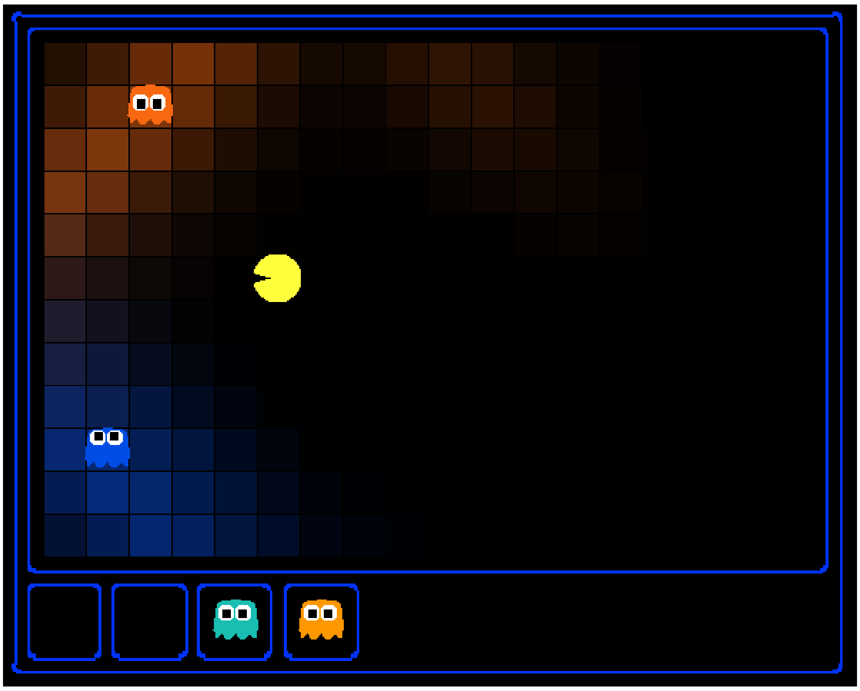
Project 4: Ghostbusters



I can hear you, ghost.
Running won't save you from my
Particle filter!

Introduction

Pac-Man spends his life running from ghosts, but things were not always so. Legend has it that many years ago, Pac-Man's great grandfather Grandpac learned to hunt ghosts for sport. However, he was blinded by his power and could only track ghosts by their banging and clanging.

In this project, you will design Pac-Man agents that use sensors to locate and eat invisible ghosts. You'll advance from locating single, stationary ghosts to hunting packs of multiple moving ghosts with ruthless efficiency.

The code for this project contains the following files, available as a zip archive.

Files you will edit

bustersAgents.py	Agents for playing the Ghostbusters variant of Pac-Man.
inference.py	Code for tracking ghosts over time using their sounds.

Files you will not edit

busters.py	The main entry to Ghostbusters (replacing pacman.py)
bustersGhostAgents.py	

	New ghost agents for Ghostbusters
`distanceCalculator.py`	Computes maze distances
`game.py`	Inner workings and helper classes for Pac-Man
`ghostAgents.py`	Agents to control ghosts
`graphicsDisplay.py`	Graphics for Pac-Man
`graphicsUtils.py`	Support for Pac-Man graphics
`keyboardAgents.py`	Keyboard interfaces to control Pac-Man
`layout.py`	Code for reading layout files and storing their contents
`util.py`	Utility functions

What to submit: You will fill in portions of `bustersAgents.py` and `inference.py` during the assignment. You should submit this file with your code and comments. Please *do not* change the other files in this distribution or submit any of our original files other than `inference.py` and `bustersAgents.py`. Directions for submitting are on the course website; this assignment is submitted with the command `submit p4`.

Evaluation: Your code will be autograded for technical correctness. Please *do not* change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation -- not the autograder's judgements -- will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; *please* don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours, section, and the newsgroup are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Ghostbusters and BNs

In the cs188 version of Ghostbusters, the goal is to hunt down scared but invisible ghosts. Pac-Man, ever resourceful, is equipped with sonar (ears) that provides noisy readings of the Manhattan distance to each ghost. The game ends when pacman has eaten all the ghosts. To start, try playing a game yourself using the keyboard.

```
python busters.py
```

The blocks of color indicate where the each ghost could possibly be, given the noisy distance readings provided to Pac-Man. The noisy distances at the bottom of the display are always non-negative, and always within 7 of the true distance. The probability of a distance reading decreases exponentially with its difference from the true distance.

Your primary task in this project is to implement inference to track the ghosts. A crude form of inference is implemented for you by default: all squares in which a ghost could possibly be are shaded by the color of the ghost. Option `-s` shows where the ghost actually is.

```
python busters.py -s -k 1
```

Naturally, we want a better estimate of the ghost's position. We will start by locating a single, stationary ghost using multiple noisy distance readings. The default `BustersKeyboardAgent` in [bustersAgents.py]() uses the `ExactInference` module in [inference.py]() to track ghosts.

