ها به روز کنید. در تکرار دوم، فقط مقدار دوم را به روز کنید. این کار را ادامه دهید تا زمانی که مقدار هر حالت را یک بار به روز کنید، سپس از حالت اول برای تکرار بعدی شروع کنید. **اگر حالت انتخاب شده برای به روز رسانی ترمینال باشد، در آن تکرار هیچ اتفاقی نمی افتد.** شما می توانید این را به عنوان indexing در متغیر حالت های تعریف شده در اسکلت کد پیاده سازی کنید. به عنوان یادآوری، معادله به روزرسانی وضعیت تکرار ارزش در اینجا آمده است:

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

همانطور که در کلاس بحث شد، تکرار ارزش یک معادله fixed-point را تکرار می کند. همچنین می توان مقادیر حالت را به روش های مختلف به روزرسانی کرد، مثلاً به ترتیب تصادفی (به عنوان مثال، یک حالت را به طور تصادفی انتخاب کنید، مقدار آن را به روزرسانی کنید و تکرار کنید) یا به سبک دسته ای (مانند سوال اول). در این سوال، تکنیک دیگری را بررسی خواهیم کرد.

**AsynchronousValueIterationAgent** از **ValueIterationAgent** در سوال اول ارث بری میکند، بنابراین تنها متدی که نیاز دارید پیاده سازی کنید **runValueIteration** است. از آنجایی که constructor سوپرکلاس **runValueIteration** را فراخوانی می کند، override کردن آن برای تغییر رفتار عامل به نحو خواسته شده، کافی است. **توجه:** مطمئن شوید حالتی که یک state هیچ action ممکن در MDP ندارد را در پیاده سازی خود در نظر بگیرید. (به معنای این موضوع برای پاداش های آینده فکر کنید).

برای آزمایش پیاده سازی خود، autograder را اجرا کنید. اجرای آن باید کمتر از یک ثانیه طول بکشد. **اگر خیلی بیشتر طول بکشد، ممکن است بعداً در پروژه با مشکلاتی مواجه شوید، بنابراین همین حالا اجرای خود را کارآمدتر کنید.**

<sup>10</sup> cyclic value iteration

<sup>11</sup> iteration

```
python autograder.py -q q4
```

دستور زیر **AsynchronousValueIterationAgent** شما را لود می کند، که یک سیاست را محاسبه و ۱۰ بار اجرا می کند. دکمه ای را روی صفحه کلید فشار دهید تا مقادیر، Q-value ها و شبیه سازی را مشاهده کنید. باید متوجه شوید که مقدار حالت شروع ( $V(\text{start})$ ) که می توانید آن را از رابط کاربری گرافیکی بخوانید) و میانگین پاداش تجربی حاصل (چاپ شده پس از اتمام ۱۰ دور اجرا) کاملاً نزدیک هستند.

```
python gridworld.py -a asynchvalue -i 1000 -k 10
```

## (۵) تکرار ارزش اولویت بندی شده<sup>12</sup> (۳ امتیاز)

این الگوریتم تلاش می کند به روزرسانی های مقادیر حالت را به سمتی متمرکز کند که احتمالاً سیاست ها را تغییر دهد. مراحل الگوریتم به صورت زیر است.

- ابتدا باید برای هر حالت، همه پسین ها مشخص شود.
- یک صف خالی برای نگه داری اولویت ها تعریف کنید.
- برای هر حالت غیرپایانی  $s$  :

○ قدر مطلق تفاضل بین مقدار فعلی حالت  $s$  (که در `self.values` نگه داری می شود) و بیشترین مقدار  $Q$  ممکن از حالت  $s$  که با استفاده از اقدام های ممکن قابل تعریف است را محاسبه کنید. این مقدار را **diff** بنامید.

