

به نام خدا



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

مبانی و کاربردهای هوش مصنوعی ترم بهار ۱۴۰۲

پروژه دوم

مهلت تحویل ۳ اردیبهشت ۱۴۰۲ ساعت ۲۳:۵۹

مقدمه

در این پروژه عاملی را برای بازی کلاسیک پکمن طراحی خواهید کرد که این بار شامل روح ها نیز می باشد. در این مسیر از جستجوی مینیماکس^۱ و مینیماکس احتمالی^۲ استفاده خواهید کرد و تابع ارزیابی^۳ طراحی می کنید.

ساختار پروژه دوم نسبت به پروژه قبل تغییر چندانی نداشته است اما توصیه می کنیم به جای تغییر در کدهای پروژه ی قبل، پروژه جدیدی را بارگیری کرده و شروع کنید. مشابه با پروژه قبل برای دیباگ و تست درستی الگوریتم های 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
```

¹ Minimax

² Expectiminimax

³ Evaluation Function

به صورت پیش فرض اجرای autograder.py با گزینه ی t- شامل نمایش گرافیکی خواهد بود اما گزینه q- بدون نمایش گرافیکی است. برای اجبار به اجرای گرافیکی می توانید از فلگ --graphics- و برای اجبار به عدم نمایش گرافیکی از --no-graphics استفاده کنید.

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

فایل هایی که باید ویرایش کنید:	
multiAgents.py	شامل تمامی عامل های جستجوی چند عاملی می باشد.
فایل هایی که شاید بخواهید آن ها را ببینید:	
pacman.py	فایل اصلی که بازی های پکمن را اجرا می کند. این فایل کلاس GameState را برای بازی پکمن توصیف می کند که در این پروژه از آن استفاده می کنید.
game.py	منطق پیاده شده برای دنیای پکمن در این فایل قرار دارد. این فایل شامل چندین کلاس مانند AgentState (وضعیت عامل)، Agent (عامل)، Grid (نقشه بازی) و Direction (جهت) می شود.
util.py	ساختمان داده های مفید برای پیاده سازی الگوریتم های جستجو در این فایل قرار دارند. در این پروژه از این ساختمان های داده استفاده نخواهید کرد اما ممکن است توابع تعریف شده در آن به کارتان آیند.
فایل هایی که می توانید آن ها را رد کنید:	
graphicsDisplay.py	گرافیک های پیاده سازی شده برای بازی پکمن
graphicsUtils.py	پشتیبانی برای گرافیک بازی
textDisplay.py	گرافیک ASCII برای پکمن
ghostAgents.py	عامل های کنترل کننده ارواح
keyboardAgents.py	رابط صفحه کلید برای کنترل پکمن

برنامه برای خواندن فایل‌های نقشه و ذخیره اطلاعات آن‌ها	layout.py
تصحیح‌کننده خودکار پروژه	autograder.py
Parse کردن تست‌های مصحح خودکار و فایل‌های راه‌حل	testParser.py
کلاس‌های کلی تست خودکار	testClasses.py
پوشه دربردارنده تست‌های مختلف برای هر سوال	test_cases/
کلاس‌های تست خودکار پروژه دوم	searchTestClasses.py

آنچه باید انجام دهید:
 شما باید بخش‌هایی از فایل multiAgents.py را تغییر دهید.
 لطفاً سایر بخش‌های پروژه را به هیچ عنوان تغییر ندهید.

به پکمن چند عاملی خوش آمدید!

پس از بارگیری کد پروژه از سامانه کورسز و خارج کردن آن از حالت فشرده مشابه با پروژه قبل می توانید با تایپ کردن فرمان های زیر بازی را اجرا کنید:

```
python pacman.py
```

حال ReflexAgent از فایل multiAgents.py را به عنوان عامل بازی انتخاب کنید:

```
python pacman.py -p ReflexAgent
```

مشاهده خواهید کرد که عامل به خوبی بازی نمی کند. حتی در زمین های ساده:

