

به نام خدا



دانشکده مهندسی کامپیوتر

مبانی و کاربردهای هوش مصنوعی ترم پاییز ۱۴۰۲

پروژه دوم

مهلت تحویل ۱۲ آذر ۱۴۰۲

مقدمه

در این پروژه عاملی را برای بازی رولیت طراحی خواهید کرد که علاوه بر بازی ۲ نفره، بازی ۴ نفره را نیز پشتیبانی می‌کند. در این مسیر عوامل جستجوی مینیماکس^۱ و مینیماکس احتمالی^۲ را پیاده‌سازی خواهید کرد، با هرس آلفا-بتا آشنا خواهید شد و تابع ارزیابی^۳ طراحی خواهید کرد.

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

فایل‌هایی که باید ویرایش کنید:	
multiAgents.py	شامل تمامی عامل‌های جستجوی چند عاملی می‌باشد.
فایل‌هایی که شاید بخواهید آن‌ها را ببینید:	
Game.py	فایل اصلی که بازی‌های رولیت را اجرا می‌کند. این فایل حاوی GameStateData, GameState است که داده‌های هر بازی را نگهداری می‌کند و همچنین Game که بازی به وسیله آن اجرا می‌شود.

¹Minimax

²Expectimax

³Evaluation Function

Agents.py	این فایل حاوی کلاس پایه Agent است و همراه با آن، عامل‌هایی که می‌توانید برای خود انتخاب کنید مانند MouseAgent و KeyboardAgent نیز در آن قرار گرفته‌اند.
rollit.py	پارسی در این فایل تعریف شده‌است که دستورات شما را تعبیر و سپس یک نمونه از بازی را راه‌اندازی می‌کند. با مراجعه به این پارسر می‌توانید انواع دستورات مختلفی که می‌توانید استفاده کنید را مشاهده کنید.
فایل‌هایی که می‌توانید آن‌ها را رد کنید:	
Display.py	گرافیک‌های پیاده‌سازی شده برای بازی پکمن
Util.py	توابع کمکی استفاده شده در قسمت‌های مختلف
images/.	پوشه دربردارنده اسپریت‌های استفاده شده برای کاراکترها

آنچه باید انجام دهید:

ابتدا باید کد پایه پروژه را از مخزن <https://github.com/AUT-AI-Fall-2023/Project-2.git> دریافت کنید.

شما باید بخش‌هایی از فایل `multiAgents.py` را تغییر دهید.

توجه ۱: لطفاً سایر بخش‌های پروژه را به هیچ عنوان تغییر ندهید.

توجه ۲: برای بازی ۲ نفره عمق جستجوی عوامل را بیشتر از ۲ و برای بازی ۴ نفره بیشتر از ۱ انتخاب نکنید زیرا به جهت بزرگ بودن فضای جستجو، نیاز به زمان زیاد و توان پردازشی بالایی برای اجرای کامل دارد.

به رولیت چند عاملی خوش آمدید!

می‌توانید بازی را با فرمان زیر اجرا کنید:

```
python rollit.py -a MouseAgent
```

با اجرای این فرمان شما می‌توانید به وسیله کلیک کردن بر روی نقاط مختلف روی صفحه، مهره خود را آنجا قرار داده و به این وسیله با یک عامل از پیش تعریف شده، بازی کنید. همچنین می‌توانید با اضافه کردن فلگ `-n` تعداد بازیکنان را به ۴ تغییر دهید.

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

```
Python rollit.py -a ReflexAgent
```

مشاهده خواهید کرد که عامل به خوبی بازی نمی کند. همچنین برای تغییر به حالت بدون گرافیک می توانید توسط فلگ `-ds` گزینه `minimal` را انتخاب کنید.

کد `ReflexAgent` را در فایل `multiAgent.py` بررسی کنید تا متوجه شوید این تابع چگونه برای ارزیابی میزان مطلوب بودن یک عمل، از حالت فعلی و عمل انتخاب شده استفاده می کند.

سوال ۱: به نظر شما چرا این عامل در بازی اکثر اوقات شکست می خورد؟

(۱) مینیماکس (۵ امتیاز)

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

همچنین درخت مینیماکس شما باید تا عمق دلخواه گسترش یابد تا در آخرین سطح به برگ ها برسد. برگ ها باید به وسیله 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 `depth` و `evaluationFunction` را در خود دارد که از `depth` باید برای بررسی رسیدن به عمق دلخواه و از `evaluationFunction` برای ارزیابی برگ ها 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.
- برای بررسی عملکرد مینیماکس می توانید از دستور زیر استفاده کنید:

```
python rollit.py -q q1
```

