تمرین سری سوم: جست‌وجوی رقابتی

لطفاً به نکات زیر توجه کنید:

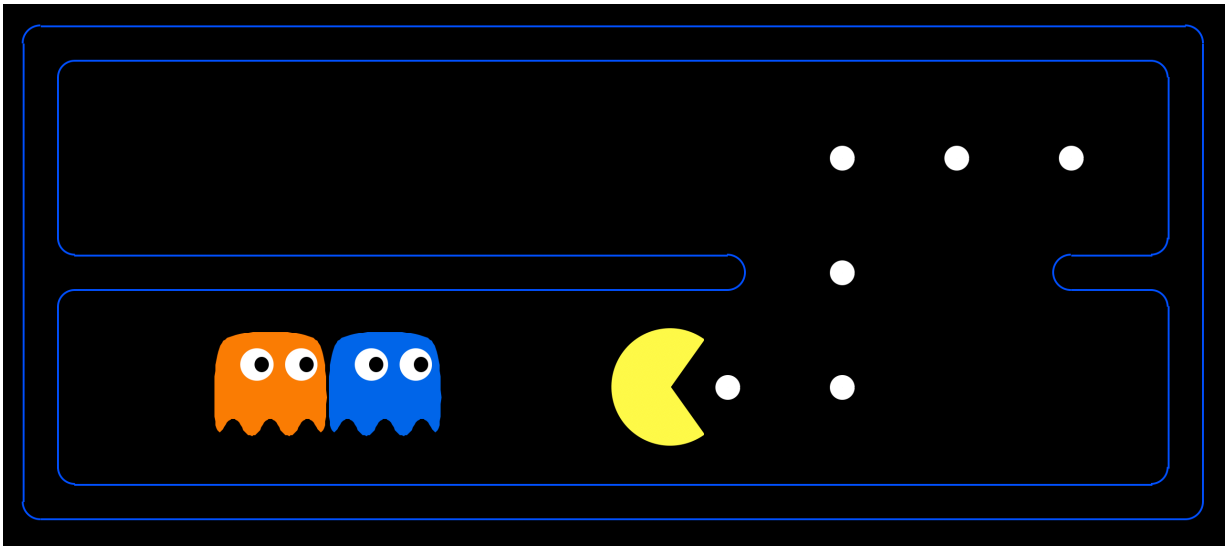
- مهلت ارسال این تمرین برای هر دو گروه **۱۷ آبان ماه** است.
- در صورتی که به اطلاعات بیشتری نیاز دارید می‌توانید به صفحه‌ی تمرین در وب‌سایت درس مراجعه کنید.

- این تمرین شامل سوال‌های برنامه‌نویسی می‌باشد، بنابراین توجه کنید که حتماً موارد خواسته‌شده در سوال را رعایت کنید. در صورتی که به هر دلیلی سامانه‌ی داوری نتواند آن را اجرا کند مسئولیت آن تنها به عهده‌ی شماست.
- ما همواره هم‌فکری و هم‌کاری را برای حلِ تمرین‌ها به دانشجویان توصیه می‌کنیم. اما هر فرد باید تمامی سوالات را به تنهایی تمام کند و پاسخ ارسالی حتماً باید توسط خود دانش‌جو نوشته‌شده باشد. لطفاً اگر با کسی هم‌فکری کردید نام او را ذکر کنید. در صورتی که سامانه‌ی تطبیق، تقلبی را تشخیص دهد متأسفانه هیچ مسئولیتی بر عهده‌ی گروه تمرین نخواهد بود.
- لطفاً برای ارسال پاسخ‌های خود از راهنمای موجود در صفحه‌ی تمرین استفاده کنید.
- هر سؤالی درباره‌ی این تمرین را می‌توانید در گروه درس مطرح کنید و یا از دستیاران حلِ تمرین بپرسید.

- آدرس صفحه‌ی تمرین: https://iust-courses.github.io/ai97/assignments/03_adversarial_search
- آدرس گروه درس: https://groups.google.com/forum/#!forum/ai97

موفق باشید

سؤال‌ها



۱. درخت جست‌وجوی مینی‌مکس (۲۵ نمره)

در کلاس، الگوریتم جست‌وجوی مینی‌مکس[1] را یاد گرفتیم. حال می‌خواهیم آن را پیاده‌سازی کنیم. در این سری از تمرین‌ها دوباره با فریم‌ورک پک‌من کار خواهیم داشت. لطفاً فایل زیپ را از صفحه‌ی تمرین دانلود و آن را از حالت فشرده خارج کنید. هدف این سری، آزمایش و پیاده‌سازی الگوریتم مینی‌مکس در محیط پک‌من است تا به عامل‌هایی با رفتار هوشمندانه‌تر برسیم.