Question 1 (3 points) Update the `observe` method in `ExactInference` class of [inference.py]() to correctly update the agent's belief distribution over ghost positions. When complete, you should be able to accurately locate a ghost by circling it.

```
python busters.py -s -k 1 -g StationaryGhost
```

Because the default `RandomGhost` ghost agents move independently of one another, you can track each one separately. The default `BustersKeyboardAgent` is set up to do this for you. Hence, you should be able to locate multiple stationary ghosts simultaneously. Encircling the ghosts should give you precise distributions over the ghosts' locations.

```
python busters.py -s -g StationaryGhost
```

Note: your busters agents have a separate inference module for each ghost they are tracking. That's why if you print an observation inside the `observe` function, you'll only see a single number even though there may be multiple ghosts on the board.

Hints:

- You are implementing the online belief update for observing new evidence. Before any readings, pacman believes the ghost could be anywhere: a uniform prior (see `initializeUniformly`. After receiving a reading, the `observe` function is called, which must update the belief at every position.
- Before typing any code, write down the equation of the inference problem you are trying to solve.
- Try printing `noisyDistance`, `emissionModel`, and `pacmanPosition` (in the `observe` function) to get started.
- In the Pac-Man display, high posterior beliefs are represented by bright colors, while low beliefs are represented by dim colors. You should start with a large cloud of belief that shrinks over time as more evidence accumulates.
- Beliefs are stored as `util.Counter` objects (like dictionaries) in a field called `self.beliefs`, which you should update.
- You should not need to store any evidence. The only thing you need to store in `ExactInference` is `self.beliefs`.

Ghosts don't hold still forever. Fortunately, your agent has access to the action distribution for any `GhostAgent`. Your next task is to use the ghost's move distribution to update your agent's beliefs when time elapses.

Question 2 (4 points) Fill in the `elapseTime` method in `ExactInference` to correctly update the agent's belief distribution over the ghost's position when the ghost moves. When complete, you should be able to accurately locate moving ghosts, but some uncertainty will always remain about a ghost's position as it moves.

```
python busters.py -s -k 1

python busters.py -s -k 1 -g DirectionalGhost
```

Hints:

- Instructions for obtaining a distribution over where a ghost will go next, given its current position and the `gameState`, appears in the comments of `ExactInference.elapseTime` in inference.py.
- A `DirectionalGhost` is easier to track because it is more predictable. After running away from one for a while, your agent should have a good idea where it is.
- We assume that ghosts still move independently of one another, so while you can develop all of your code for one ghost at a time, adding multiple ghosts should still work correctly.

Now that Pac-Man can track ghosts, try playing without peeking at the ghost locations. Beliefs about each ghost will be overlaid on the screen. The game should be challenging, but not impossible.

```
python busters.py -l bigHunt
```

Now, pacman is ready to hunt down ghosts on his own. You will implement a simple greedy hunting strategy, where Pac-Man assumes that each ghost is in its most likely position according to its beliefs, then moves toward the closest ghost.

Question 3 (4 points) Implement the `chooseAction` method in `GreedyBustersAgent` in bustersAgents.py. Your agent should first find the most likely position of each remaining (uncaptured) ghost, then choose an action that minimizes the distance to the closest ghost. If correctly implemented, your agent should win `smallHunt` with a score greater than 700 at least 8 out of 10 times.

```
python busters.py -p GreedyBustersAgent -l smallHur
```

Hints:

- When correctly implemented, your agent will thrash around a bit in order to capture a ghost.
- The comments of `chooseAction` provide you with useful method calls for computing maze distance and successor positions.
- Make sure to only consider the living ghosts, as described in the comments.

Approximate Inference

Approximate inference is very trendy among ghost hunters this season. Next, you will implement a particle filtering algorithm for tracking a single ghost.

Question 4 (5 points) Implement all necessary methods for the `ParticleFilter` class in inference.py. When complete, you should be able to track ghosts nearly as effectively as with exact inference. This means that your agent should win `oneHunt` with a score greater than 100 at least 8 out of 10 times.

```
python busters.py -k 1 -s -a inference=ParticleFilt
```

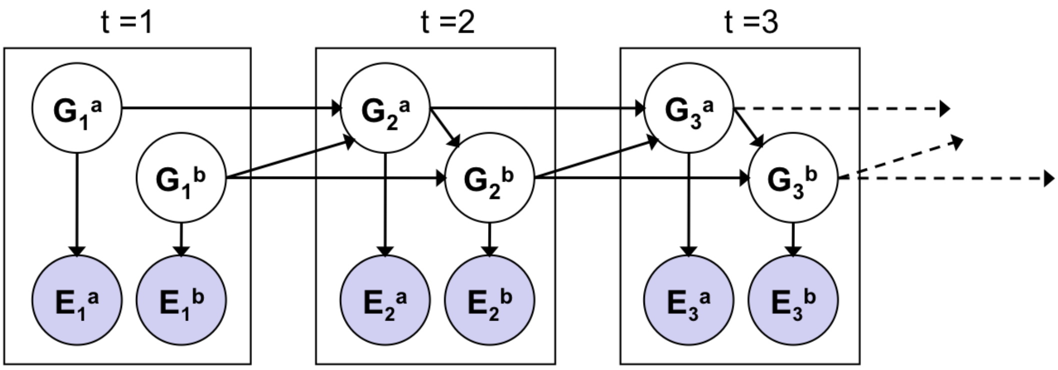
Hints:

- A particle (sample) is a ghost position in this inference problem.

