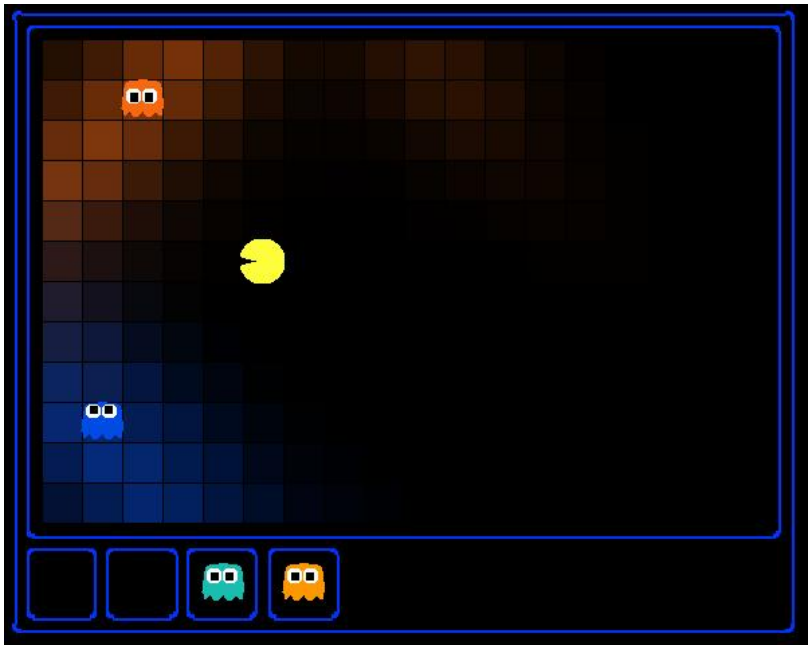
信息与计算机工程学院

人工智能导论

课程项目 4：捉鬼记



1、介绍

吃豆人一生都在逃避鬼魂，但事情并不总是如此。传说，许多年前，吃豆人的曾祖父学会了捉鬼。然而，他被自己的力量蒙蔽了双眼，只能通过鬼魂的撞击声和叮当声来追踪鬼魂。

在本项目中，你将设计使用传感器定位来捕捉隐身幽灵的吃豆人。你将从定位单个静止的幽灵发展到以无情的效率狩猎多个移动幽灵。

与项目 1 一样，该项目包括一个自动评分器，供你在机器上对答案进行评分。可以使用以下命令对所有问题运行此操作：

```
python autograder.py
```

它可以针对一个特定问题运行，例如 q2，方法是：

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

它可以通过以下形式的命令为一个特定的测试运行：

```
python autograder.py -t test_cases/q1/1-ObsProb
```

有关使用自动批改程序的详细信息，请参阅 Project 0 中的自动批改程序教程。

此项目的代码包含以下文件，这些文件以 zip 文档形式提供。

你需要编辑的文件	
bustersAgents.py	扮演捉鬼敢死队的吃豆人。
inference.py	使用幽灵的声音来跟踪鬼魂的代码。
factorOperations.py	用于计算新的联合或边缘化概率表的操作。
你需要查看的文件	
bayesNet.py	贝叶斯网和因子类。

其它的文件你可以忽视。

需要编辑和提交的文件：你需要完成 bustersAgents.py, inference.py, and factorOperations.py，将你修改的文件和自动批改程序（submission_autograder）产生的结果，以及项目报告一同提交。请不要修改或提交其它文件。

项目评估：你的代码会通过自动批改来判断其正确性，因此请不要修改代码中其它任何函数或者类，否则你会让自动批改程序无法正常运行。然而，你的解题思路和方法是你最终成绩的决定因素。必要的话，我们会查看你的代码来保证你得到应得的成绩。

学术造假：我们会查看你的代码和其它学生提交的代码是否雷同。禁止抄袭了他人代码，或只做简单修改后提交，一旦发现，成绩立马作废，而且会影响到你能否通过此课程。

寻求帮助：当你感到自己遇到了困难，请向你的同学和老师寻求帮助。小组合作、答疑时间、课堂讨论，这些都是用来帮助你的，请积极利用这些资源。设计这些项目的目的是让你更有效地理解和掌握课堂知识，学会如何将理论知识应用于实践，解决实际问题，而不是为考核而考核，或者有意刁难你，所以请尽你所能，完成它们。遇到困难时，向课代表和老师提问。

2、捉鬼队和贝叶斯网

捉鬼队

捉鬼队的目标是追捕害怕但看不见的幽灵。吃豆人足智多谋，配备了声纳（耳朵），可以为每个幽灵提供曼哈顿距离的嘈杂读数。当吃豆人吃掉所有的鬼魂时，游戏结束。首先，尝试使用键盘自己玩游戏。

```
python busters.py
```