- همچنین می توانید به وسیله فرمان زیر همین آزمایش را انجام دهید و در صورت نیاز، تغییراتی در عمق بررسی عامل خود نیز بدهید:

```
python rollit.py -a MinimaxAgent -d 2 -ea PartiallyRandomAgent -ds minimal
```

- برای مشاهده چگونگی عملکرد عامل، می‌توانید از `console` یا `graphic` برای فلگ `-ds` استفاده کنید.
- عامل شما همواره عامل با اندیس صفر می‌باشد و عامل‌های مختلف بر اساس افزایش اندیس باید حرکت کنند، یعنی ابتدا عامل شما و سپس حریفان شما.
- این موضوع را در طراحی مینیماکس در نظر داشته باشید که تمام وضعیت‌های بازی باید از جنس `GameState` باشند چه به صورت ورودی به تابع `getLegalActions` و یا خروجی از `gameState.generateSuccessor`.

۲) 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 به ازای هر گره `max` که برای عامل خود در نظر می‌گیرید، باید به تعداد حریفان گره `min` قرار دهید.

در نتیجه‌ی این ارتقا باید شاهد افزایش سرعت الگوریتم باشید. این موضوع را میتوانید با اجرای دستور زیر در عمق ۲ و مقایسه آن با عملکرد عامل `MinimaxAgent` در عمق ۲ مشاهده کنید. انتظار می‌رود عامل `AlphaBetaAgent` بسیار سریع‌تر به نتیجه برسد.

```
Python rollit.py -q q2
```

```
python rollit.py -a AlphaBetaAgent -d 2 -ea PartiallyRandomAgent -ds minimal
```

نکات و راهنمایی‌ها:

- با وجود آنکه نتیجه بدست آمده در ریشه درخت و همچنین مقادیر محاسبه شده توسط تابع ارزیابی در برگ‌ها در هر دو درخت یکسان است، با این حال به علت انجام هرس مشاهده خواهد شد که برخی گره‌های میانی مقادیری متفاوت دارند.

- می‌توانید برای پیاده‌سازی از سودوکد زیر استفاده نمایید.