```
python pacman.py -p ReflexAgent -l testClassic
```

کد ReflexAgent را در فایل multiAgents.py بررسی کنید تا متوجه شوید چگونه کار میکند.

(۱) عامل عکس العمل (۴ امتیاز)

در این سوال هدف بهبود عملکرد ReflexAgent در multiAgents.py است. برای این کار می‌بایست تابع `evaluationFunction` را به گونه ای تغییر دهید که بر اساس نتایج عمل (action) انجام شده و حالت (state) ثانویه ارزیابی انجام شود و نه حالت حاصل شده به تنهایی. در این راستا متد هایی در ابتدای `evaluationFunction` آورده شده است که اطلاعات مهم مانند موقعیت جدید (`newPos`) پکمن و یا وضعیت غذاها پس از انجام عمل (`newFood`) را از `GameState` استخراج می‌کند. برای بررسی محتوای اطلاعات فراهم شده می‌توانید آنها را چاپ کنید. در نهایت ReflexAgent باید با در نظر گرفتن موقعیت غذاها و روح‌ها، بهترین انتخاب را انجام دهد و در نقشه `testClassic` همواره برنده شود. برای بررسی این موضوع می‌توانید از دستور زیر استفاده کنید:

```
python pacman.py -p ReflexAgent -l testClassic
```

برای امتحان کردن عامل خود در نقشه `mediumClassic` با یک یا دو روح و اجرای بازی با سرعت بالا از دستورات زیر استفاده کنید:

```
python pacman.py --frameTime 0 -p ReflexAgent -k 1
```

```
python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

نکات و راهنمایی‌ها:

- عامل شما به احتمال زیاد در بازی با دو روح شکست می‌خورد مگر آنکه تابع ارزیابی بسیار قوی طراحی کرده باشید.
- توجه کنید که برای نوشتن یک تابع ارزیابی مناسب به جای استفاده از مقادیر اطلاعات فراهم شده باید از رابطه‌ی بین آن‌ها مانند فاصله تا غذا، استفاده کنید.
- **مهم:** برای کسب نمره کامل از این سوال باید از هر چهار پارامتر `newGhostStates`، `newFood`، `newPos` و `newScaredTimes` استفاده کنید. در صورت دریافت نمره کامل از `autograder` و عدم استفاده از هر چهار پارامتر مذکور همچنان نمره‌ای به شما تعلق نخواهد گرفت.

سوال: توضیح دهید که از هر کدام از پارامترهای دخیل در تابع ارزیابی چگونه استفاده کرده‌اید و هر کدام چگونه بر روی خروجی تاثیر می‌گذراند (تاثیر کدام یک از پارامترها بیشتر است؟).

سوال: چگونه می‌توان پارامترهایی که مقادیرشان در یک راستا نمی‌باشند را با یکدیگر برای تابع ارزیابی ترکیب کرد؟ (مانند فاصله تا غذا و روح که ارزش آن‌ها بر خلاف یکدیگر می‌باشد)

سوال: راجع به نحوه پیاده‌سازی تابع evaluationfunction توضیح دهید همچنین توضیح دهید چرا امتیازی که برمیگردانید مناسب است و علت انتخاب نحوه محاسبه آن را بیان کنید.

نحوه ارزیابی سوال اول:

عامل شما در نقشه ی openClassic ده بار اجرا خواهد شد. اگر عامل هیچ گاه برنده نشود و یا قبل از برنده شدن زمانش تمام شود، صفر امتیاز می‌گیرید. اگر عامل حداقل ۵ بار برنده شود یک امتیاز و اگر هر ۱۰ بار برنده شود دو امتیاز دریافت می‌کنید. اگر میانگین امتیاز عامل بیش از ۵۰۰ باشد یک امتیاز بیشتر و اگر بالا ۱۰۰۰ باشد دو امتیاز بیشتر دریافت خواهید کرد. در نتیجه برای دریافت نمره کامل می‌بایست هر ده بار برنده شده و میانگین امتیازات بالای ۱۰۰۰ باشد. برای ارزیابی می‌توانید از دستور زیر استفاده کنید:

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

```
python autograder.py -q q1 --no-graphics
```

(۲) مینیماکس (۵ امتیاز)

در این سوال باید کل اس MinimaxAgent را کامل کنید. عامل شما باید به ازای هر تعداد روح درست عمل کند که برای این کار باید به ازای هر روح یک لایه min و به ازای پکمن تنها یک لایه max در درخت مینیماکس خود داشته باشید.

