به نام خدا

دانشکده مهندسی کامپیوتر

مبانی و کاربردهای هوش مصنوعی ترم پاییز ۱۴۰۰

پروژه چهارم

مهلت تحویل ۱۰ تیر ۱۴۰۱

مقدمه

پکمن روزهای خود را به فرار از دست روح‌ها می‌پردازد. اما طبق افسانه‌ها سال‌ها پیش پدر پدربزرگ پکمن گرندپک یاد گرفت تا روح‌ها را برای تفریح شکار کند.

در این پروژه شما پکمنی را طراحی می‌کنید که از سنسورهایی برای مکان‌یابی و خوردن روح‌های نامرئی استفاده می‌کند. شما از مکان‌یابی یک روح ثابت شروع کرده و تا شکار گروه‌های روح در حال حرکت پیش می‌روید.

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

فایل‌هایی که باید ویرایش کنید:	
bustersAgents.py	عامل‌های بازی نسخه‌ی گوست‌باستر از پکمن
inference.py	کد مربوط به پیدا کردن روح‌ها با استفاده از صدای آن‌ها
فایل‌هایی که نباید ویرایش کنید:	
busters.py	ورودی اصلی به بازی گوست‌باستر (جایگزین pacman.py)
bustersGhostAgents.py	عامل‌های روح جدید برای گوست‌باسترها

distanceCalculator.py	فاصله‌ی ماز را محاسبه می‌کند
game.py	کلاس‌های اصلی و کمکی به بازی پکمن
ghostAgents.py	عامل‌های کنترل‌کننده روح‌ها
graphicsDisplay.py	گرافیک‌های پیاده‌سازی شده برای بازی پکمن
graphicsUtils.py	پشتیبانی برای گرافیک بازی
keyboardAgents.py	رابط صفحه کلید برای کنترل پکمن
layout.py	برنامه برای خواندن فایل‌های نقشه و ذخیره اطلاعات آن‌ها
util.py	توابع کاربردی

آنچه باید انجام دهید:

شما باید بخش هایی از فایل `bustersAgents.py` و `inference.py` را **تغییر دهید**.

لطفا سایر بخش های پروژه را **به هیچ عنوان** تغییر ندهید.

گوست‌باستر!

در این پروژه هدف شکار روح‌های ترسیده اما نامرئی است. پکمن، نابینا است! اما مجهز به سونار (سیستم شنوایی) می‌باشد که می‌تواند فاصله منهتن از هر روح را بر اساس صدا ارائه دهد. بازی زمانی به پایان می رسد که پکمن تمام روح‌ها را خورده باشد. برای شروع، سعی کنید خودتان یک بازی را با استفاده از صفحه کلید انجام دهید.

```
python busters.py
```

با توجه به فاصله‌های ارائه شده به پکمن، بلوک‌های رنگی نشان می‌دهند که هر روح احتمالاً کجا می‌تواند باشد. فواصل بر اساس صدا (noisy distance) در پایین نمایشگر همیشه غیرمنفی و در بازه اختلاف ۷ تایی از فاصله واقعی هستند. احتمال یک فاصله با اختلاف آن از فاصله واقعی به طور تصاعدی کاهش می یابد.

وظیفه اصلی شما در این پروژه ردیابی روح‌ها است. برای بازی مبتنی بر صفحه‌کلید، یک مدل خام استنتاجی به‌طور پیش‌فرض برای شما پیاده‌سازی شده است: همه مربع‌هایی که احتمالاً یک روح در آن‌ها وجود دارد، با رنگ روح سایه زده شده است. ما می خواهیم تخمین بهتری از موقعیت روح داشته باشیم. در این راستا شبکه های بیزین[1] ابزارهای قدرتمندی برای استفاده بهینه از اطلاعاتی که در اختیار داریم را ارائه می دهد. در طول این پروژه، شما الگوریتم هایی را برای انجام استنتاج دقیق (exact inference) و تقریبی (approximate inference) با استفاده از شبکه های بیز پیاده سازی خواهید کرد.