○ حالت  $s$  را با اولویت **-diff** به صف اولویت ها اضافه کنید. (دلیل این که از اولویت منفی استفاده می کنیم این است که صف اولویت ها به صورت min heap است و اولویت با مقدار عددی کمتر به معنی ارجحیت است و ما می خواهیم حالتی که بیشترین خطا را دارد اولویت بیشتری داشته باشد و زودتر به روزرسانی شود)

- به ازای تعداد تکرارهای مشخص شده (`self.iterations`):

---

<sup>12</sup> Prioritized Sweeping Value Iteration

- اگر صف اولویت‌ها خالی می‌باشد کار پایان یافته است.
  - در غیر این صورت حالت  $s$  را از صف اولویت‌ها بردارید.
  - در صورتی که  $s$  حالت پایانی نبود، مقدار حالت  $s$  را (**self.values**) به‌روز رسانی کنید.
  - به ازای هر پسین  $p$  از حالت  $s$  :
    - قدر مطلق تفاضل بین مقدار فعلی حالت  $p$  (که در **self.values** نگهداری می‌شود) و بیشترین مقدار  $Q$  ممکن از حالت  $p$  که با استفاده از اقدام‌های ممکن قابل تعریف است را محاسبه کنید. این مقدار را **diff** بنامید.
    - اگر  $\text{diff} < \text{theta}$  بود حالت  $p$  را با اولویت **-diff** به صف اولویت‌ها اضافه کنید. البته فقط در صورتی که حالت  $p$  با اولویت مساوی یا کمتر در صف وجود نداشته باشد.
  - توجه داشته باشید که هنگام محاسبه پسین‌های یک حالت، برای جلوگیری از به‌وجود آمدن حالت‌های تکراری، آن‌ها را در مجموعه نگهداری کنید.
- در این سوال شما باید **PrioritizedSweepingValueIterationAgent** را که تا حدی در **valueIterationAgents.py** تعریف شده است، پیاده‌سازی کنید. توجه داشته باشید که این کلاس از **AsynchronousValueIterationAgent** مشتق شده است، بنابراین تنها قسمتی که باید تغییر کند متد **runValueIteration** است.
- برای صحت‌سنجی پیاده‌سازی خود، autograder را اجرا کنید. این اجرا باید حدود 1 ثانیه طول بکشد. اگر بیشتر طول بکشد ممکن است در ادامه با مشکلاتی مواجه شوید، بنابراین در همین مرحله پیاده‌سازی خود را بهینه‌تر کنید.
- برای اینکه autograder برای این سوال کار کند، باید تکرار روی حالت‌ها را به ترتیبی که در **(self.mdp.getStates)** مشخص شده است انجام دهید.

```
python autograder.py -q q5
```

با استفاده از دستور زیر می‌توانید **PrioritizedSweepingValueIterationAgent** را در Gridworld اجرا کنید.

```
python gridworld.py -a priosweepvalue -i 1000
```

## ۶) یادگیری Q

### ۳) امتیاز

عامل تکرار کننده مقادیر، قبل از تعامل با محیط واقعی، مدل MDP خود را برای رسیدن به یک مجموعه سیاست کامل در نظر می‌گیرد و در واقع از تجربه کردن یاد نمی‌گیرد. پس از آن، هنگامی که در محیط واقعی قرار بگیرد، به سادگی از سیاست‌هایی که از قبل محاسبه شده پیروی می‌کند. این تمایز ممکن است در یک محیط شبیه سازی شده مانند Gridworld قابل ملاحظه نباشد ولی در دنیای واقعی، جایی که MDP واقعی در دسترس نیست، بسیار مهم است.

در این سوال شما باید یک عامل یادگیری Q بنویسید، که در هنگام ایجاد، تلاش زیادی برای یادگیری نمی‌کند ولی با آزمون و خطا از تعامل با محیط و از طریق متد **update** (وضعیت‌ها، اقدامات، حالت بعدی، پاداش) یاد می‌گیرد.

قسمتی از یک یادگیرنده Q در **QLearningAgent** در **qlearningAgents.py** تعریف شده است.

در این سوال باید متدهای **update** و **computeValueFromQValues** و **getQValue** و **computeActionFromQValues** را پیاده سازی کنید.

برای رفتار بهتر در متد **computeActionFromQValues** باید پیوندها را به صورت تصادفی قطع کنید. برای این کار می‌توانید از تابع **random.choice()** استفاده کنید.

اقداماتی که عامل قبلاً ندیده است، نیز یک مقدار Q دارند (مثلاً می‌توان مقدار اولیه صفر به آن‌ها اختصاص داد). در حالتی که همه اقداماتی که عامل قبلاً دیده است مقدار Q منفی داشته باشند، یک قبلاً مشاهده نشده می‌تواند بهترین اقدام باشد.