基于提供给吃豆子的嘈杂距离读数，颜色块指示每个幽灵可能在哪里。显示屏底部的噪声距离始终为非负，并且始终在真实距离的 7 以内。距离读数的概率随着其与真实距离的差异呈指数下降。

你在此项目中的主要任务是实现推理以跟踪鬼魂。对于上面基于键盘的游戏，默认情况下为你实现了一种粗略的推理形式：所有可能存在鬼魂的方块都被鬼魂的颜色所遮蔽。当然，我们希望更好地估计幽灵的位置。幸运的是，贝叶斯网络为我们提供了强大的工具来充分利用我们拥有的信息。在本项目的其余部分，你将实现使用贝叶斯网络执行精确和近似推理的算法。该项目具有挑战性，因此我们鼓励你尽早开始。

在使用自动评分器监视和调试代码时，了解自动评分器正在执行的操作会很有帮助。此项目中有两种类型的测试，通过 test_cases 文件夹的子目录中找到的 .test 文件来区分。对于 DoubleInferenceAgentTest 一类的测试，你将看到代码生成的推理分布的可视化，但所有 Pacman 操作都将根据教员实现的操作进行预先选择。这是必要的，以便将你的分布与员工的分布进行比较。第二种类型的测试是 GameScoreTest，其中你的 BustersAgent 实际上会为吃豆人选择动作，你将观看你的吃豆人玩并赢得比赛。

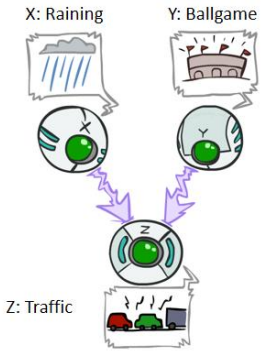
对于此项目，如果使用图形运行测试，有时自动评分器可能会超时。若要准确确定代码是否足够高效，应使用 --no-graphics 运行测试。如果自动评分器在无图形下通过，则即使自动评分器因图形超时，你也将获得满分。

贝叶斯网络和因子

首先，检查 bayesNet.py 来看你将使用的类——BayesNet 和 Factor。你可以运行此文件以查看一个 BayesNet 的例子和相关因素：python bayesNet.py。

你应该查看 printStarterBayesNet 功能——有一些有用的评论可以使以后的编码更轻松。

在此函数中创建的贝叶斯网络（下雨 ---->交通<---- 球赛）如下所示：



我们用到的术语如下：

贝叶斯网络：这是概率模型的有向无环图和一组条件概率表的表示，每个变量一个，如课堂中所示。上面的交通贝叶斯网就是一个例子。

因子：这将存储概率表，尽管表中条目的总和不一定为 1。因子的一般形式为

$$f(X_1,\ldots,X_m,y_1,\ldots y_n|Z_1,\ldots,Z_{p,},w_1,\ldots,w_q)$$

回想一下，小写的变量是已经分配了值的。对于每个可能分配给X_i和Z_j的值，因子会存储单个数字。在这里，Z_j和w_k 变量被称为条件的，而X_i和y_l变量是无条件的。

条件概率表（CPT）：这是一个满足两个属性的因子：

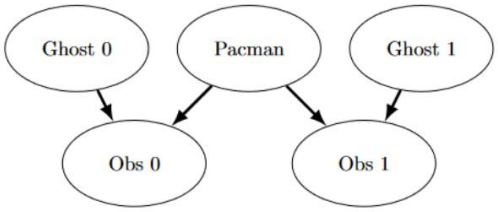
- 对于条件变量的每个赋值，其条目的总和必须为 1。
- 只有一个无条件变量。交通贝叶斯网络存储以下 CPT：P(Raining), P(Ballgame), 和 P(Traffic|Raining, Ballgame)

3、项目内容

问题 1（2 分）：贝叶斯网结构

在 inference.py 中实现 constructBayesNet 函数。它使用下面描述的结构构造一个空的贝叶斯网。贝叶斯网络在没有实际概率的情况下是不完整的，但因子是由教员代码定义和分配的；你不需要担心它。如果你好奇，你可以看看它在 bayesNet.py 的 printStarterBayesNet 中是如何工作的。阅读此函数也有助于解决此问题。

简化的捉鬼世界是根据以下贝叶斯网生成的：



如果这看起来很复杂，请不要担心！我们将逐步进行。如 constructBayesNet 的代码中所述，我们通过列出所有变量、它们的值以及它们之间的边来构建空结构。此图显示了变量和边，但它们的域呢？