نکته مهم: برای این پروژه، گاهی اوقات ممکن است که autograder در صورت اجرای تست ها به همراه گرافیک، timeout کند. برای تعیین کارآمودی کد خود می‌توانید تست ها را با پرچم `--no-graphics` اجرا کنید. اگر autograder با این پرچم اجرا شود، حتی در صورت timeout امتیاز کامل دریافت خواهید کرد.

[1] Bayes' Nets

۱) احتمال مشاهده (دو امتیاز)

در این سوال شما متد `getObservationProb` را در کلاس پایه `InferenceModule` در `inference.py` پیاده سازی خواهید کرد. این متد یک مشاهده (observation) فاصله تا روح، بدست آمده از صدای آن)، موقعیت پکمن، موقعیت روح، و موقعیت زندان روح را ورودی می گیرد و با توجه به موقعیت پکمن و روح، احتمال فاصله (noisy distance) را بر می‌گرداند. به عبارت دیگر، ما می خواهیم `P(noisyDistance | pacmanPosition,)` `ghostPosition` را محاسبه کنیم.

با توجه به فاصله واقعی پکمن تا روح، سنسور فاصله دارای توزیع احتمال (probibality distribution) بر روی فاصله است. این توزیع توسط تابع `busters.getObservationProbability(noisyDistance,)` `trueDistance` مدل شده است که `P(noisyDistance | trueDistance)` را برمی گرداند. شما باید از این تابع برای حل سوال استفاده کنید و از تابع `manhattanDistance` در ماژول `util` ارائه شده برای پیدا کردن فاصله بین مکان پکمن و مکان روح استفاده کنید.

همچنین یک مورد خاص قرار گرفتن روح در زندان پس از شکار شدن وجود دارد که باید در نظر گرفته شود. به طور خاص، هنگامی که یک روح را شکار می‌کنیم و آن را به محل زندان می فرستیم، سنسور فاصله ما `None` را برمی گرداند. بنابراین، اگر موقعیت روح موقعیت زندان باشد، مشاهده `None` با احتمال ۱ و هر چیز دیگر با احتمال ۰ است. بالعکس، اگر فاصله گزارش شده `None` نباشد، روح با احتمال ۰ در زندان است. مطمئن شوید که این مورد خاص را در اجرای خود مدیریت می‌کنید.

برای تست کردن کد خود و اجرای autograder برای این سوال از دستور زیر استفاده کنید:

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

ممکن است که اجرای autograder زمان زیادی طول بکشد تا جواب شما را ارزیابی شود.

۲) مشاهده استنتاج دقیق (سه امتیاز)

در این سوال، شما متد `observeUpdate` را در کلاس `ExactInference` از `inference.py` پیاده‌سازی می‌کنید تا با یک مشاهده از سنسورهای پکمن، توزیع باور (belief distribution) عامل بر روی موقعیت‌های روح‌ها را به‌درستی به‌روزرسانی کنید. با این کار باورها بر اساس شواهد جدید به روزرسانی می‌شوند. در این سوال، متد مشاهده (observe)

باید، باور را در هر موقعیت روی نقشه پس از دریافت آنچه توسط سنسور حس شده است، به روز کند. باید به‌روزرسانی‌های خود را روی متغیر `self.allPositions` که شامل تمام موقعیت‌های مجاز به اضافه موقعیت ویژه زندان است، تکرار کنید. باورها نشان دهنده احتمال آن هستند که روح در یک مکان خاص باشد و به عنوان یک شی `DiscreteDistribution` و در فیلدی به نام `self.beliefs` ذخیره می شود که شما باید آن را بروز کنید.

قبل از نوشتن کد، معادله مسئله استنتاجی را که می خواهید حل کنید بنویسید. شما باید از تابع `self.getObservationProb` که در سوال قبل نوشتید استفاده کنید که احتمال یک مشاهده را با توجه به موقعیت پکمن، یک موقعیت بالقوه برای روح و موقعیت زندان برمی‌گرداند. شما می توانید موقعیت پکمن را با استفاده از `gameState.getPacmanPosition()` و موقعیت زندان را با استفاده از `self.getJailPosition()` بدست آورید.