دسترسی به مقادیر Q باید فقط از طریق فراخوانی تابع **getQValue** امکان پذیر باشد در غیر این صورت در سوال دهم مشکل خواهید داشت.

با دستور زیر می‌توانید یادگیری یادگیرنده Q خود را تحت کنترل دستی مشاهده کنید که **-k** تعداد قسمت هایی که عامل برای یادگیری صرف می‌کند را مشخص می‌کند:

```
python gridworld.py -a q -k 5 -m
```

برای ارزیابی پیاده سازی خود می‌توانید autograder را اجرا کنید:

```
python autograder.py -q q6
```

## ۷) epsilon حریصانه (۲ امتیاز)

عامل یادگیری Q خود را با اضافه کردن انتخاب اقدام **epsilon-greedy** در **getAction** تکمیل کنید. این اقدام به این معناست که عامل در کسری از زمان اقدامات تصادفی انتخاب می‌کند و در غیر این صورت از بهترین مقادیر Q فعلی خود پیروی می‌کند. انتخاب یک اقدام تصادفی ممکن است منجر به انتخاب بهترین اقدام شود یعنی می‌توانید هر اقدام تصادفی مجازی را انتخاب کنید.

پس از پیاده سازی متد **getAction** با استفاده از دستور زیر رفتار عامل را در **gridworld** مشاهده کنید (از  $\epsilon = 0.3$  استفاده کنید)

```
python gridworld.py -a q -k 100 -e 0.3
```

برای ارزیابی پیاده سازی خود **autograder** را اجرا کنید:

```
python autograder.py -q q7
```

با پایان این قسمت باید بتوانید یک ربات **crawler** یادگیری Q را بدون کد اضافی اجرا کنید. دستور زیر با استفاده از یادگیرنده Q طراحی شده شما، ربات خزنده را از کلاس **Fraخوانی** می‌کند.

```
python crawler.py
```

## ۸) بررسی دوباره عبور از پل (۱ امتیاز)

ابتدا، یک **Q-learner** کاملاً تصادفی را با ضریب یادگیری پیش‌فرض بر روی **BridgeGrid** بدون نویز، با 50 اپیزود آموزش دهید و بررسی کنید که آیا سیاست بهینه در این حالت یافت می‌شود یا خیر.

```
python gridworld.py -a q -k 50 -n 0 -g BridgeGrid -e 1
```

حال، همین کار را با اپسیلون 0 دوباره تکرار کنید. آیا مقدار اپسیلون و ضریب یادگیری‌ای وجود دارد که با استفاده از آن ها، سیاست بهینه با احتمال خیلی بالا (بیشتر 99 درصد) بعد از 50 بار تکرار یاد گرفته شود؟

تابع `question8` در `analysis.py` یا یک تاپل دوتایی `(epsilon, learning rate)` برمیگرداند و یا در صورتی که جوابی پیدا نکند، رشته `'NOT POSSIBLE'` برگردانده میشود. اپسیلون با `-e` و ضریب یادگیری، با `-1` کنترل میشود.

**توجه:** پاسخ شما نباید به مکانیزم تعیین کننده‌ای که برای انتخاب اکشن‌ها استفاده شد، وابسته باشد؛ یعنی پاسخ شما باید حتی در حالتی که مثلاً ما جهان Bridge Grid را 90 درجه هم چرخانده‌ایم صحیح باشد. برای ارزیابی پاسخ خود، با استفاده از دستور زیر autograder را اجرا کنید:

```
python autograder.py -q q8
```