همچنین درخت مینیماکس شما باید تا عمق دلخواه گسترش یابد و برگ های آن با تابع مناسب ارزیابی شوند. به این منظور کلاس MinimaxAgent طوری در نظر گرفته شده است که از MultiAgentSearchAgent ارث می برد و به این واسطه دو ویژگی self.depth و self.evaluationFunction را در خود دارد که باید حتماً از آنها برای بررسی رسیدن به عمق دلخواه و ارزیابی برگ ها در MinimaxAgent استفاده کنید. این تابع ارزیابی به طور پیش فرض scoreEvaluationFunction است که می توانید در همان فایل multiAgents.py آن را مشاهده کنید. توجه کنید که نیازی به اعمال تغییر در تابع ارزیابی نیست.

نکات و راهنمایی ها:

- با پیاده سازی صحیح مینیماکس همچنان عامل بعضی بازی ها را می بازد که این امر طبیعی است.
- توجه داشته باشید که تابع scoreEvaluationFunction بر خلاف تابع ارزیابی که در سوال اول پیاده سازی کردید بر اساس حالت (state) ارزیابی را انجام می دهد و نه عمل (action).
- مقدار مینیماکس برای حالت اولیه در نقشه ی minimaxClassic به ازای عمق های ۱، ۲، ۳ و ۴ به ترتیب برابر با ۹، ۸، ۷ و ۴۹۲- می باشد. در این نقشه با فرض عمق برابر با ۴، عامل از هر ۱۰۰۰ بازی حدوداً ۶۶۵ بازی را خواهد برد. برای بررسی عملکرد مینیماکس در این نقشه می توانید از دستور زیر استفاده کنید:

```
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
```

- پکمن همواره عامل با اندیس صفر می باشد و عامل ها بر اساس افزایش اندیس باید حرکت کنند، یعنی ابتدا پکمن و سپس روح ها.
- این موضوع را در طراحی مینیماکس در نظر داشته باشید که تمام وضعیت های بازی باید از جنس GameStates باشند چه به صورت ورودی به تابع getAction و یا خروجی از GameState.generateSuccessor.

- در نقشه‌های بزرگ مانند openClassic و mediumClassic عامل پکمن مبتنی بر مینیماکس در فرار از مردن خوب عمل می‌کند اما در بردن ضعف دارد. عموماً عامل نمی‌تواند افقی را دورتر از خوردن غذاها ببیند به همین خاطر خوردن یا نخوردن آنها اهمیت چندانی ندارد و بارها مشاهده می‌شود که در کنار یک dot می‌چرخد و در مسیر خلاف جهت حرکت می‌کند. این موضوع در سوال پنجم اصلاح خواهد شد.

نحوه ارزیابی سوال دوم:

اجرای صحیح کد شما بستگی به این دارد که آیا تعداد درستی از حالت‌های بازی در درخت پیمایش می‌شود یا خیر. این موضوع وابسته به دفعات فراخوانی GameState.generateSuccessor در کد MinimaxAgent شما است. فراخوانی بیشتر یا کمتر از حد انتظار منجر به کسر نمره خواهد شد. برای ارزیابی کد خود می‌توانید از دستور زیر استفاده کنید:

```
python autograder.py -q q2
python autograder.py -q q2 --no-graphics
```

سوال: وقتی پکمن به این نتیجه برسد که مردن آن اجتناب ناپذیر است، تلاش می‌کند تا به منظور جلوگیری از کم شدن امتیاز، زودتر ببازد. این موضوع را می‌توانید با اجرای دستور زیر مشاهده کنید:

```
python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
```

بررسی کنید چرا پکمن در این حالت به دنبال باخت سریع‌تر است؟

سوال: راجع به نحوه پیاده‌سازی min-max توضیح دهید.