- 根据关系图添加变量和边。
- 吃豆人和两个幽灵可以在网格中的任何位置（我们为此忽略墙壁）。为这些添加所有可能的位置元组。
- 这里的观测结果是非负的，等于吃豆人到鬼魂的曼哈顿距离±噪音。

评分：若要测试和调试代码，请运行

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

问题 2（3 分）：联合因子

在 factorOperations.py 中实现 joinFactors 函数。它接受 Factor 列表并返回一个新的 Factor，其概率条目应该是输入 Factor 的相应行的乘积。换句话说，你需要实现课堂上教授的从边缘分布和条件分布计算出联合分布的过程。

joinFactors 可以用作乘积规则，例如，如果我们有一个形式为$P(X|Y)$的因子，形式为$P(Y)$的另一个因子，然后联合这些因子将产生$P(X,Y)$。因此，joinFactors 允许我们合并条件变量的概率（在本例中，Y）但是，你不应该假设在概率表上调用 joinFactors，可以在行总和不等于 1 的因子上调用 joinFactors。

评分：若要测试和调试代码，请运行

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

在调试期间运行特定测试可能很有用，以便仅查看打印出的一组因素。例如，若要仅运行第一个测试，请运行：

```
python autograder.py -t test_cases/q2/1-product-rule
```

提示和观察：

- 你的 joinFactor 应返回一个新的 Factor。
- 以下是 joinFactors 可以执行的操作的一些示例：
  - joinFactors($P(X|Y),P(Y)$)=$P(X,Y)$
  - joinFactors($P(V,W|X,Y,Z),P(X,Y|Z)$)=$P(V,W,X,Y|Z)$
  - joinFactors($P(X|Y,Z),P(Y)$)=$P(X,Y|Z)$
  - joinFactors($P(V|W),P(X|Y),P(Z)$)=$P(V,X,Z|W,Y)$
- 对于常规 joinFactors 操作，返回 Factor 中的哪些变量是无条件的？哪些变量是有条件的？
- Factor 存储一个 variableDomainsDict，它将每个变量映射到它可以采用的值列表（其域）。一个 Factor 从实例化的 BayesNet 获取其 variableDomainsDict。因此，它包含贝叶斯网的所有变量，而不仅仅是 Factor 中使用的无条件和条件变量。对于此问题，你可以假设所有输入 Factor 都来自同一个 BayesNet，因此它们的 vavriableDomainsDict 都不变。

问题 3（2 分）：消除（还不是鬼）

在 factorOperations.py 中实现 eliminate 函数。它需要一个 Factor 和一个变量来消除，并返回一个不包含该变量的新因子。这对应于对因子中的所有条目求和，这些条目仅在要消除的变量的值上有所不同。

评分：若要测试和调试代码，请运行

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

在调试期间运行特定测试可能很有用，以便仅查看打印出的一组因素。例如，若要仅运行第一个测试，请运行：

```
python autograder.py -t test_cases/q3/1-simple-eliminate
```

提示和观察：

- 你的 eliminate 应返回一个新的 Factor。
- 消除可用于边缘化概率表中的变量。例如：

- eliminate($P(X,Y|Z),Y$)=$P(X|Z)$
- eliminate($P(X,Y|Z),X$)=$P(Y|Z)$

- 对于一般 eliminate 操作，返回因子中的哪些变量是无条件的？哪些变量是有条件的？
- 请记住，Factor 存储原始 BayesNet 的 variableDomainsDict，而不仅仅是它们使用的无条件和条件变量。因此，返回的 Factor 应具有与输入 Factor 相同的 variableDomainsDict。

问题 4（2 分）：变量消除

在 inference.py 中实现 inferenceByVariableElimination 函数。它回答概率查询，该查询使用 BayesNet、查询变量列表和证据表示。

评分：若要测试和调试代码，请运行

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

在调试期间运行特定测试可能很有用，以便仅查看打印出的一组因素。例如，若要仅运行第一个测试，请运行：

```
python autograder.py -t test_cases/q4/1-disconnected-eliminate
```

提示和观察：

- 该算法应按消除顺序循环操作隐藏变量，执行联接和消除该变量，直到只剩下查询和证据变量。
- 输出因子中的概率总和应为 1（因此它是以证据为条件的真实条件概率）。
- 查看 inference.py 中的 inferenceByEnumeration 函数，以获取有关如何使用所需函数的示例。（提醒：枚举推理首先联接所有变量，然后消除所有隐藏变量。相反，变量消除通过遍历所有隐藏变量来交错连接和消除，并在移动到下一个隐藏变量之前对单个隐藏变量执行连接和消除。
- 你需要处理特殊情况，即你加入的因子只有一个无条件变量（注释文档更详细地指定要做什么），你可以直接忽视该因子，因为消除该变量的结果应该为 1。