## ۹) پک من و Q-Learning

### (۱ امتیاز)

حال زمانی بازی پک‌من است! پک من بازی‌ها را در دو فاز انجام می‌دهد. در فاز اول، آموزش<sup>13</sup>، پک‌من در مورد امتیاز موقعیت‌ها و اکشن‌ها، آموزش میبیند. از آن جایی که حتی برای `grid`های کوچک هم زمان زیادی طول می‌کشد که `Q-value`های دقیق یاد گرفته شوند، بازی‌های آموزش پک‌من به صورت پیش‌فرض در `quiet mode` اجرا می‌شوند، بدون هیچ‌گونه نمایش گرافیکی یا کنسول. زمانی که آموزش پک‌من به انجام رسید، وارد فاز آزمون می‌شود. در زمان آزمون، `self.epsilon` و `self.alpha` در پک‌من به مقدار 0.0، به جهت توقف `Q-learning` و غیرفعال کردن جستجو، ست خواهند شد. این امر به پک‌من امکان بهره‌بردن از سیاست‌هایی که یاد گرفته را می‌دهد. بازی‌های آزمون به صورت پیش‌فرض، به صورت GUI نمایش داده میشوند. بدون هیچ‌گونه تغییری در کد، شما باید بتوانید `Q-Learning` را برای پک‌من، برای `grid`های بسیار کوچک اجرا کنید. در زیر دستور لازم برای این امر را مشاهده می‌فرمایید:

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

توجه داشته باشید که `PacmanQAgent` از قبل برای شما تعریف شده است. (`QLearningAgent` که از پیش آن را نوشته‌اید). تنها تفاوت این دو در این است که `PacmanQAgent` پارامترهای یادگیری پیش‌فرض را دارد که برای مسالهی پک‌من، مفیدتر است (`epsilon=0.05, alpha=0.2, gamma=0.8`).

---

<sup>13</sup> Training



شما نمره کامل این بخش را در صورتی که دستور بالا بدون خطا اجرا شود و عامل شما حداقل در ۸۰ درصد موارد، برنده شود. autograder بازی‌های آزمون را ۱۰۰ بار بعد از اجرای ۲۰۰۰ بار بازی‌های آموزش، اجرا خواهد کرد.

**راهنمایی:** اگر `QLearningAgent` شما برای `gridworld.py` و `crawler.py` کار می‌کند اما به نظر نمی‌رسد که سیاست خوبی را برای پک‌من در `smallGrid` یاد گرفته است، احتمالاً به این دلیل است که توابع `getAction` و یا `computeActionFromQValues` شما در بعضی از موارد، اکشن‌های دیده نشده (unseen) را به صورت مناسبی در نظر نگرفته‌اند. در واقع، از آن جایی که اکشن‌های دیده نشده بر اساس تعریف، مقدار Q-value برابر با 0 دارند؛ اگر تمام اکشن‌های دیده شده تا به اینجای کار، مقدار Q-value منفی داشته باشند، یک اکشن دیده نشده می‌تواند بهینه باشد. حتماً هم حواستان به تابع `argmax` از `util.counter` باشد!

**نکته:** برای ارزیابی پاسخ خود، دستور زیر را اجرا کنید:

```
python autograder.py -q q9
```

**نکته:** اگر قصد تست کردن پارامترهای یادگیری را دارید، میتوانید از `-a` استفاده کنید. برای مثال، `-a epsilon=0.1,alpha=0.3,gamma=0.7`، این مقادیر در `self.epsilon`، `self.alpha` و `self.gamma` در داخل خود عامل قابل دسترسی خواهند بود.

**نکته:** هنگامی که ۲۰۱۰ تا بازی انجام میشود، ۲۰۰۰ تا بازی اول به دلیل گزینه 2000 -x نمایش داده نمیشوند، که درواقع نشان میدهد که ۲۰۰۰ بازی اول، برای آموزش هستند(خروجی نخواهند داشت). بنابراین، شما فقط بازی پک من را در ۱۰ بازی آخر خواهید دید. تعداد بازی های آموزشی نیز تحت گزینه `numTraining` پاس داده می‌شود.

**نکته:** اگر می‌خواهید ۱۰ بازی آموزشی را ببینید که چه اتفاقی می افتد، از دستور زیر استفاده کنید:

```
python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a
numTraining=10
```

در طول آموزش، خروجی هر ۱۰۰ بازی را با آماری در مورد نحوه عملکرد پک‌من مشاهده خواهید کرد. اپسیلون در طول آموزش مثبت است، بنابراین پک‌من حتی پس از آموختن یک سیاست خوب هم ضعیف بازی می‌کند: این به این دلیل است که او گهگاه یک حرکت اکتشافی تصادفی به سمت یک روح انجام می‌دهد. به عنوان یک معیار، بین ۱۰۰۰ تا ۱۴۰۰ بازی طول می‌کشد تا پاداش پک‌من برای یک دسته ۱۰۰ تایی بازی مثبت شود، که نشان‌دهنده این است که عامل شروع

به دریافت تعداد برد های بیشتری از باخت کرده است. در پایان آموزش نیز پاداش باید مثبت بماند و نسبتاً بالا باشد (بین ۱۰۰ تا ۳۵۰).

