گزارش پروژه سوم هوش مصنوعی: محمدرضا صادقیان 9731121

بخش اول)

در این بخش باید توابع computeActionFromValues، computeQValueFromValues، runValueIteration و maxQvalue را پیاده کنیم که به طور مختصر در ادامه در مورد پیاده سازی هر کدام توضیحاتی آورده ام.

- computeActionFromValues: این تابع بر روی Action های ممکن در State فعلی پیمایش را انجام می دهند و با استفاده از تابعی که مقدارQ value را محاسبه می کند، سپس هر اکشنی که ماکس Q value را داشت برمی‌گرداند.
- computeQValueFromValues: این تابع اکشن و State را دریافت می کند و بر اساس آن مقدار Q value را بدست می آورد .در این تابع با استفاده از getTransitionStatesAndProbs که یک آرایه ای از تاپل(prime_s, transition) می باشد و prime_s استیت بعدی و transition مقدار احتمال برای ورود به استیت بعدی می باشد ، مقدار Q value را بدست می آوریم.
- runValueIteration: این تابع یک سری Iteration را انجام می دهد. در ابتدای هر تکرار یک دیکشنری خالی را می سازیم و مقادیر valueها را در تکرار آپدیت می کنیم. پس از تمام شدن هر تکرار، مقادیر value که در این تکرار بدست آوردیم را برابر با مقادیر اصلی value قرار می دهیم. در هر تکرار بر روی استیت ها پیمایش می کنیم و ماکزیمم Q value را بد ست می آوریم و مقادیر values_iterationرا آپدیت می کنیم.
- maxQvalue: یک تابع کمکی نیز تعریف کرده ام که به وسیله ی آن میتوان مقدار value هر استیت که برابر با ماکزیمم مقدار Q value ها ی آن استیت می باشد را بدست آورد.

خروجی :

```
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 20:38:48

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

بخش دوم)

من صرفا تلاش کردم با تغییر یک که نویز باشد به نتیجه برسم و اعداد زیر رسیدم:

Discount = 0.9
Noise = 0.002

مقدار نویز که کم شود عامل با قطعیت از خانه های شامل 100- فرار می کند.

خروجی:

```
Question q2
===========

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

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


Finished at 21:00:00

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

بخش سوم)

هر چه مقدار تخفیف یا Discount بیشتر باشد، میزان آینده نگری عامل ما بیشتر می شود. چرا که طبق معادله ی Q value هرچه مقدار تخفیف به عدد یک نزدیکتر شود، تاثیر Q value استیت های بعدی در مقدار Value فعلی بیشتر می شود. از طرف دیگر هرچه مقدار نویز بیشتر باشد، عامل برای اینکه ریسک کمتری کند به صخره ها نزدیک نمی شود. و برعکس و در نهایت هرچهliving reward بیشتر باشد، امکان اینکه عامل به ترمینال نرسد بیشتر است چرا که در حال گشتن در grid می باشد و پاداش نیز دریافت می کند.

3a)

Discount = 0.1
Noise = 0
LivingReward = -0.1

3b)

Discount = 0.5
Noise = 0.4
LivingReward = -0.7

3c)

Discount = 1
Noise = 0
LivingReward = -0.1

3d)

Discount = 1
Noise = 0.4
LivingReward = -0.1

3e)

Discount = 1
Noise = 0
LivingReward = 15

بخش چهارم)

کاملا مشابه بخش اول عمل می کنیم فقط نحوه انتخاب State ها متفاوت است. در این حالت برای هر بار اجرا For، State ای آپدیت می شود که شماره آن در ()Mdp.getState برابر (State(i) % #Statas) که مقدار آنها برابر می شود با ()Mdp.getReward

بخش پنجم)

کاملا مشابه دستور پروژه عمل می کنیم و از ساختمان داده های Set خود پایتون چون نیاز به مفهموم مجموعه داشتم استفاده کردم و برای Heap هم از فایل Util کمک گرفتم.