问题 5a（0 分）：DiscreteDistribution 类

不幸的是，有时间步长会使我们的图增长太多，以至于变量消除不可行。相反，我们将使用 HMM 的前向算法进行精确推理，并使用粒子过滤进行更快但近似的推理。

对于项目的其余部分，我们将使用 inference.py 中定义的离散分布类来建模信念分布和权重分布。此类是内置 Python 字典类的扩展，其中键是我们分布的不同离散元素，相应的值与分布分配给该元素的置信度或权重成正比。这个问题要求你填写这个类的缺失部分，这对后面的问题至关重要（即使这个问题本身不值一分）。

首先，填写 normalize 方法，该方法将分布中的值归一化为一，但保持值的比例相同。使用 total 方法查找分布中值的总和。对于空分布或所有值均为零的分布，不执行任何操作。请注意，此方法直接修改分布，而不是返回新的分布。

其次，填写 sample 方法，该方法从分布中提取样本，其中对键进行采样的概率与其相应的值成正比。假设分布不为空，并且并非所有值都为零。请注意，在调用此方法之前，不一定必须对分布进行规范化。你可能会发现 Python 内置的 random.random（） 函数对这个问题很有用。

此问题没有自动评分器测试，但可以轻松检查实现的正确性。我们提供了 Python 文档测试作为起点，你可以随意添加更多并实现自己的其他测试。你可以使用以下命令运行文档测试：

```
python -m doctest -v inference.py
```

请注意，根据示例方法的实现细节，某些正确的实现可能无法通过提供的 doctest。要彻底检查样本方法的正确性，你应该抽取许多样本，看看每个键的频率是否收敛到与其相应值成正比。

问题 5b（1 分）：观测概率

在本期中，你将在 inference.py 的推理模块基类中实现 getObservationProb 方法。此方法接受观察（这是对到幽灵的距离的嘈杂读数）、吃豆人的位置、幽灵的位置和幽灵监狱的位置，并返回给定吃豆人的位置和幽灵的位置的噪声距离读数的概率。换句话说，我们想返回

$$P(noisyDistance|pacmanPosition, ghostPosition)$$

距离传感器在给定吃豆人到幽灵的真实距离的情况下，在距离读数上具有概率分布。此分布由函数 busters.getObservationProbability（noisyDistance， trueDistance）建模，该函数返回 $P(noisyDistance|trueDistance)$并已经提供给你了。你应该使用这个函数来帮助你解决问题，并使用提供的曼哈顿距离函数来查找吃豆人的位置和幽灵的位置之间的距离。

但是，我们还必须处理监狱的特殊情况。具体来说，当我们捕获一个鬼魂并将其发送到监狱位置时，我们的距离传感器会确定地返回 None，并且不会返回其他任何内容（观察 = None 当且仅当鬼魂在监狱中时）。这样做的一个结果是，如果鬼魂的位置是监狱位置，那么观测值是概率为 1 的 None，其他所有结果的概率为 0。确保在实现中处理此特殊情况;我们实际上有一套不同的规则，用于何时幽灵入狱，以及何时观察为 None。

要测试代码并针对此问题运行自动评分器，请执行以下操作：

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

问题 6（2 分）：精确的推理观察

在本问题中，你将在 inference.py 的精确推理类中实现 observeUpdate 方法，以正确更新代理在给定 Pacman 传感器观察结果的情况下对幽灵位置的置信分布。你正在实施在线信念更新以观察新证据。对于此问题，观察更新方法应在收到传感器读数后更新地图上每个位置的置信度。你应该在变量 self.allPositions 上迭代更新，该变量包括所有合法职位以及特殊监狱职位。信念表示幽灵位于特定位置的概率，并作为离散分布对象存储在名为 self.faith 的字段中，你应该更新该字段。

在键入任何代码之前，请写下你尝试解决的推理问题的方程式。你应该使用你在上一个问题中写的函数 self.getObservationProb，它返回给定吃豆人的位置、潜在的幽灵位置和监狱位置的观察概率。你可以使用 gameState.getPacmanPosition（） 获取 Pacman 的位置，并使用 self.getJailPosition（） 获取监狱位置。

在吃豆人显示中，高后验信念由明亮的颜色表示，而低信念由暗淡的颜色表示。你应该从一大片信念云开始，随着更多证据的积累，这种云会随着时间的推移而缩小。在观看测试用例时，请确保你了解正方形如何收敛到其最终颜色。