همچنین به این که در اینجا دقیقاً چه اتفاقی در حال رخ دادن است هم توجه داشته باشید: حالت MDP پیکربندی دقیق بردی است که پکمن با آن روبرو است، با انتقالهای پیچیده که اکنون کل تغییرات را در آن حالت توصیف می کند. پیکربندی های میانی بازی که پکمن در آنها حرکت کرده است، اما ارواح پاسخی نداده اند، حالت های MDP نیستند، بلکه در انتقالها قرار گرفته اند.

زمانی که آموزش پکمن به پایان برسد، باید حداقل در ۹۰ درصد مواقع، بازی های تست را ببرد، چرا که در این مرحله، عامل از سیاست آموخته شده استفاده می کند. با این حال، متوجه خواهید شد که آموزش همان عامل بر روی `mediumGrid` به ظاهر ساده، به خوبی کار نمی کند. در پیاده سازی ما، میانگین پاداش ها در زمان آموزش پکمن منفی باقی می ماند. در زمان آزمون، پکمن بد بازی کرده و احتمالاً تمام بازی های آزمون را خواهد باخت. آموزش نیز با وجود بی اثر بودن، زمان زیادی را می طلبد. پکمن در طرح بندی های بزرگتر نیز برنده نمی شود، زیرا هر پیکربندی برد یک حالت جداگانه با مقادیر Q-value جداگانه است. او هیچ راهی برای تعمیم این مساله ندارد که برخورد با یک روح برای همه موقعیت ها بد است. بدیهی است که این رویکرد مقیاس پذیر نخواهد بود.

## قسمت اختیاری و امتیازی: ۱۰ Q-learning تقریبی (۳ امتیاز)<sup>14</sup>

یک عامل Q-learning تقریبی را پیاده سازی کنید که وزن ویژگی های حالت ها را یاد می گیرد (بسیاری از حالت ها ممکن است ویژگی های مشترکی داشته باشند). پیاده سازی خود را در کلاس `ApproximateQAgent` واقع در `qlearningAgents.py` انجام دهید. این کلاس در واقع زیرکلاسی از `PacmanQAgent` می باشد. نکته: Q-learning تقریبی وجود یک تابع ویژگی را فرض می کند  $f(s, a)$ ، که این ورودی این تابع، حالت و اکشن است. این توابع ویژگی، یک بردار از مقادیر ویژگیها به صورت  $[f_1(s, a), ..., f_i(s, a), ..., f_n(s, a)]$  تشکیل میدهند. این توابع ویژگیها، در فایل `featureExtractors.py` قرار گرفته اند. بردارهای ویژگیها، از جنس آبجکت های `util.Counter` هستند (مانند دیکشنری) که دوتایی های متشکل از ویژگی ها و مقادیر آنها را دارا هستند؛ توجه داشته باشید که تمام ویژگی های حذف شده، مقدار 0 را دارا هستند.

<sup>14</sup> Approximate Q-learning

<sup>15</sup> Features

Q-function تقریبی، فرم زیر را دارا است:

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

در این رابطه، هر یک از وزنهای  $w_i$ ، با یکی از وزنهای  $f_i(s, a)$  مرتبط است. در کد، شما باید بردار وزن را به عنوان یک دیکشنری پیاده‌سازی کنید که ویژگی‌ها را (که feature extractor ها برمی‌گردانند) به مقدار وزنها، نگاشت کند. همچنین شما باید بردارهای وزن خود را به همان صورتی که Q-value ها را آپدیت کردید، آپدیت نمایید:

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

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

توجه داشته باشید که *difference* در این جا، در واقع مشابه در Q-learning معمول است و r هم پاداش آزمایش شده می‌باشد.<sup>16</sup>

