

به نام خدا



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

مبانی و کاربردهای هوش مصنوعی ترم پاییز ۱۴۰۰

پروژه دوم

مهلت تحویل ۱۶ آذر ۱۴۰۰

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 کد های 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
```

^۱ Minimax

^۲ Expectiminimax

^۳ Evaluation Function

به صورت پیش فرض اجرای `autograder.py` با گزینه `-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. همچنین گزارشی مختصر از نحوه پیاده‌سازی‌های انجام شده تهیه کرده و در نهایت این دو فایل را در سامانه بارگذاری خواهید کرد.

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 می بایست 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 می کند. برای بررسی محتوای اطلاعات فراهم شده می توانید آنها را چاپ کنید. در نهایت `ReflexAgent` باید با در نظر گرفتن موقعیت غذاها و روح ها، بهترین انتخاب را انجام دهد و در نقشه `testClassic` همواره برنده شود. برای بررسی این موضوع می توانید 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 راهنمایی ها:

- عامل شما به احتمال زیاد در بازی با دو روح شکست می خورد 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 رابطه ی بین آن ها مانند فاصله تا غذا، استفاده 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 می گیرد. اگر عامل حداقل ۵ بار برنده شود یک امتیاز و اگر هر ۱۰ بار برنده شود دو امتیاز دریافت می کنید. 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 می بایست هر ده بار برنده شده و میانگین امتیازات بالای ۱۰۰۰ باشد. برای ارزیابی می توانید 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 کلاس `MinimaxAgent` را کامل کنید. عامل شما باید به ازای هر تعداد روح درست کار کند که برای این کار باید به ازای هر روح یک لایه `min` و به ازای پکمن تنها یک لایه `max` داشته باشید.

درخت مینیماکس شما باید تا عمق دلخواه گسترش یابد و برگ های آن با تابع مناسب ارزیابی شوند. به این منظور کلاس `MinimaxAgent` از `MultiAgentSearchAgent` ارث می برد که `self.evaluationFunction` و `self.depth` را در خود دارد و باید حتماً از آنها برای 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 تغییر در این تابع 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 نقشه های بزرگ مانند `openClassic` و `mediumClassic` عامل پکمن مبتنی بر مینیماکس در فرار از مردن خوب عمل می کند اما در بردن ضعف دارد. عموماً عامل نمی تواند افقی را دورتر از خوردن غذاها ببیند به

همین خاطر خوردن یا نخوردن آنها اهمیت چندانی ندارد و بارها مشاهده می شود که در کنار یک dot می چرخد و در مسیر خلاف جهت حرکت می کند. این موضوع در سوال پنجم اصلاح خواهد شد.

- وقتی پکمن به این نتیجه برسد که مردن آن اجتناب ناپذیر است، تلاش می کند تا به منظور جلوگیری از کم شدن امتیاز، زودتر ببازد. این موضوع را می توانید 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 سریع تر است.

نحوه ارزیابی سوال دوم:

اجرای صحیح کد شما بستگی به این دارد که آیا تعداد درستی از حالت های بازی در درخت پیمایش می شود 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 می توانید از دستور زیر استفاده کنید:

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

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

(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 نتیجه ی این ارتقا باید شاهد افزایش سرعت باشید. این موضوع را می توانید با اجرای دستور زیر در عمق ۳ و مقایسه آن با عامل `MinimaxAgent` در عمق ۲ بررسی کنید. هر دو باید تقریباً در یک زمان به نتیجه برسند.

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

مقادیر مینیماکس در `AlphaBetaAgent` عیناً برابر با مقادیر `MinimaxAgent` می باشد 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.

نحوه ارزیابی سوال سوم:

مشابه با سوال دوم، برد و باخت بازی ملاک نمره دهی نیست بلکه ارزیابی بر اساس تعداد حالت های پیمایش شده در درخت مینیماکس انجام می گیرد. مهم است تا هرس آلفا-بتا را بدون تغییر در ترتیب فرزندان هر گره اعمال کنید. به عبارت دیگر با همان ترتیبی که action های مجاز را از `GameState.getLegalActions` بدست می آورید باید `GameState.generateSuccessor` را فراخوانی کنید و تعداد این فراخوانی ها نباید بیشتر یا کمتر از آنچه از هرس آلفا-بتا انتظار می رود باشد.

نکته مهم: در شرایط برابری هرس انجام ندهید تا تعداد حالت های کاوش شده با انتظار `autograder` همخوانی داشته باشد. پیاده سازی 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:  
         $v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$   
        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:  
         $v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$   
        if  $v < \alpha$  return  $v$   
         $\beta = \min(\beta, v)$   
    return  $v$ 
```

برای ارزیابی کد خود می توانید از دستور زیر استفاده کنید:

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

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


(۴) مینیماکس احتمالی (۵ امتیاز)

در مینیماکس و هرس آلفا بتا فرض می‌شود حریف بهینه‌ترین انتخاب‌ها را انجام می‌دهد در حالیکه در واقعیت این‌گونه نیست و مدل‌سازی احتمالی رفتار عاملی که ممکن است انتخاب‌های غیربهینه‌ای داشته باشد، می‌تواند نتیجه بهتری داشته باشد. ارواح تصادفی نیز انتخاب‌های بهینه ندارند بنابراین 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. برای ساده‌سازی مدل احتمالی تان فرض کنید ارواح حرکات‌شان را از بین ۴ حرکت مجاز شان به صورت یکنواخت و تصادفی انتخاب می‌کنند.

در این سوال باید تغییرات لازم را در کلاس `ExpectimaxAgent` وارد کنید. برای ارزیابی کد خود می‌توانید از دستور زیر استفاده کنید:

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

برای مشاهده عملکرد عامل خود می‌توانید از دستور زیر استفاده کنید:

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

همان‌طورکه در سوال دوم ذکر شد، وقتی پکمن به این نتیجه برسد که مردنش اجتناب ناپذیر است، تلاش می‌کند تا به منظور از جلوگیری از کم شدن امتیاز، زودتر ببازد. اما در این حالت خاص، اگر تلاش کند برای خوردن چند تکه غذای دیگر فرار کند، ممکن است بازی ادامه پیدا کند.

نحوه ارزیابی سوال چهارم:

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 روش امتحان کنید و درستی این گزاره را **نشان 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
```

(۵) تابع ارزیابی (۵ امتیاز)

یک تابع ارزیابی بهتر برای پکمن در `betterEvaluationFunction` بنویسید. این تابع ارزیابی باید به جای ارزیابی عمل (action)، حالت ها (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 ۱۰۰۰ امتیاز گرفته باشد.

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 نماینده ای که این معیارها را برآورده نکند صفر امتیاز دریافت می‌کند.

- 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 تمامی 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 و پیاده سازی انجام شده است، لذا **ضروری** است که تمامی دانشجویان گزارشی برای پروژه تهیه نموده و در آن به طور مختصر نحوه ی پیاده سازی های انجام شده را شرح دهند.
- فایل **multiAgents.py** به همراه گزارش را در قالب یک فایل فشرده با فرمت AI_P2_9931099.zip در سامانه کورسز آپلود کنید.
- در صورت هرگونه سوال یا ابهام از طریق ایمیل 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 [@Aminhbll](https://t.me/Aminhbll)
- ددلاین این پروژه **۱۶ آذر ۱۴۰۰ ساعت ۲۳:۵۵** است و امکان ارسال با تاخیر وجود ندارد، بنابراین بهتر است انجام پروژه را به روزهای پایانی موکول نکنید.