注意：你的破坏者代理对于他们正在跟踪的每个幽灵都有一个单独的推理模块。这就是为什么如果你在 observUpdate 函数中打印一个观察结果，即使板上可能有多个幽灵，你也只会看到一个数字。

要针对此问题运行自动评分器并可视化输出，请执行以下操作：

```
python autograder.py -q q6
```

如果要在没有图形的情况下运行此测试（或任何其他测试），可以添加以下标志：

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

问题 7（2 分）：随时间推移的精确推理观察

在上一个问题中，你根据吃豆人的观察为他实现了信念更新。幸运的是，吃豆人的观察并不是他关于鬼魂可能在哪里的唯一知识来源。吃豆人还了解幽灵可能移动的方式;也就是说，幽灵不能在一个时间步长内穿过一堵墙或多个空间。

要理解为什么这对吃豆人有用，请考虑以下场景，其中有吃豆人和一个幽灵。吃豆人收到许多观察结果，表明幽灵非常接近，但随后有一个表明幽灵非常远。显示幽灵很远的读数很可能是错误传感器的结果。吃豆人对幽灵如何移动的先验知识将减少此读数的影响，因为吃豆人知道幽灵不能只移动这么远。

在本问题中，你将在精确推理中实现 elapseTime 方法。对于此问题，经过时间步骤应在经过一个时间步长后更新地图上每个位置的置信度。你的代理可以通过 self.getPositionDistribution 访问幽灵的操作分发。为了获得重影新位置的分布，给定其先前的位置，请使用以下代码行：

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

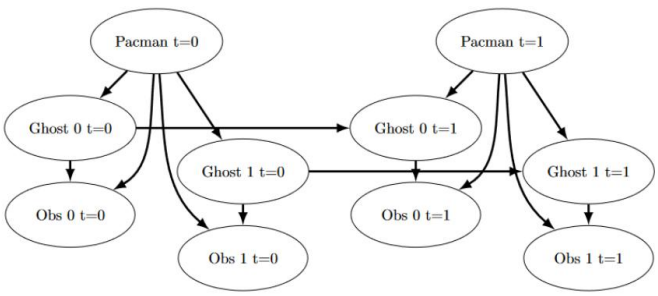
其中 oldPos 指的是之前的幽灵位置。newPosDist 是一个离散分布对象，其中对于 self.allPositions 中的每个位置 p，newPosDist[p] 是幽灵在时间 t + 1 处位置 p 的概率，假设幽灵在时间 t 处位于位置 oldPos。请注意，此调用可能相当昂贵，因此如果你的代码超时，需要考虑的一件事是是否可以减少对 self.getPositionDistribution 的调用次数。

在键入任何代码之前，请写下你尝试解决的推理问题的方程式。为了将预测实现与上一个问题中的更新实现分开测试，此问题不会使用更新实现。

由于吃豆人没有观察幽灵的行为，这些动作不会影响吃豆人的信念。随着时间的推移，吃豆人的信念将反映棋盘上他认为鬼魂最有可能被赋予棋盘几何形状和鬼魂可能的法律行动的地方，吃豆人已经知道了。

对于这个问题中的测试，我们有时会使用随机移动的幽灵，有时我们将使用 GoSouthGhost。随着时间的推移，这个幽灵往往会向南移动，并且在没有任何观察的情况下，吃豆人的信念分布应该开始集中在棋盘底部。要查看每个测试用例使用哪个幽灵，你可以查看 .test 文件。

下图显示了贝叶斯网/隐马尔可夫模型。尽管如此，你还是应该依靠上面的描述来实现，因为有些部分是为你实现的（即 getPositionDistribution 被抽象为：$P(G_{t+1}|gameState, G_t)$）。



要针对此问题运行自动评分器并可视化输出，请执行以下操作：

```
python autograder.py -q q7
```

如果要在没有图形的情况下运行此测试（或任何其他测试），可以添加以下标志：

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

当你观看自动评分器输出时，请记住，较浅的方块表示吃豆人认为幽灵更有可能占据该位置，而较暗的方块表示幽灵不太可能占据该位置。对于哪些测试用例，你注意到正方形阴影中出现了差异？你能解释一下为什么有些方块变亮而有些方块变暗吗？

问题 8（1 分）：精确推理的完整测试

现在吃豆人知道如何利用他的先验知识和观察来弄清楚鬼魂在哪里，他准备自己追捕鬼魂。我们将使用你的 observUpdate 和 elapseTime 实现来保持更新的信念分布，你的简单贪婪代理将根据每个时间步的最新特征选择一个操作。在简单的贪婪策略中，吃豆人假设每个鬼魂都根据他的信仰处于最有可能的位置，然后向最近的鬼魂移动。到目前为止，吃豆人已经通过随机选择一个有效的动作来移动。