- The belief cloud generated by a particle filter will look noisy compared to the one for exact inference.
- To debug, you may want to start with `-g StationaryGhost`.

So far, we have tracked each ghost independently, which works fine for the default `RandomGhost` or more advanced `DirectionalGhost`. However, the prized `DispersingGhost` chooses actions that avoid other ghosts. Since the ghosts' transition models are no longer independent, all ghosts must be tracked jointly in a dynamic Bayes net!

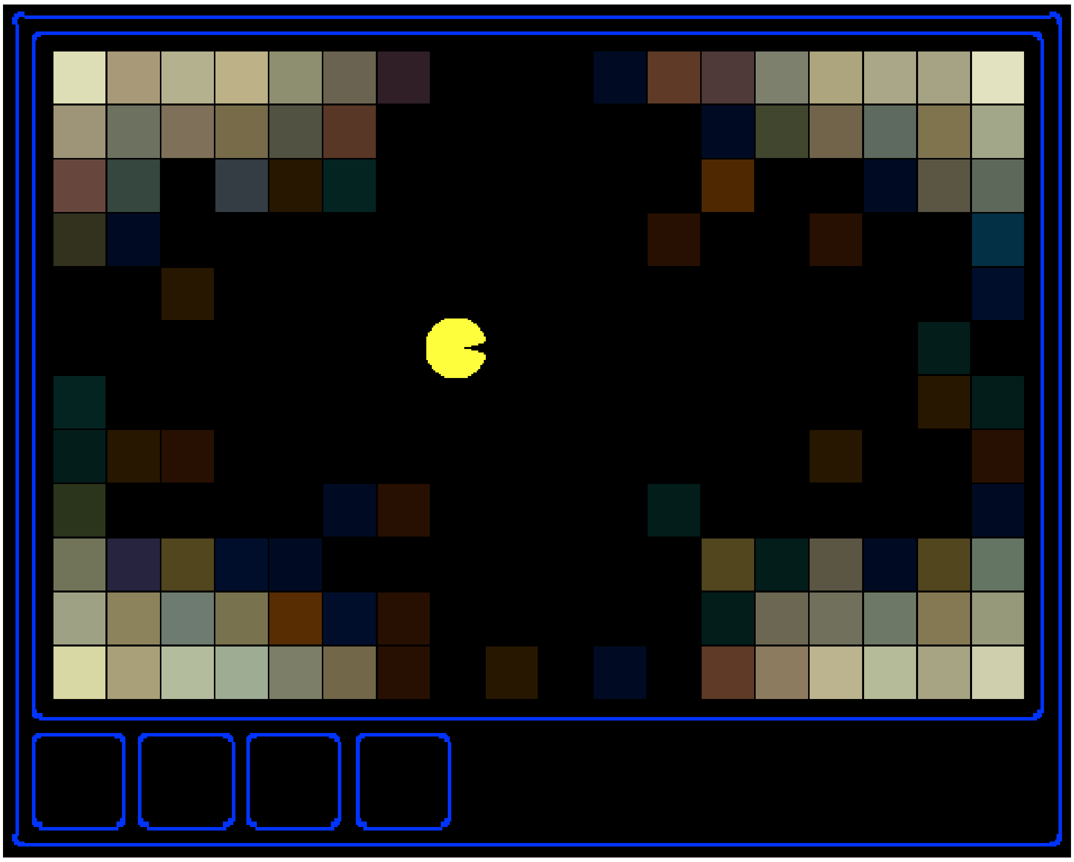
Since the ghosts move in sequence, the Bayes net has the following structure, where the hidden variables G represent ghost positions and the emission variables are the noisy distances to each ghost. This structure can be extended to more ghosts, but only two are shown below.



You will now implement a particle filter that tracks multiple ghosts simultaneously. Each particle will represent a tuple of ghost positions that is a sample of where all the ghosts are at the present time. The code is already set up to extract marginal distributions about each ghost from the joint inference algorithm you will create, so that belief clouds about individual ghosts can be displayed.

Question 5 (3 points) Complete the `elapseTime` method in `JointParticleFilter` in [inference.py]() to resample each particle correctly for the Bayes net. The comments in the method provide instructions for helpful support functions. With only this part of the particle filter completed, you should be able to predict that ghosts will flee to the perimeter of the layout to avoid each other, though you won't know which ghost is in which corner (see image).

```
python busters.py -s -a inference=MarginalInference
```



Question 6 (6 points) Complete the `observeState` method in `JointParticleFilter` to weight and resample the whole list of particles based on new evidence. A correct implementation should also handle two special cases: (1) when all your particles receive zero weight based on the evidence, you should resample all particles from the prior to recover. (2) when a ghost is eaten, you should update all particles to place that ghost in its prison cell, as described in the comments of `observeState`. You should now effectively track dispersing ghosts. If correctly implemented, your agent should win `oneHunt` with a 10-game average score greater than 480.

```
python busters.py -s -k 3 -a inference=MarginalInfe
```

Congratulations! Only one more project left.