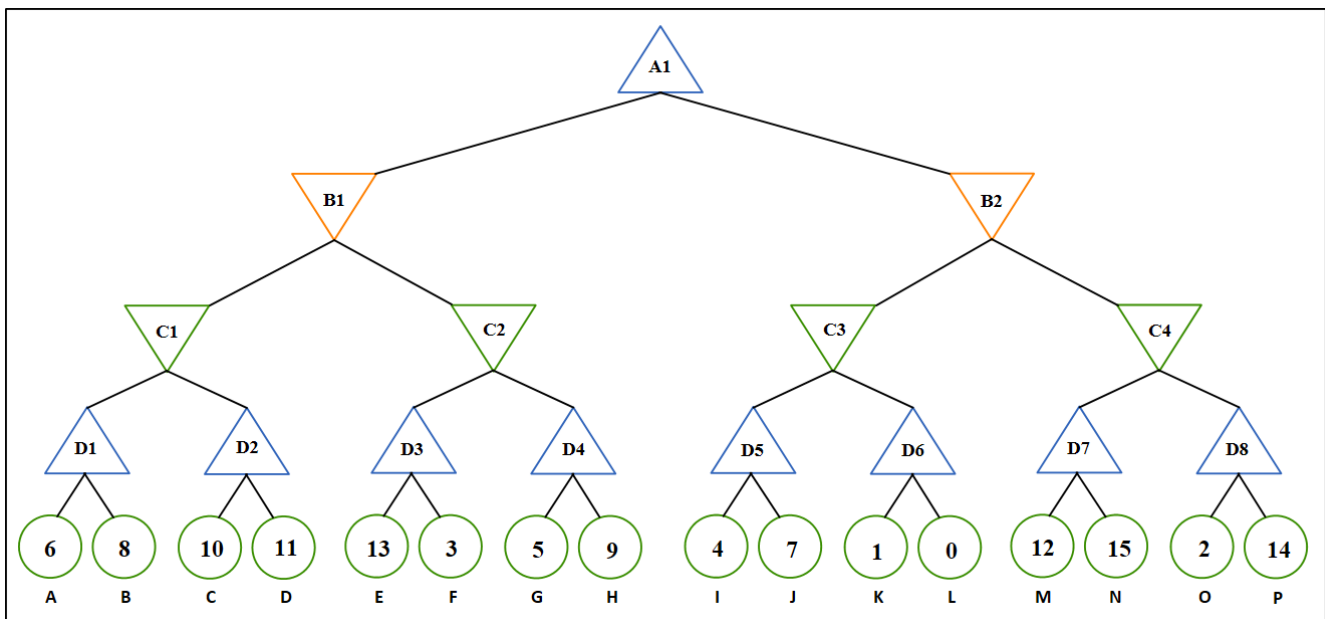
Alpha-Beta Implementation

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  $v$ 
```

سوال ۲: فرض کنید درخت زیر یکی از تست‌های داده‌شده به الگوریتم هرس آلفا-بتای شماست. گره‌های مربوط به عامل شما با مثلث آبی و گره‌های دو حریف شما با مثلث برعکس نارنجی رنگ و سبز رنگ نمایش داده شده‌اند. دایره‌ها نیز برگ‌های درخت هستند که حالت‌های A تا P را در آنها ارزیابی کرده و ارزش آنها را داخل این دایره‌ها نوشته‌ایم. الگوریتم مینیماکس خود به همراه هرس آلفا-بتا را برای این درخت اجرا کنید و مشخص کنید که کدام گره‌ها هرس می‌شوند و به چه دلیل. همچنین در پایان مشخص کنید که عامل شما در گره A1 کدام عمل را انتخاب می‌کند (آنکه منتهی به B1 است یا B2؟).



۳) 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 عاملی که انتخاب‌های غیربهینه دارد، ممکن است با نتیجه‌ی بهتری برای ما همراه باشد. عوامل تصادفی حریف 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 عامل حریف حرکات خود را از بین حرکات مجازشان به صورت یکنواخت و تصادفی انتخاب می‌کند. 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 rollit.py -q q3
```

```
python rollit.py -a ExpectimaxAgent -d 2 -ea PartiallyRandomAgent -ds  
minimal
```

سوال ۳: چه تفاوتی در عملکرد عامل خود نسبت به عامل `AlphaBetaAgent` احساس می‌کنید؟ درباره آن توضیح دهید. می‌توانید از فلگ `-ea` برای تعیین عامل حریف خود استفاده کنید و مشاهدات خود را برای عامل `MysteriousAgent` تکرار کنید. همچنین با `-ed` می‌توانید عمق جستجوی عامل حریف خود را نیز انتخاب کنید.

(۴) تابع ارزیابی (۵ امتیاز)

یک تابع ارزیابی بهتر برای پکمن در `betterEvaluationFunction` بنویسید. این تابع ارزیابی برخلاف آنچه که `ReflexAgent` استفاده می‌کرد، تنها حالت را ارزیابی می‌کند و نه جفت حالت و عمل. همراه دستور پروژه، یک مقاله نیز برای شما قرار گرفته است که به طراحی و بررسی چند مدل تابع ارزیابی برای بازی اوتلو پرداخته است. (An analysis of heuristics in othello 2015)

این مقاله به توضیح چهار هیوریستیک مختلف برای حالت دو نفره بازی رولیت پرداخته است. این چهار هیوریستیک عبارتند از `stability` و `mobility`، `corners`، `parity`. از شما خواسته شده است که این چهار هیوریستیک را پیاده‌سازی کنید. برخلاف سه هیوریستیک اول که تعریف همواره مشخصی دارند، برای طراحی هیوریستیک `stability` آزادی عمل دارید به همین خاطر سعی کنید از خلاقیت خود استفاده کنید. پس از طراحی این توابع، می‌توانید به صورت نظام‌مند به جستجوی وزن‌دهی مناسب برای آنها بپردازید تا سرانجام با ترکیب وزن‌دار آنها، به یک تابع ارزیابی مناسب دست پیدا کنید.

با اجرای فرمان‌های زیر می‌توانید تابع ارزیابی طراحی شده خود را بیازمایید.

```
python rollit.py -q q4
```

```
python rollit.py -a AlphaBetaAgent -d 2 -ea PartiallyRandomAgent -fn  
betterEvaluationFunction -ds minimal
```

توضیحات تکمیلی

- کد مربوط به پروژه را می‌توانید از مخزن زیر دریافت کنید:
<https://github.com/AUT-AI-Fall-2023/Project-2.git>
- پاسخ به تمرین‌ها 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.
- فایل `multiAgents.py` به همراه پاسخ خود به سوالات که در فایل به شکل **سوال** مشخص شده‌اند را در قالب یک فایل فشرده با فرمت `AI_P2_[StudentNumber].zip` در سامانه کورسز آپلود کنید.
- در صورت هرگونه سوال یا ابهام از طریق ایمیل AUT.AI.Fall2023@gmail.com با تدریس‌یاران 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:
 - Pmoonesi@
 - sam_re@
- توجه شود که این پروژه تحویل آنلاین دارد. در روز تحویل فایل `multiAgents.py` را از کورسز دریافت خواهید کرد بنابراین سعی کنید کدی که حاوی آخرین تغییرات شما است را در کورسز بارگزاری نمایید.
- ددلاین این پروژه ۱۲ آذر ۱۴۰۲ ساعت ۲۳:۵۵ است. بنابراین بهتر است انجام پروژه را به روزهای پایانی موکول نکنید.