به صورت پیش‌فرض، **ApproximateQAgent** از **IdentityExtractor** استفاده می‌کند که هر ویژگی را به هر دوتایی **(state, action)** اختصاص می‌دهد. عامل Q-learning تقریبی شما باید مشابه **PacmanQAgent** کار کند. می‌توانید این موضوع را با استفاده از دستور زیر، تست کنید:

```
python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid
```

**نکته مهم:** **ApproximateQAgent** زیرکلاسی از **QLearningAgent** است؛ بنابراین بسیاری از توابع آن اعم از **getAction** مشترک است. توجه داشته باشید که توابع شما در **QLearningAgent** به جای دسترسی مستقیم به Q-value ها، تابع **getQValue** را صدا کنند تا زمانی که شما در کد approximate agent خود **getQValue** را override میکنید، مقادیر جدید approximate Q-value ها برای محاسبه اکشن‌ها استفاده شوند.

زمانی که مطمئن شدید که approximate learner شما به درستی با identity feature کار میکند، عامل Q-learning تقریبی خود را با feature extractor اختصاصی ما اجرا کنید، که می‌تواند برنده شدن به راحتی را یاد بگیرد:

<sup>16</sup> Note that the difference term is the same as in normal Q-learning, and r is the experienced reward.

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor  
-x 50 -n 60 -l mediumGrid
```

حتی طرح بندی‌های بزرگتر نیز برای **ApproximateQAgent** شما نیز نباید مشکلی ایجاد کند. (البته دقت داشته باشید که ممکن است که آموزش مدل چند دقیقه‌ای طول بکشد.)

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor  
-x 50 -n 60 -l mediumClassic
```

اگر به اروری در حین اجرا برخورد نکنید، عامل approximate Q-learning شما باید تقریباً در تمامی بازی‌ها با استفاده از همین ویژگی‌های ساده، برنده شود، حتی با تنها 50 بازی آموزشی.

ارزیابی) ما عامل approximate Q-learning شما را اجرا و بررسی خواهیم کرد که وقتی هر کدام با مجموعه‌ای از مثال‌ها ارائه می‌شوند، همان Q-value و وزن ویژگی‌های ارائه شده در پیاده‌سازی مرجع ما را یاد می‌گیرد.

برای ارزیابی پیاده سازی خود، autograder را اجرا کنید:

```
python autograder.py -q q10
```

تبریک! شما یک عامل پک من یادگیرنده دارید!

## توضیحات تکمیلی

- پاسخ به تمرین ها باید به صورت فردی انجام شود. در صورت استفاده مستقیم از کدهای موجود در اینترنت و مشاهده تقلب، برای همه‌ی افراد نمره صفر لحاظ خواهد شد.
- برای این پروژه به صورت رندوم از تعدادی از دانشجویان تحویل آنلاین گرفته خواهد شد و نمره دهی مابقی دانشجویان بر اساس گزارش پروژه، نمره autograder و پیاده سازی انجام شده است، لذا **ضروری** است که همه‌ی دانشجویان گزارشی برای پروژه تهیه نموده و در آن به طور مختصر نحوه ی پیاده سازی های انجام شده را شرح دهند.
- فایل های `valueIterationAgents.py` و `qlearningAgents.py` و `analysis.py` به همراه گزارش را در قالب یک فایل فشرده با فرمت `AI_P3_9931099.zip` در سامانه کورسز آپلود کنید.
- در صورت هرگونه سوال یا ابهام از طریق ایمیل [ai.aut.fall1400@gmail.com](mailto:ai.aut.fall1400@gmail.com) با تدریسپاران در تماس باشید، همچنین خواهشمند است در متن ایمیل به شماره دانشجویی خود اشاره کنید.
- همچنین می‌توانید از طریق تلگرام نیز با آیدی‌های زیر در تماس باشید و سوالاتتان را مطرح کنید:
  - o [@MohmdRad](https://t.me/MohmdRad)
  - o [@lilhedi](https://t.me/lilhedi)
- ددلاین این پروژه **1 بهمن ۱۴۰۰ ساعت ۲۳:۵۵** است و امکان ارسال با تاخیر وجود ندارد، بنابراین بهتر است انجام پروژه را به روزهای پایانی موکول نکنید.