وظیفه‌ی شما تنها پیاده‌سازی یک عامل هوشمند است که کنترل شخصیت پک‌من یا یکی از روح‌ها را بر عهده می‌گیرد. کلاس Agent به همین منظور تعبیه شده است. در هر مرحله موتور بازی وضعیت همه‌ی المان‌های بازی را محاسبه می‌کند و سپس با فراخوانی متد `getAction` از این کلاس و هم‌چنین پاس‌دادن وضعیت زمین به آن، حرکت بعدی عامل را درخواست می‌کند. موتور بازی این روند را برای تمامی عامل‌های موجود (پک‌من و تمامی روح‌ها) به ترتیب اعمال می‌کند. پس از اینکه آخرین عامل حرکت خود را انجام داد موتور بازی دوباره به عامل اول باز می‌گردد بدین ترتیب یک مرحله از بازی سپری می‌شود.

حال می‌خواهیم عاملی پیاده‌سازی کنیم که در هر مرحله با استفاده از الگوریتم مینی‌مکس امتیاز همه‌ی حرکت‌های مجاز را به دست بیاورد و سپس از بین آن‌ها بهترین را انتخاب کند. در این سری از سوال‌ها فرض می‌کنیم فقط دو عامل در زمین وجود دارند: پک‌من و یک روح. بنابراین هر لایه از درخت مینی‌مکس فقط مربوط به یکی از این عامل‌هاست.

[1] Minimax

برای پاسخ به این سوال باید بدنه‌ی مِتُد `minimax(...)` را از کلاس `MinimaxAgent` در فایل adversarialAgents.py پر کنید. خروجی تابع، مقدار مینی‌مکس مربوطه است. توضیحات هر کدام از پارامترها در کد آمده‌است. برای آزمایش درستی کد خود از دستور زیر استفاده کنید:

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

اگر کد شما درست باشد باید امتیاز ۲۵ از ۲۵ بگیرد. اما برای اینکه عملکرد این عامل را در بازی ببینیم از دستور زیر استفاده کنید. دقت کنید که مهم‌ترین بخش در عامل‌های بر پایه‌ی مینی‌مکس، **تابع ارزیابی** آن است. این تابع با ورودی گرفتن یک حالت از بازی، میزان مناسب بودن آن حالت را از نگاه عاملی که برای آن نوشته شده است خروجی می‌دهد. اما در این سوال از تابع ارزیابی پیش‌فرض (و البته ضعیف) استفاده می‌کنیم. بنابراین ممکن است در بعضی حالات حتی پک‌من ببازد و یا بی‌هدف در زمین بماند. در سوال‌های بعد تابع ارزیابی قوی‌تری پیاده‌سازی می‌کنیم.

```
$ python pacman.py -l smallClassic -k 1 -p MinimaxAgent
```

لطفاً دقت کنید که کد شما حتماً باید عمق درخت را از متغیر `self.depth` بخواند. مطمئن شوید پیاده‌سازی شما عمومی است و می‌توان از آن هم در پک‌من و هم در روح‌ها استفاده کرد، برای این‌کار می‌توانید از `self.index` و `self.getOpponentIndex()` استفاده کنید.

۲. پک‌منِ واقعاً هوشمند (۲۵ نمره)

در سوال قبل الگوریتم مینی‌مکس را پیاده‌سازی کردیم. اما برای اینکه پک‌من واقعاً رفتار هوشمندی از خود نشان دهد باید تابع ارزیابی قوی‌تری برای آن داشته باشیم. همان‌طور که گفته شد وظیفه‌ی تابع ارزیابی، بررسی میزان خوب‌بودن یک حالتِ بازی از نگاه اجرای کننده‌ی آن (در این سوال پک‌من) است. به طور مثال هرچه در یک حالت، تعداد غذاهای خورده شده، میزان نزدیکی به سایر غذاها و ... بهتر باشد باید خروجی تابع ارزیابی بزرگ‌تر شود.