۳) هرس آلفا-بتا (۵ امتیاز)

در این سوال باید با اضافه کردن هرس آلفا-بتا، پیمایش درخت مینیماکس را ارتقا دهید. برای این کار کلاس AlphaBetaAgent را تکمیل کنید. توجه داشته باشید که همچنان چندین لایه min (به ازای هر روح) و یک لایه max (به ازای پکمن) خواهیم داشت.

در نتیجه‌ی این ارتقا باید شاهد افزایش سرعت الگوریتم باشید. این موضوع را میتوانید با اجرای دستور زیر در عمق ۳ و مقایسه آن با عملکرد عامل MinimaxAgent در عمق ۲ مشاهده کنید. انتظار می‌رود هر دو عامل تقریباً در یک زمان به نتیجه برسند.

```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

نکات و راهنمایی‌ها:

- مقادیر گره‌های درخت مینیماکس در AlphaBetaAgent عیناً برابر با مقادیر MinimaxAgent می‌باشد چرا که هر دو از یک تابع ارزیابی استفاده خواهند کرد. اما مقادیر انتخاب شده، به دلیل شرایط مرزی، ممکن است در هرس آلفا-بتا متفاوت باشد.
- **مهم:** در شرایط برابری هرس انجام ندهید تا تعداد حالت‌های کاوش شده با انتظار autograder همخوانی داشته باشد. پیاده‌سازی شما باید برگرفته از سودو-کد زیر باشد که در حذف حالت برابر، با اسلاید صفحه ۳۱ تدریس شده در کلاس متفاوت است.

Alpha-Beta Implementation

α : MAX's best option on path to root
 β : MIN's best option on path to root

```
def max-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize v =  $-\infty$   
    for each successor of state:  
        v = max(v, value(successor,  $\alpha$ ,  $\beta$ ))  
        if v >  $\beta$  return v  
         $\alpha$  = max( $\alpha$ , v)  
    return v
```

```
def min-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize v =  $+\infty$   
    for each successor of state:  
        v = min(v, value(successor,  $\alpha$ ,  $\beta$ ))  
        if v <  $\alpha$  return v  
         $\beta$  = min( $\beta$ , v)  
    return v
```

نحوه ارزیابی سوال سوم:

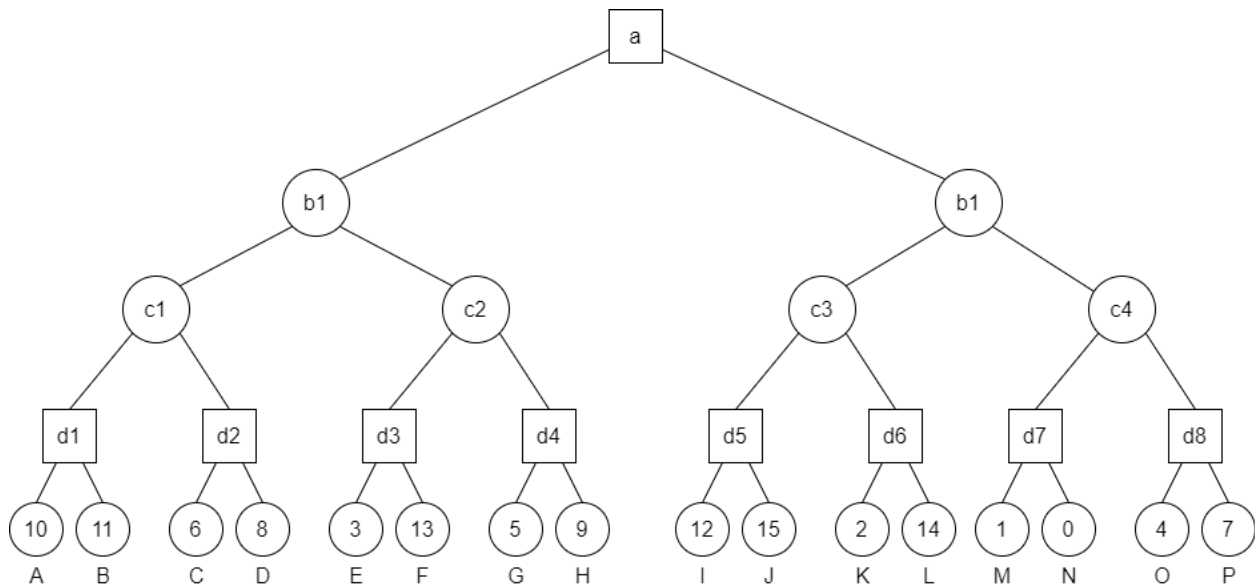
مشابه با سوال دوم، برد و باخت بازی ملاک نمره‌دهی نیست بلکه ارزیابی بر اساس تعداد حالت‌های پیمایش شده در درخت مینیماکس انجام می‌گیرد. مهم است تا هرس آلفا-بتا را بدون تغییر در ترتیب فرزندان هر گره اعمال کنید. به عبارت دیگر باید با همان ترتیبی که action های مجاز را از

GameState.getLegalActions بدست می‌آورید GameState.generateSuccessor را فراخوانی کنید و تعداد این فراخوانی‌ها نباید بیشتر یا کمتر از آنچه از هرس آلفا-بتا انتظار می‌رود باشد. برای ارزیابی کد خود می‌توانید از دستور زیر استفاده کنید:

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