在 bustersAgents.py 年的 GreedyBustersAgent 中实现 chooseAction 方法。你的代理应首先找到每个剩余未捕获的幽灵的最可能位置，然后选择一个操作，以最小化迷宫与最近幽灵的距离。

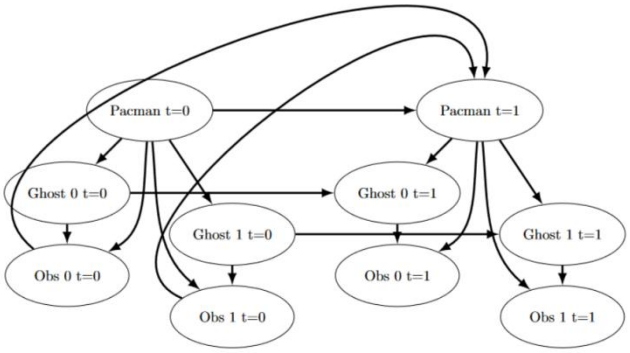
要查找任意两个位置 pos1 和 pos2 之间的迷宫距离，请使用 self.distancer.getDistance（pos1，pos2）。要在操作后查找后续位置：

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

你将获得 livingGhostPositionDistributions，这是一个离散分布对象列表，表示每个尚未捕获的幽灵的位置信念分布。

如果正确实施，你的代理应在 q8/3-gameScoreTest 中赢得游戏，分数大于 700 至少 8 次（满分 10 次）。注意：自动评分器也会直接检查你的推断的正确性，但游戏的结果是合理的健全性检查。

我们可以通过对前图的修改来表示贪婪代理的工作原理：



要针对此问题运行自动评分器并可视化输出，请执行以下操作：

```
python autograder.py -q q8
```

如果要在没有图形的情况下运行此测试（或任何其他测试），可以添加以下标志：

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

问题 9（1 分）：近似推理初始化和信念

近似推断是本季幽灵猎人非常流行的。对于接下来的几个问题，你将实现一种粒子过滤算法来跟踪单个鬼影。

首先，在 inference.py 的 ParticleFilter 类中实现函数 initializeUniform 和 getBeliefDistribution。粒子（样本）是这个推理问题中的幽灵位置。请注意，对于初始化，粒子应均匀（而不是随机）分布在合法位置上，以确保均匀的先验。我们建议考虑一下 mod 运算符对 initializeUniform 有何用处。

请注意，存储粒子的变量必须是列表。列表只是未加权变量（在本例中为仓位）的集合。将粒子存储为任何其他数据类型（如字典）是不正确的，并且会产生错误。然后，getBeliefDistribution 方法获取粒子列表并将其转换为离散分布对象。

要针对此问题运行自动评分器并可视化输出，请执行以下操作：

```
python autograder.py -q q9
```

问题 10（2 分）：近似推理观察

接下来，我们将在 inference.py 中的 ParticleFilter 类中实现 observeUpdate 方法。这种方法在 self.particles 上构建了一个权重分布，其中粒子的权重是给定吃豆人的位置和粒子位置的观测概率。然后，我们从这个加权分布中重新采样以构建我们的新粒子列表。

你应该再次使用函数 self.getObservationProb 来查找给定吃豆人位置，潜在幽灵位置和监狱位置的观察概率。DiscreteDistribution 类的示例方法也将很有用。提醒一下，你可以使用 gameState.getPacmanPosition（）获取 Pacman 的位置，并使用 self.getJailPosition（）获取监狱位置。

正确的实现必须处理一种特殊情况。当所有粒子的权重为零时，应通过调用 initializeUniform 重新初始化粒子列表。离散分布的总方法可能很有用。

要针对此问题运行自动评分器并可视化输出，请执行以下操作：

```
python autograder.py -q q10
```

如果要在没有图形的情况下运行此测试（或任何其他测试），可以添加以下标志：

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

问题 11（2 分）：随时间推移的近似推理

在 inference.py 中的 ParticleFilter 类中实现 elapseTime 函数。此函数应构造一个新的粒子列表，该列表对应于 self.particle 中前进一个时间步长的每个现有粒子，然后将这个新列表分配回 self.particles。完成后，你应该能够像精确推理一样有效地跟踪鬼魂。