در صفحه نمایش بازی پکمن، posterior belief های با مقدار بالا به رنگ های روشن نشان داده می شوند، در حالی که beliefهای با مقدار پایین با رنگ‌های تیره‌تر نشان داده می‌شوند. شما باید با ابر بزرگی از belief شروع کنید که به مرور زمان با جمع شدن شواهد بیشتر کاهش می یابد.

برای تست کردن کد خود و اجرای autograder برای این سوال از دستور زیر استفاده کنید:

```
python autograder.py -q q2

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

۳) استنتاج دقیق با گذشت زمان (سه امتیاز)

در سوال قبلی شما به روز رسانی باور را برای پکمن بر اساس مشاهدات آن پیاده‌سازی کردید. خوشبختانه، مشاهدات پکمن تنها منبع دانش او در مورد جایی که یک روح ممکن است باشد نیست. پکمن همچنین در مورد راه هایی که یک روح ممکن است حرکت کند، آگاهی دارد. یعنی می‌داند که روح نمی تواند در یک مرحله زمانی از دیوار عبور کند یا بیش از یک قدم بردارد.

برای درک اینکه چرا این دانش برای پکمن مفید است، سناریوی زیر را در نظر بگیرید که در آن پکمن و یک روح وجود دارد. پکمن مشاهدات زیادی دریافت می کند که نشان می دهند روح بسیار نزدیک است، اما یکی از مشاهدات نشان می دهد که روح بسیار دور است. این مشاهده احتمالاً نتیجه یک سنسور خراب است. دانش قبلی پکمن از نحوه حرکت روح، تأثیر این مشاهده را کاهش می دهد زیرا پکمن می داند که روح نمی تواند تنها در یک حرکت تا این مقدار حرکت کرده باشد.

در این سوال شما متد `elapseTime` در `ExactInference` را پیاده سازی خواهید کرد. در این مسئله، `elapseTime` باید باور را در هر موقعیت روی نقشه پس از سپری شدن یک مرحله زمانی (time step) به روز کند. عامل شما از طریق `self.getPositionDistribution` به توزیع عمل (action distribution) برای روح دسترسی دارد. برای به دست آوردن توزیع بر روی موقعیت های جدید برای روح، با توجه به موقعیت قبلی آن، از این خط کد استفاده کنید:

```
newPosDist = self.getPositionDistribution(gameState, oldPos)
```

به طوری که `oldPos` به موقعیت قبلی روح اشاره دارد. `newPosDist` یک شی `DiscreteDistribution` است که برای هر موقعیت `p` در `self.allPositions`، احتمال آن که روح در زمان `t + 1` در موقعیت `p` باشد به شرط آنکه در زمان `t` در موقعیت `oldPos` بوده است، از `newPosDist[p]` بدست می‌آید. این فراخوانی زمان‌بر است پس اگر با timeout مواجه می‌شوید، ممکن است تعداد فراخوانی‌های `self.getPositionDistribution` در کد شما بیش از حد باشد.

قبل از نوشتن کد، معادله مسئله استنتاجی را که می خواهید حل کنید بنویسید. برای اینکه پیاده سازی پیش بینی خود را جدا از پیاده سازی به روز رسانی در سوال قبلی آزمایش کنید، این سوال از اجرای به روز رسانی شما استفاده نمی کند.

از آنجایی که پکمن روح را به طور مستقیم مشاهده نمی کند، این بدان معناست که اعمال روح بر باورهای پکمن تأثیری نخواهد داشت. با گذشت زمان، باورهای پکمن منعکس کننده مکان هایی روی تخته خواهند شد که پکمن معتقد است که روح‌ها به احتمال زیاد در این موقعیت ها حضور دارند و آنچه که پکمن از قبل در مورد حرکات معتبر آنها می داند داده می شود.

برای تست کردن کد خود و اجرای autograder برای این سوال از دستور زیر استفاده کنید:

```
python autograder.py -q q3

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

همانطور که خروجی autograder را بررسی می‌کنید، به یاد داشته باشید که مربع‌های روشن‌تر نشان می‌دهند که پکمن باور دارد احتمال آنکه یک روح آن مکان را اشغال کند بیشتر است، و مربع‌های تیره‌تر نشان می‌دهد که به باور پکمن احتمال اشغال شدن آن خانه توسط روح کمتر است.