خروجی:

```
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 21:25:55

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

بخش ششم)

برای راحت سازی پیاده سازی بهتر است مقادیر Q value ها را همانند ویدیو آموزشی در ساختمان داده Counter ای که در فایل Util قرار دارد ذخیره کنیم و آن را در Constructor کلاس خود قرار می دهیم.

تابع getQValue(self, state, action) باید Q value مربوط به State با انجام اکشن را برگرداند، پس خانه مربوط به آن State و اکشن از لیست Q values را بر می گرداند. از آنجا که دیفالت مقدار صفر دارد، اگر State را مشاهده نکنیم صفر بر میگرداند و عملکردش درست است.
در تابع computeValueFromQValues(self, state) ابتدا همه اکشن های ممکن از State را می گیریم. اگر هیچ اکشنی ممکن نباشد باید صفر برگرداند. در غیر این صورت برای هر اکشن ممکن، Q value را حساب می کنیم، به این ترتیب که ابتدا با تابع self.getPolicy(state) اکشنی که انجام می دهد و بعد با تابع self.getQValue(state, policy) مقدار Q value به حاصل از آن State و اکشن را پیدا کرده و بر میگردانیم.در تابع computeActionFromQValues(self, state) می خواهیم بهترین اکشن که بیشترین مقدار را دارد پیدا کنیم. از این رو ابتدا اکشن های ممکن از State را میگیریم، اگر هیچ اکشنی ممکن نباشد، None بر می گرداند همان طور که سوال خواسته است. در غیر این صورت از بین اکشن های ممکن، Q valueهر کدام با پیدا کرده و در یک لیست می ریزیم و همچین اگر از مقدار Q اکشن های قبلی بیشتر باشد، آن را به عنوان بیشترین مقدار Q value ذخیره می کنیم. سپس از بین مقادیر، اکشن آنهایی که بیشترین مقدار را دارند در لیست policies میریزیم. در نهایت برای اینکه رفتار بهتری داشته باشد، یک اکشن رندوم از بین این اکشن ها انتخاب می کنیم و بر می گردانیم.

خروجی:

```
*** 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 21:37:57

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

بخش هفتم)

ابتدا تمامی اکشن های مجاز را استخراج میکنیم سپس با استفاده از متد:

randomAction = util.flipCoin(self.epsilon)

با توجه به اپسیلون وارد شده مقدار رندومی احتمالا تولید می شود.

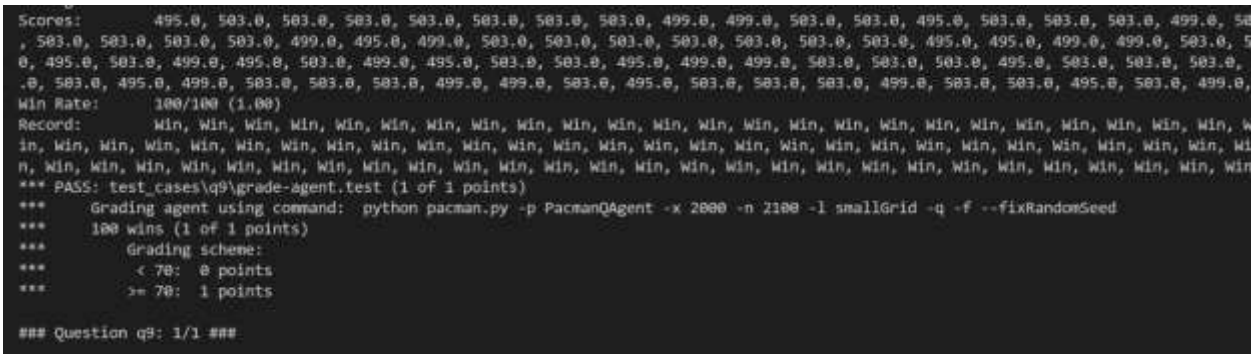
درصورتی که اکشن مجازی وجود نداشت None برگردانده می شود. اگر اکشن تصادفی تولید شده بود آن برگردانده می شود و در غیر این صورت همان سیاست بهینه محاسبه شده در قسمت قبل برگردانده می شود.

بخش هشتم)

صرفا کافی است حالتی که تابع جوابی ندارد عبارت "NOT POSSIBLE" را برگردانیم.

بخش نهم)

چون قبلا به طور کامل Q learning را پیاده سازی کردیم، عامل پکمن با استفاده از همان توابع محیط را یادمی گیرد و اکشن مناسب را انجام می دهد ولی باید تعداد دفعات زیادی آزمون و خطا کند و یاد بگیرد.



مشخص است که عامل پس از یادگیری تمام بازی ها را برده است! پس به درستی از اطلاعاتی که یادگرفته برای کشف سیاست بهینه استفاده می کند.

بخش دهم)

در تابع update، مقادیر weight ها باید آپدیت شوند. مطابق با فرمول، ابتدا لیست feature ها را با استفاده از (action, state(getFeatures.featExtractor.self برای استیت و اکشن ورودی به دست آورده و سپس مقدار diff را دقیقا عین فرمول به دست می آوریم. سپس برای هر feature ، مقدار f[weights] را به عالوهی f[features*diff*alpha] میکنیم. در تابع getQValue مقادیر Q value ها باید آپدیت شوند. لیست feature ها را به دست آوردم و برای هر feature مطابق با فرمول مقدار Q value را آپدیت کردم. در نهایت Q value که همان Q(S,A) است را برمی گرداند.

خروجی:

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


Finished at 21:52:36

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.
```