请注意，在本期中，我们将单独测试 elapseTime 函数，以及结合 elapseTime 和 observe 的粒子过滤器的完整实现。

与 ExactInference 类的 elapseTime 方法一样，你应该使用：

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

这行代码获取了幽灵新位置的分布，给定其先前的位置（oldPos）。DiscreteDistribution 类的示例方法也将很有用。

要针对此问题运行自动评分器并可视化输出，请执行以下操作：

```
python autograder.py -q q11
```

如果要在没有图形的情况下运行此测试（或任何其他测试），可以添加以下标志：

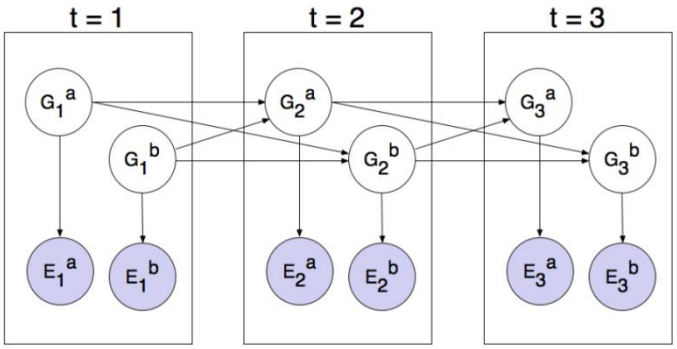
```
python autograder.py -q q11 --no-graphics
```

请注意，即使没有图形，此测试也可能需要几分钟才能运行。

问题 12（1 分）：联合粒子过滤器初始化

到目前为止，我们已经独立跟踪了每个幽灵，这对于默认的 RandomGhost 或更高级的 DirectionalGhost 来说效果很好。然而，珍贵的分散幽灵会选择避开其他幽灵的行动。由于幽灵的过渡模型不再是独立的，因此必须在动态贝叶斯网中共同跟踪所有幽灵！

贝叶斯网络具有以下结构，其中隐藏变量 *G* 表示鬼影位置和发射变量 *E* 是到每个鬼魂的嘈杂距离。这种结构可以扩展到更多的鬼魂，但只有两个（*a* 和 *b*） 如下所示。



现在，你将实现一个同时跟踪多个幽灵的粒子过滤器。每个粒子将代表一个幽灵位置的元组，这是当前所有幽灵所在位置的样本。该代码已经设置为从你将创建的联合推理算法中提取有关每个鬼魂的边际分布，以便可以显示有关单个鬼魂的信念云。

在 inference.py 中完成 JointParticleFilter 中的 initializeUniform 方法。初始化应与统一的先验一致。你可能会发现 Python 迭代工具包很有帮助。具体来说，查看 itertools.product 以获得笛卡尔产品的实现。但是，请注意，如果使用它，则不会以随机顺序返回排列。因此，你必须打乱排列列表，以确保粒子在电路板上均匀放置。洗牌后，你应该使用 mod 运算符索引到排列列表中，以将粒子添加到 self.particles。

为了澄清起见，我们不需要在 q9 的 initializeUniform 中进行洗牌，因为 q12 与 q9 不同，有一些 self.numParticle 小于排列总数的测试，因此我们需要打乱列表以确保排列列表中的第一个 self.numParticle 项目均匀分布在整个板上。

和以前一样，使用 self.legalPositions 获取幽灵可能占据的位置列表。同样和以前一样，存储粒子的变量必须是一个列表。

要针对此问题运行自动评分器并可视化输出，请执行以下操作：

```
python autograder.py -q q12
```

如果要在没有图形的情况下运行此测试（或任何其他测试），可以添加以下标志：

```
python autograder.py -q q12 --no-graphics
```

重要说明：通常，如果使用图形运行测试，有时自动评分器可能会超时。若要准确确定代码是否足够高效，应使用 --no-graphics 标志运行测试。如果自动评分器通过此标志，则即使自动评分器因图形超时，你也将获得满分。

问题 13（2 分）：联合粒子过滤器观察

在本问题中，你将在 inference.py 的 JointParticleFilter 类中完成观察更新方法。正确的实现将根据对所有幽灵距离的观察对整个粒子列表进行加权和重新采样。

要遍历所有幽灵，请使用：

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

你仍然可以使用 gameState.getPacmanPosition（） 获得吃豆人的位置，但要获得幽灵的监狱位置，请使用 self.getJailPosition（i），因为现在有多个幽灵，每个幽灵都有自己的监狱位置。

你的实现还应该再次处理所有粒子都获得零权重的特殊情况。在这种情况下，self.particle 应该通过调用 initializeUniform 从先前的发行版重新创建。

与 ParticleFilter 类的更新方法一样，你应该再次使用函数 self.getObservationProb 来查找给定吃豆人位置、潜在幽灵位置和监狱位置的观察概率。DiscreteDistribution 类的示例方法也将很有用。