۴) استنتاج دقیق با تست کامل (دو امتیاز)

اکنون که پکمن می‌داند چگونه از دانش قبلی و مشاهدات خود برای تشخیص اینکه یک روح کجاست استفاده کند، آماده است تا روح‌ها را به تنهایی شکار کند. این سوال از پیاده سازی `observeUpdate` و `elapseTime` همراه با یک استراتژی شکار حریصانه استفاده می کند که برای این سوال شما این استراتژی را پیاده سازی خواهید کرد. در این استراتژی ساده حریصانه، پکمن مطابق با باور خود فرض می کند که هر روح در محتمل‌ترین موقعیت خود قرار دارد، سپس به سمت نزدیک‌ترین روح حرکت می‌کند. تا این مرحله، پکمن در واقع با انتخاب تصادفی یک عمل معتبر حرکت کرده است.

متد `chooseAction` را در `GreedyBustersAgent` در `ustersAgents.py` پیاده‌سازی کنید. عامل شما ابتدا باید محتمل ترین موقعیت هر روح دستگیر نشده را پیدا کند، سپس عملی را انتخاب کند که فاصله تا نزدیکترین روح را به حداقل برساند.

برای یافتن فاصله بین هر دو موقعیت `pos1` و `pos2`، از `self.distancer.getDistance(pos1, pos2)` استفاده کنید. برای یافتن موقعیت جانشین (successor) یک موقعیت پس از یک عمل از کد زیر استفاده کنید:

```
successorPosition = Actions.getSuccessor(position, action)
```

`livingGhostPositionDistributions` لیستی از اشیاء `DiscreteDistribution` است که توزیع باورهای موقعیت برای هر یک از روح هایی که هنوز دستگیر نشده اند را نشان می‌دهد.

در صورتی که پیاده سازی به درستی انجام شود، عامل شما باید بازی را در تست `q4/3-gameScoreTest` با امتیاز بیشتر از ۷۰۰ حداقل ۸ بار از ۱۰ بار برنده شود.

برای تست کردن کد خود و اجرای autograder برای این سوال از دستور زیر استفاده کنید:

```
python autograder.py -q q4

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

۵) تقریب استنتاج اولیه و باورها (دو امتیاز)

استنتاج تقریبی (Approximate inference) در بین شکارچیان روح‌ها در این فصل بسیار مرسوم است. برای چند سوال بعدی، یک الگوریتم فیلتر ذرات (participle filtering algorithm) را برای ردیابی یک روح پیاده‌سازی خواهید کرد.

ابتدا توابع `initializeUniformly` و `getBeliefDistribution` را در کلاس `ParticleFilter` در `inference.py` پیاده‌سازی کنید. یک ذره(نمونه)، یک موقعیت روح در این مسئله است. توجه داشته باشید که برای مقداردهی اولیه، ذرات باید به طور برابر (نه به طور تصادفی) در موقعیت‌های مجاز توزیع شوند تا از یک uniform prior اطمینان حاصل شود.

توجه داشته باشید که متغیری که ذرات خود را در آن ذخیره می کنید باید یک لیست باشد. ذخیره کردن ذرات در هر نوع داده دیگری، مانند دیکشنری، نادرست است و باعث ایجاد خطا می‌شود. متد `getBeliefDistribution` لیستی از ذرات را می‌گیرد و آن را به یک شی `DiscreteDistribution` تبدیل می‌کند.

برای تست این سوال از دستور زیر استفاده کنید:

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

۶) مشاهده استنتاج تقریبی (سه امتیاز)

در این سوال، متد `observeUpdate` را در کلاس `ParticleFilter` در `inference.py` پیاده سازی می‌کنیم. این متد، توزیع وزنی (weight distribution) را روی `self.particles` ایجاد می‌کند طوری که وزن یک ذره برابر با احتمال مشاهده با توجه به موقعیت پکمن و مکان ذره است. سپس از این توزیع وزنی نمونه‌برداری می‌کنیم تا لیست جدید ذرات خود را بسازیم.

شما باید دوباره از تابع `self.getObservationProb` برای یافتن احتمال مشاهده با توجه به موقعیت پکمن، یک موقعیت بالقوه روح و موقعیت زندان استفاده کنید. می توانید از روش نمونه برداری در کلاس `DiscreteDistribution` نیز استفاده کنید. موقعیت پکمن را با استفاده از `gameState.getPacmanPosition` و موقعیت زندان را با استفاده از `self.getJailPosition` بدست آورید.

یک مورد خاص وجود دارد که حین پیاده سازی باید به آن توجه شود. وقتی همه ذرات وزن صفر دریافت می کنند، لیست ذرات باید با فراخوانی `initializeUniformly` مجدداً مقداردهی شود. می‌توانید از روش کلی کلاس `DiscreteDistribution` نیز استفاده کنید.

برای تست این سوال می‌توان از دستور زیر استفاده کرد:

```
python autograder.py -q q6

python autograder.py -q q6 --no-graphics
```

۷) استنتاج تقریبی با گذشت زمان (سه امتیاز)

در این سوال باید تابع `elapseTime` را در کلاس `ParticleFilter` در `inference.py` پیاده سازی کنید. این تابع باید لیست جدیدی از ذرات را بسازد به طوری که هر ذره معادل با ذره موجود در `self.particles` است که یک مرحله زمانی پیش رفته است، و سپس این لیست جدید را در `self.particles` قرار دهید. در انتها باید بتوانید روح‌ها را تا حد خوبی مانند استنتاج دقیق ردیابی کنید.

توجه داشته باشید که در این سوال، هم تابع `elapseTime` و هم اجرای کامل فیلتر ذرات را با ترکیب `elapseTime` و `observe` آزمایش خواهد شد.

علاوه بر استفاده از متد `elapseTime` در کلاس `ExactInference`، باید از کد زیر هم استفاده کنید:

```
newPosDist = self.getPositionDistribution(gameState, oldPos)
```

این خط کد با توجه به موقعیت قبلی (oldPos)، توزیع موقعیت های جدید روح را پیدا می‌کند. می‌توانید از روش نمونه برداری در کلاس `DiscreteDistribution` استفاده کنید.

برای تست این سوال می‌توان از دستور زیر استفاده کرد:

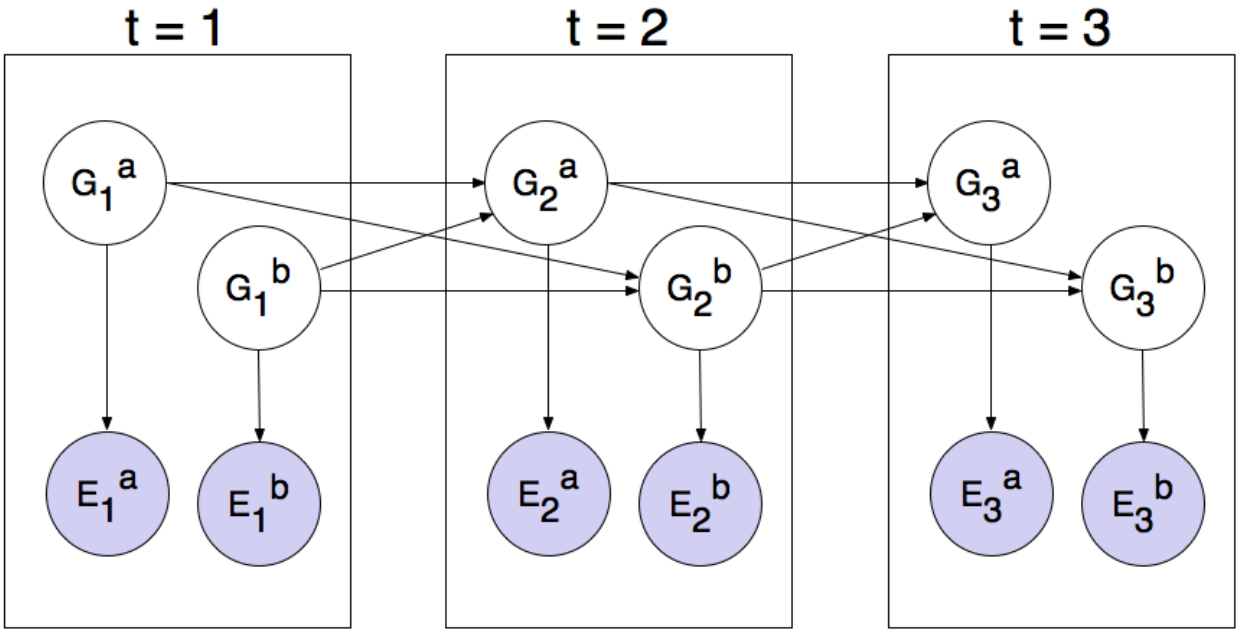
```
python autograder.py -q q7