برای پاسخ به این سوال باید بدنه‌ی مِتُد `evaluationFunction(...)` را از کلاس `SmartPacmanAgent` در فایل adversarialAgents.py پر کنید. خروجی تابع، عددی برای میزان خوب بودن حالت ورودی است. توضیحات هر کدام از پارامترها در کد آمده‌است. برای اجرای کد خود از دستور زیر استفاده کنید:

```
$ python pacman.py -l smallClassic -k 1 -p SmartPacmanAgent -a depth=2
```

کد شما نباید در هیچ حالتی ببازد (دقت کنید که به صورت پیش‌فرض روح‌ها حرکت تصادفی دارند) و هم‌چنین باید تمامی غذاهای روی زمین را هم بخورد.

اما ارزیابی اصلی کد شما با استفاده از روح‌های هوشمندتری خواهد بود. حال برای آزمایش نهایی کد خود دستور زیر را اجرا کنید:

```
$ python pacman.py -l smallClassic -k 1 -p SmartPacmanAgent -g DirectionalGhost
```

برای دریافت امتیاز کامل از این سوال، کد شما باید از هر ۱۰ بار اجرای بازی حداقل ۸ بار آن را برنده شود.

برای پیاده‌سازی خود می‌توانید از توابع و داده‌ساختارهای موجود در util.py استفاده کنید.

۳. روحِ واقعاً هوشمند (۲۵ نمره)

در سوال قبلی تابع ارزیابی را برای بُرد پک‌من پیاده‌سازی کردیم. حال در این قسمت می‌خواهیم کنترل روح را به یک عامل مینی‌مکسی بدهیم. از سوال یک، الگوریتم مینی‌مکس را پیاده‌سازی‌شده داریم، بنابراین در این‌جا کافیست تابع ارزیابی جدیدی طراحی کنیم.

برای پاسخ به این سوال باید بدنه‌ی مِتُد `evaluationFunction(...)` را از کلاس `SmartGhostAgent` در فایل adversarialAgents.py پر کنید. خروجی تابع، عددی برای میزان خوب بودن حالت ورودیست. توضیحات هر کدام از پارامترها در کد آمده‌است. برای اجرای کد خود از دستور زیر استفاده کنید:

```
$ python pacman.py -l smallClassic -k 1 -p GreedyAgent -g SmartGhostAgent
```

عاملی که پیاده‌سازی می‌کنید باید در تمامی حالات جلوی بُردن پک‌من را بگیرد.

۴. روح‌های واقعاً هوشمند (امتیازی، ۲۵ نمره)

در سوال‌های قبل، فقط دو عامل در بازی وجود داشت. حال در این سوال می‌خواهیم به جای یک روح، دو روح داشته باشیم. بنابراین عامل‌های موجود در بازی شامل پک‌من و دو روح خواهد بود. شما باید عاملی طراحی کنید که کنترل هر یک از روح‌ها را به دست بگیرد.

در این سوال شما کاملاً آزادید از هر الگوریتمی استفاده کنید. دقت کنید عاملی که طراحی می‌کنید به صورت جداگانه برای هر کدام از روح‌ها اجرا می‌شود. برای پاسخ به این سوال باید کلاس `SuperGhostAgent` را در فایل adversarialAgents.py پر کنید. برای اجرای کد خود از دستور زیر استفاده کنید:

```
$ python pacman.py -l mediumClassic -p SPAgent -g SuperGhostAgent -k 2 --frameTime=0
```

کد شما باید از هر ۱۰ بازی حداقل ۸ بار جلوی برد پک‌من را بگیرد. اگر از مینی‌مکس استفاده می‌کنید حداکثر عمق مجاز ۲ می‌باشد. آیا می‌توانید شکلی از همکاری را بین روح‌های خود مشاهده کنید؟ آیا می‌توان بدون برنامه‌ریزی قبلی به شکلی از همکاری بین آن‌ها رسید؟

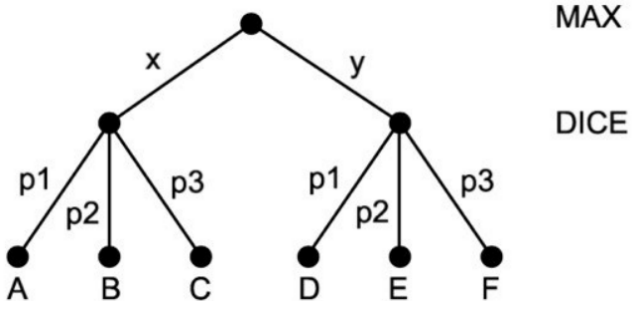
۵. عنصر شانس (۲۵ نمره)

الگوریتم ExpectiMax یک روش منشا گرفته از الگوریتم مینی‌مکس است که معمولاً برای بازی‌هایی که در آن‌ها عنصر شانس وجود دارد استفاده می‌شود. رویکرد زیر را در تحلیل بازی‌ها با یک عنصر شانس در نظر بگیرید. فرض کنید که در این جا عنصر شانس خروجی دو تاس است.