```
python autograder.py -q q3 --no-graphics
```

سوال: فرض کنید درخت زیر یکی از تست‌های داده شده به الگوریتم آلفا-بتا شما است. گره‌های مربوط به پکمن با مربع و گره‌های هر روح با دایره نمایش داده شده است. در وضعیت فعلی پکمن دو حرکت مجاز دارد، یا می‌تواند به سمت راست حرکت کرده و وارد زیر درخت b شود و یا به سمت چپ حرکت کرده و وارد زیر درخت b شود. الگوریتم آلفا-بتا را تا عمق ۴ روی درخت زیر اجرا کرده و مشخص کنید کدام گره‌ها و به چه دلیل هرس می‌شوند. همچنین مشخص کنید در وضعیت فعلی، حرکت بعدی پکمن باید به سمت راست باشد یا چپ؟



سوال: آیا در حالت کلی هرس آلفا-بتا قادر است که مقداری متفاوت با مقدار به دست آمده بدون هرس را در ریشه درخت تولید کند؟ در گره‌های میانی چگونه؟ به طور خلاصه دلیل خودتان را توضیح دهید.

سوال: نحوه پیاده سازی کدهای الگوریتم هرس آلفا-بتا را توضیح دهید. در چه زمانی از این الگوریتم نمیتوانیم استفاده کنیم؟

۴) مینیماکس احتمالی (۵ امتیاز)

در مینیماکس و هرس آلفا-بتا فرض می‌شود حریف بهینه‌ترین انتخاب‌ها را انجام می‌دهد در حالیکه در واقعیت این گونه نیست و مدل‌سازی احتمالی عاملی که انتخاب‌های غیربهینه دارد، ممکن است با نتیجه‌ی بهتری برای ما همراه باشد. ارواح تصادفی نیز انتخاب‌های بهینه ندارند و بدین ترتیب مدل‌سازی آن‌ها با جستجوی مینیماکس، ممکن است نتیجه‌ی بهینه‌ای نداشته باشد. روش مینیماکس احتمالی به جای در نظر گرفتن کوچک‌ترین حرکات حریف، مدلی از احتمال حرکات را در نظر می‌گیرد پس برای ساده‌سازی مدل احتمالی، فرض کنید ارواح حرکات خود را از بین ۴ حرکت مجازشان به صورت یکنواخت و تصادفی انتخاب می‌کنند.

در این سوال باید تغییرات لازم را در کلاس ExpectimaxAgent اعمال کنید. همچنین برای مشاهده عملکرد عامل خود می‌توانید از دستور زیر استفاده کنید:

```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

نحوه ارزیابی سوال چهارم:

برای امتحان کردن عامل خود از دستورات زیر استفاده کنید:

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

```
python autograder.py -q q4 --no-graphics
```

سوال: همانطور که در سوال دوم اشاره شد روش مینیماکس در موقعیتی که در دام قرار گرفته باشد خودش اقدام به باختن و پایان سریع‌تر بازی می‌کند ولی در صورت استفاده از مینیماکس احتمالی در ۵۰ درصد از موارد برنده می‌شود. این سناریو را با هر دو دستور زیر امتحان کنید و درستی این گزاره را نشان دهید. همچنین دلیل این تفاوت در عملکرد مینیماکس و مینیماکس احتمالی را توضیح دهید.

```
python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
```

```
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
```

سوال: الگوریتم [رولت ویل](#) را بررسی کنید و بیان کنید که انتخاب هر کروموزوم در این الگوریتم بر چه اساسی است؟ اگر در بازی پکمن خودمان از آن استفاده کنیم، چه معیاری برای انتخاب هر **action** مناسب است؟ بر فرض اگر نیاز بود تا با کمک الگوریتم رولت ویل بیش‌تر از یک حالت انتخاب شود (با کمک مقدار تابع ارزیابی برای هر حالت) و درخت با توجه به این دو حالت گسترش پیدا کند و حالت‌های بعدی آن‌ها هم بررسی شوند (تا بتوانیم برای حالت بعدی انتخاب بهتری داشته باشیم)، چه راهی به نظر شما منطقی می‌باشد؟

سوال: در سناریوهای احتمالاتی که حرکات ارواح کاملاً قطعی نیستند، استراتژی بهینه برای یک عامل Pac-Man با استفاده از الگوریتم کمینه احتمالی یا Expectimax چیست؟ این استراتژی چه تفاوتی با استراتژی در سناریوهای قطعی دارد؟ تحلیل پیچیدگی استراتژی بهینه و مقایسه آن با استراتژی در سناریوهای قطعی. مدل‌های احتمالی مختلف و تأثیر آن‌ها بر استراتژی بهینه را در نظر بگیرید.

سوال: در سناریوهای احتمالاتی که عامل Pac-Man فقط مشاهدات جزئی از وضعیت بازی دارد، چگونه عامل می‌تواند از probabilistic minimax یا Expectimax احتمالی برای تصمیم‌گیری بهینه استفاده کند؟ چگونه عامل می‌تواند باور خود را در مورد وضعیت بازی بر اساس مشاهدات جدید به روز کند؟ مبادلات بین اکتشاف و بهره‌برداری را در سناریوهای مشاهده جزئی⁴ تجزیه و تحلیل کنید و آن‌ها را با مبادلات در سناریوهای مشاهده کامل مقایسه کنید. مدل‌های مختلف مشاهده جزئی و تأثیر آن‌ها بر فرآیند تصمیم‌گیری را در نظر بگیرید.

⁴ Partial Observation

(۵) تابع ارزیابی (۶ امتیاز)

یک تابع ارزیابی بهتر برای پکمن در `betterEvaluationFunction` بنویسید. این تابع ارزیابی باید به جای ارزیابی عمل (`action`)، حالت‌ها (`states`) را ارزیابی کند، بر خلاف آنچه در سوال اول انجام دادید. تابع ارزیابی شما باید در جستجوی عمق ۲، سناریوی `smallClassic` را با یک روح تصادفی و در نیمی از زمان به اتمام برساند و برنده شود. برای دریافت نمره کامل، پکمن باید در لحظه‌ای که برنده می‌شود به طور میانگین حدود ۱۱۰۰ امتیاز گرفته باشد.

مهم: برای کسب نمره کامل از این سوال باید از هر چهار پارامتر `pacmanPosition`، `food`، `scares` و `ghostPositions` استفاده کنید. در صورت دریافت نمره کامل از `autograder` و عدم استفاده از هر چهار پارامتر مذکور همچنان نمره‌ای به شما تعلق نخواهد گرفت.