python autograder.py -q q7 --no-graphics
```

۸) مشاهدات مشترک فیلتر ذرات (یک امتیاز)

تاکنون، ما هر روح را به طور مستقل ردیابی کرده‌ایم که برای `RandomGhost` و یا `DirectionalGhost` به خوبی کار می‌کند. با این حال، `DispersingGhost` اقداماتی (action) را انتخاب می کند که از روح‌های دیگر جلوگیری می کند. از آنجایی که مدل‌های انتقال روح‌ها دیگر مستقل نیستند، همه روح‌ها باید به طور مشترک در یک شبکه پویای بیزین ردیابی شوند!

شبکه بیزین دارای ساختار زیر است که در آن متغیرهای پنهان G موقعیت ارواح را نشان می‌دهند و متغیرهای انتشار E، فواصل هر روح (noisy distance) را نشان می‌دهند. این ساختار را می توان به ارواح بیشتری تعمیم داد، اما تنها برای دو روح در زیر نشان داده شده است.



اکنون یک فیلتر ذرات را پیاده‌سازی خواهید کرد که چندین روح را به طور همزمان ردیابی می‌کند. هر ذره یک tuple از موقعیت روح‌ها را نشان می دهد که نمونه ای از محلی است که همه روح‌ها در لحظه در آن هستند.

متد `initializeUniformly` را در `JointParticleFilter` در `inference.py` کامل کنید. مقداردهی اولیه شما باید باید به صورت unform prior باشد. برای ضرب کارتزین می‌توانید از `itertools.product` در ماژول `itertools` پایتون استفاده کنید. با این حال، توجه داشته باشید که در صورت استفاده از این، جایگشت ها به ترتیب تصادفی برگردانده نمی‌شوند. بنابراین، باید فهرست جایگشت‌ها را suffle کنید تا از قرارگیری یکنواخت ذرات در سراسر صفحه اطمینان حاصل شود.

مانند قبل، از `self.legalPositions` برای به دست آوردن لیستی از موقعیت هایی که یک روح ممکن است اشغال کند، استفاده کنید. همچنین مانند قبل، باید ذرات خود را در یک لیست ذخیره کنید.

برای تست این سوال می‌توان از دستور زیر استفاده کرد:

```
python autograder.py -q q8

python autograder.py -q q8 --no-graphics
```

۹) مشاهدات مشترک فیلتر ذرات (بخش دوم) (سه امتیاز)

در این سوال، متد `observeUpdate` را در کلاس `JointParticleFilter` در `inference.py` تکمیل خواهید کرد. این متد کل لیست ذرات را بر اساس مشاهده تمام فواصل روح ها وزن کرده و مجددا نمونه برداری می‌کند.

برای حلقه زدن روی همه روح‌ها، از کد زیر استفاده کنید:

```
for i in range(self.numGhosts):
    ...
```

همچنان می توانید موقعیت پکمن را با استفاده از `gameState.getPacmanPosition` بدست آورید، اما برای بدست آوردن موقعیت زندان برای یک روح، از `self.getJailPosition(i)` استفاده کنید، زیرا اکنون چندین روح وجود دارد که هر کدام موقعیت های زندان خود را دارند.

پیاده سازی شما همچنین باید دوباره حالت خاصی که همه ذرات وزن صفر دریافت می کنند را رسیدگی کند. در این حالت، `self.particles` باید از توزیع قبلی با فراخوانی `initializeUniformly` دوباره ایجاد شوند.

در بروز رسانی در کلاس `ParticleFilter`، باید دوباره از تابع `self.getObservationProb` برای یافتن احتمال مشاهده با توجه به موقعیت پکمن، یک موقعیت بالقوه روح و موقعیت زندان استفاده کنید. می‌توانید از روش نمونه برداری در کلاس `DiscreteDistribution` نیز استفاده کنید.

برای تست این سوال می‌توان از دستور زیر استفاده کرد:

```
python autograder.py -q q9

python autograder.py -q q9 --no-graphics
```

۱۰) زمان سپری شده فیلتر ذرات مشترک و تست کامل (سه امتیاز)

متد `elapseTime` را در `JointParticleFilter` در `inference.py` تکمیل کنید تا هر ذره را به درستی برای شبکه بیزین نمونه برداری کنید. به طور خاص، هر روح باید یک موقعیت جدید مشروط به موقعیت همه روح‌ها در مرحله زمانی قبلی داشته باشد.

برای حلقه زدن روی همه روح‌ها، از کد زیر استفاده کنید:

```
for i in range(self.numGhosts):
    ...
```

سپس، با فرض اینکه `i` به شاخص روح اشاره دارد، برای به دست آوردن توزیع بر روی موقعیت های جدید آن روح، از لیست (`prevGhostPositions`) موقعیت های قبلی همه روح‌ها، استفاده کنید:

```
newPosDist = self.getPositionDistribution(gameState, prevGhostPositions, i,
self.ghostAgents[i])
```

توجه داشته باشید که تکمیل و نمره این سؤال وابسته به تکمیل سؤال ۹ است.

در حین اجرای autograder توجه داشته باشید که `q10/1-JointParticlePredict` و `q10/2-JointParticlePredict` فقط اجرای `predict` شما را آزمایش می‌کنند و `q10/3-JointParticleFull` هم اجرای `predict` و هم اجرای `update` شما را آزمایش می کند.

سوال: به تفاوت بین تست ۱ و تست ۳ توجه کنید. در هر دو تست، پکمن می‌داند که روح‌ها به طرفین صفحه بازی حرکت می‌کنند. تفاوت بین تست‌ها چیست و چرا؟

برای تست این سوال می‌توان از دستور زیر استفاده کرد:

```
python autograder.py -q q10
python autograder.py -q q10 --no-graphics
```

توضیحات تکمیلی

- نسخه اصلی پروژه را می‌توانید اینجا مشاهده کنید. تنها پیاده‌سازی مواردی که در متن پروژه‌ی در اختیار شما قرار گرفته نیاز است.
- پاسخ به تمرین ها 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 **inference.py** و **bustersAgent.py** به همراه پاسخ خود به سوالات که در فایل به شکل سوال مشخص شده‌اند را در قالب یک فایل فشرده با فرمت AI_P4_[Student_Number].zip در سامانه کورسز آپلود کنید.
- در صورت هرگونه سوال یا ابهام از طریق ایمیل ai.aut.spring1401@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:
  - @Aminhbll
  - @amirrni
- برای این پروژه به صورت رندوم از تعدادی از دانشجویان تحویل آنلاین گرفته خواهد شد و نمره‌دهی مابقی دانشجویان بر اساس گزارش پروژه و پیاده‌سازی انجام شده است. تسلط کافی به سورس کد برنامه ضروری است.
- ددلاین این پروژه **۱۰ تیر ۱۴۰۱ ساعت ۲۳:۵۵** است و امکان ارسال با تاخیر وجود ندارد، بنابراین بهتر است انجام پروژه را به روز های پایانی موکول نکنید.