要针对此问题运行自动评分器并可视化输出，请执行以下操作：

```
python autograder.py -q q13
```

如果要在没有图形的情况下运行此测试（或任何其他测试），可以添加以下标志：

```
python autograder.py -q q13 --no-graphics
```

重要说明：通常，如果使用图形运行测试，有时自动评分器可能会超时。若要准确确定代码是否足够高效，应使用 --no-graphics 标志运行测试。如果自动评分器通过此标志，则即使自动评分器因图形超时，你也将获得满分。

问题 14（2 分）：全面测试随时间推移的联合粒子过滤器

在 inference.py 中完成 JointParticleFilter 中的 elapseTime 方法，以正确重新采样贝叶斯网的每个粒子。特别是，每个鬼魂都应该根据前一个时间步的所有鬼魂的位置来绘制一个新位置。

与上一个问题一样，你可以使用以下命令循环使用幽灵：

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

然后，假设我引用了幽灵的索引，要获得该单个幽灵的新仓位的分布，给定所有幽灵先前仓位的列表（prevGhostPosition），请使用：

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

请注意，完成此题涉及对问题 13 和问题 14 进行评分。由于这些问题涉及联合分布，它们需要更多的计算能力（和时间）来评分，所以请耐心等待！

运行自动评分器时，请注意，q14/1-JointParticlePredict 和 q14/2-JointParticlePredict 仅测试你的预测实现，而 q14/3-JointParticleFull 测试你的预测和更新实现。请注意测试 1 和测试 3 之间的

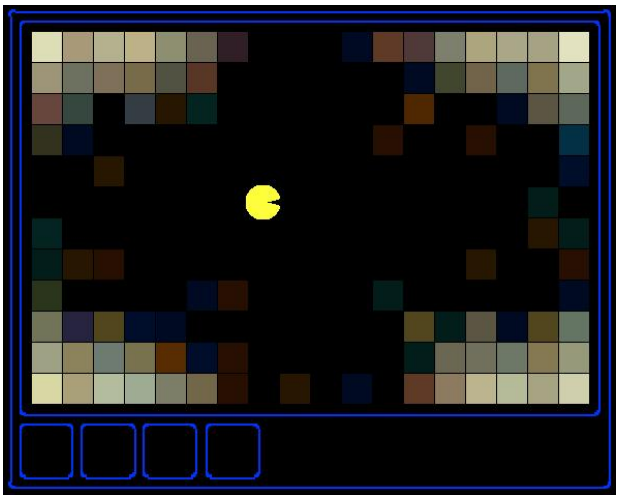
区别。在这两个测试中，吃豆人都知道幽灵会移动到游戏板的两侧。测试之间有什么不同，为什么？

要对实现进行评分，请运行自动评分器：

```
python autograder.py -q q6
```

如果要在没有图形的情况下运行此测试（或任何其他测试），可以添加以下标志：

```
python autograder.py -q Q6 --no-graphics
```



重要说明：通常，如果使用图形运行测试，有时自动评分器可能会超时。若要准确确定代码是否足够高效，应使用 --no-graphics 标志运行测试。如果自动评分器通过此标志，则即使自动评分器因图形超时，你也将获得满分。

4、项目报告

简要清晰地描述完成项目时遇到的困难，采用的解决方法，提出改进意见，和总结每个小组成员的贡献。

5、提交

在提交你的解答之前，你需要通过执行 submission_autograder.pyc 来产生几个文件。在运行这个程序之前，你必须确认所有与 autograde 有关的文件都处在原始状态，没有做过任何的改动。假如你编辑过任何 autograde 的文件，请重新下载一份项目代码，仅仅替换你作解答的文件，否则运行 submission_autograder.pyc 将无法通过。

此外，submission_autograder.pyc 要在 Python 3.6（准确的说是 3.6.13，你可以用 Anaconda 来安装正确的 Python 版本）下执行，否则会报错。

最后，submission_autograder.pyc 需要用 rsa 库来给你的成绩加密，假如你没有的话，请用下面的命令安装 rsa 库。

```
conda install -c conda-forge rsa
```

或者用下面的命令，假如你没有 conda。

```
pip install rsa
```

进到你的 reinforcement 文件夹里，执行以下命令：

```
python submission_autograder.pyc
```

成功执行后，该命令会输出你的各个题目的得分和最后总分，并在 grade 文件夹里会生成一个 log 文件和一个 token 文件。确认该分数和你自己运行 autograder.py 得到的分数相同后，将整个 grade 文件夹和你修改过的文件（其它没有修改过的，例如 autograder.py，不需要）以及项目报告，放在一个以你的组号命名的文件夹里，生成一个 zip 文件，一并提交上来。