سوال: تفاوت‌های تابع ارزیابی پیاده شده در این بخش را با تابع ارزیابی بخش اول بیان کنید و دلیل عملکرد بهتر این تابع ارزیابی را بررسی کنید.

سوال: طراحی بهینه یک تابع ارزیابی برای یک عامل Pac-Man چیست؟ آیا می‌توان یک تابع ارزیابی طراحی کرد که بازی بهینه را در همه حالت‌های بازی تضمین کند؟ اگر نه، محدودیت‌های رویکرد عملکرد ارزیابی چیست؟ معاوضه بین دقت و کارایی محاسباتی^۵ در طراحی تابع ارزیابی را تجزیه و تحلیل کنید. طراحی‌های مختلف عملکرد ارزیابی و تأثیر آنها بر رفتار عامل را در نظر بگیرید.

سوال: سه تابع ارزیابی مختلف را آزمایش کرده و نتیجه‌های آنان را گزارش کنید. همچنین نقاط قوت و ضعف هر کدام و علت کارایی یا ناکارآمدی هر کدام را توضیح دهید دقت کنید که تمام توابع شما باید با امتیاز بالا (امتیازی که خودتان از این بخش گرفتید) این بخش را تمام کنند. در مقایسه‌تان از دقت، کارایی محاسباتی و دیگر فاکتورهای لازم استفاده کنید.

نحوه ارزیابی سوال پنجم:

`autograder` عامل شما را ۱۰ بار روی سناریوی `smallClassic` اجرا می‌کند و به صورت زیر نمره‌دهی می‌شود:

اگر حداقل یک بار بدون زمان بندی خودکار برنده شوید، ۱ امتیاز دریافت می‌کنید. هر نماینده‌ای که این معیارها را برآورده نکند صفر امتیاز دریافت می‌کند.

⁵ Computational Complexity

- ۱ امتیاز در صورت برنده شدن حداقل ۱ بار و ۲ امتیاز در صورت برنده شدن در تمام ۱۰ بار
 - ۱ امتیاز در صورت کسب میانگین امتیاز حداقل ۵۰۰ و ۲ امتیاز در صورت کسب میانگین امتیاز حداقل ۱۰۰۰
 - ۱ امتیاز در صورتی که بازی‌های شما به‌طور میانگین در کمتر از ۳۰ ثانیه در autograder در حالت no-graphics اجرا شود.
- برای امتحان کردن عامل خود با تابع ارزیابی تعریف شده از دستورات زیر استفاده کنید:

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

```
python autograder.py -q q5 --no-graphics
```

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 همه ی افراد نمره صفر لحاظ خواهد شد.
- فایل **multiAgents.py** را به همراه پاسخ خود به سوالات که در فایل به شکل **سوال** مشخص شده اند و توضیحاتتان برای پیاده سازی های انجام شده به همراه اسکرینشات را در قالب یک فایل فشرده با فرمت **AI_P2_[Student_Number].zip** در سامانه کورسز آپلود کنید.
- لازم به ذکر است که پاسخ سوالات باید کامل و دقیق باشد ولی توضیحات اضافه تر برای پیاده سازی اجباری نبوده اما برای آنکه تا روز ارائه توضیحات را فراموش نکنید، این کار برای خودتان مفید خواهد بود.
- در صورت هرگونه سوال یا ابهام از طریق ایمیل aispring1401@gmail.com با تدریسپاران در تماس باشید، همچنین خواهشمند است در متن ایمیل به شماره دانشجویی خود اشاره کنید.
- همچنین می توانید از طریق تلگرام نیز با آیدی های زیر در تماس باشید و سوالاتتان را مطرح کنید:
 - @Hasti_Jalali
 - @Farshid_Nooshi
- این پروژه تحویل آنلاین از همه دانشجویان خواهد داشت تسلط کافی به سورس کد برنامه ضروری است. بخشی از نمره به صورت ضریب به تسلط شما وایسته است.
- ددلاین این پروژه **۳ اردیبهشت ۱۴۰۲ ساعت ۲۳:۵۹** است، بنابراین بهتر است انجام پروژه را به روز های پایانی موکول نکنید.