- تعدادی دنباله از خروجی تاس‌ها ایجاد کنید، به صورتی که هر دنباله متشکل از چند بار انداختن تاس است (به اندازه کافی برای تعداد حرکاتی که می‌خواهید در جستجو کشف کنید).
- برای هر دنباله از تاس‌ها، شما می‌توانید از مینی‌مکس معمولی (با یا بدون هرس آلفا-بتا) برای تجزیه و تحلیل بازی استفاده کنید و مشخص کنید که کدام حرکت باید توسط Max انجام شود.
- هر کدام از دنباله‌ها به یک حرکت برای Max رای می دهد؛ در نهایت حرکتی که بیشترین رای را دارد انتخاب کنید.

این رویکرد مسیرهایی از درخت بازی کامل را نمونه‌برداری می‌کند. اگرچه این رویکرد در عمل به خوبی کار می‌کند، اما درست نیست.

نمودار بازی زیر را در نظر بگیرید. Max یکی از دو حرکت x یا y را انتخاب می‌کند. سپس یک تاس با سه خروجی به احتمالات p1 و p2 و p3 ریخته می‌شود. در نهایت حالت بازی ارزیابی می‌شود و به یکی از حالت‌های A، B، C، D، E و یا F تبدیل می‌شود.



حال به پرسش‌های زیر پاسخ دهید:

الف) یک عبارت ریاضی برای ارزش گره MAX تحت الگوریتم ExpectiMax معمولی بنویسید.

ارزش گره MAX تحت الگوریتم ExpectiMax برابر است با:

$$Vmax = MAX\ \{(P1.A + P2.B + P3.C)\ ,\ (P1.D + P2.E + P3.F)\}$$

ب) مقادیر $\{p1, p2, p3\}$ و $\{A, B, C, D, E, F\}$ را طوری تعیین کنید که الگوریتم بالا خروجی متفاوتی نسبت به الگوریتم ExpectiMax معمولی داشته باشد. همچنین نشان دهید که این اعداد طبق رفتار بالا عمل می‌کنند.

فرض می‌کنیم که:

$$A = 1\ ,\ B = 2\ ,\ C = 3\ ,\ D = 3\ ,\ E = 1\ ,\ F = 1$$

و

$$P1 = 0.5\ ,\ P2 = 0.3\ ,\ P3 = 0.2$$

خروجی طبق الگوریتم ExpectiMax برابر است با:

$$Max\ \{x(0.5 + 0.6 + 0.6)\ ,\ y(1.5 + 0.3 + 0.2)\} =$$
$$Max\ \{x(1.7)\ ,\ y(2)\} = y(2)$$

مقدار شاخه‌ی سمت راست (۲) بیشتر از مقدار شاخه‌ی سمت چپ (۱.۷) است. بنابراین حرکت y انتخاب می‌شود.

اما اگر از روشی که در این سوال مطرح شده استفاده کنیم، ممکن است اتفاق زیر یکی از جواب‌هایی باشد که بدست می‌آید:

فرض می‌کنیم ۳بار درخت Minimax را درست می‌کنیم (یعنی ۳بار تاس می‌ریزیم).

تکرار اول:

مقادیر $C(3)$ و $F(1)$ اتفاق می‌افتند. بنابراین حرکت x انتخاب می‌شود زیرا:

$$Max\ \{x(3)\ ,\ y(1)\} = x(3)$$

تکرار دوم:

مقادیر $A(1)$ و $D(3)$ اتفاق می‌افتند. بنابراین حرکت y انتخاب می‌شود زیرا:

$$Max\ \{x(1)\ ,\ y(3)\} = y(3)$$

تکرار سوم:

مقادیر $C(3)$ و $E(1)$ اتفاق می‌افتند. بنابراین حرکت x انتخاب می‌شود زیرا:

$$\text{Max } \{x(3) \ , \ y(1)\} = x(3)$$

در نتیجه به دلیل این‌که ۲بار حرکت x و ۱بار حرکت y انتخاب شده است، خروجی الگوریتم می‌شود حرکت x که با خروجی الگوریتم Expectimax متفاوت است.