1 Introduction

Thing are not going well on board the Heart of Gold. Zaphod, Arthur and Ford find themselves unexpectedly foiled in their quest towards the Restaurant at the End of the Universe. The sensors on the Heart of Gold seems to have become faulty, courtesy of a hack by evil video-game mogul ZyngaMania that is steadily miscalibrating them and throwing the ship off course. ZyngaMania is attempting to strong-arm the group on board to help him solve a particularly hard variant of Pacman called the Ghostbusters Edition. Luckily for them, Marvin is acquainted with a knowledge of Hidden Markov Models, which he uses to model the discrepancy between the observed sensor readings and their real life position. With a little help from you, he believes he can steady the ship long enough for ZyngaMania’s Pacman challenge to be solved, and the hack to be disabled! Of course, Zaphod, Arthur and Ford aren’t particularly good programmers, and will merely provide you with warm towels and plenty of tea, as you help them to actually solve ZyngaMania’s Pacman challenge, and disable the hack on their ship.

2 HMM Basics

Consider an HMM with two states $X$ and $Y$. The transition model is: $P(X_{t+1} \mid X_t) = 0.5, P(X_{t+1} \mid Y_t) = 0.25$. The HMM has two observations $a$ and $b$, with $P(a_t \mid X_t) = 0.25$ and $P(a_t \mid Y_t) = 0.5$. Here, $t$ is an arbitrary time index. We will assume that the initial state at time $t = 0$ is $X$.

**Question 1 (10 points).** Compute (showing your work) the predicted distribution over states at time 2. (This is the case where we have not made any observations yet.)

**Question 2 (10 points).** Suppose that the observation at time 1 is $a$, and the observation at time 2 is $b$. Compute (showing your work) the distribution over states at time 2 after making these observations.
3 Pacman: Ghostbusters Edition

Pacman spends his life running from ghosts, but things were not always so. Legend has it that many years ago, Pacman’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 assignment, you will design Pacman 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.

3. The Code

The stencil code this assignment is located in `/course/cs141/stencil/GhostBusters`

To get the code to your current directory, use the command:

```bash
cp -r /course/cs141/stencil/GhostBusters .
```

**Files you will edit**

- `bustersAgents.py` Agents for playing the Ghostbusters variant of Pacman.
- `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 Pacman
- `ghostAgents.py` Agents to control ghosts
- `graphicsDisplay.py` Graphics for Pacman
- `graphicsUtils.py` Support for Pacman graphics
- `keyboardAgents.py` Keyboard interfaces to control Pacman
- `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`.

**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.
4. Ghostbusters and HMMs

In our version of Ghostbusters, the goal is to hunt down scared but invisible ghosts. Pacman, 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 Pacman. 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 implementation task in this assignment 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.

```
python busters.py -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. Hint: As you’re debugging, you’ll find it useful to actually see where the ghost is. Use option `-s`, when running Pacman

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

**Question 3 (10 points)** Update the observe method in ExactInference class of `inference.py` to correctly update the agent’s belief distribution over ghost positions. A correct implementation should also handle one special case: when a ghost is eaten, you should place that ghost in its prison cell, as described in the comments of observe. 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 StationaryGhost ghost agents don’t move, 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:

1. 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.

2. Before typing any code, write down the equation of the inference problem you are trying to solve.

3. Try printing `noisyDistance`, `emissionModel`, and `PacmanPosition` (in the observe function) to get started.

4. In the Pacman 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.

5. Beliefs are stored as `util.Counter` objects (like dictionaries) in a field called `self.beliefs`, which you should update.

6. You should not need to store any evidence. The only thing you need to store in `ExactInference` is `self.beliefs`.

**Question 4 (10 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. To test it out, you can use the `DirectionalGhost` ghost agent, which causes the ghosts to move in a somewhat predictable fashion. If you don’t include `-g DirectionalGhost`, then the ghost will move randomly, which will be harder to track, though it should still be possible.

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

Hints:

1. 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`. 

4
2. 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.

3. 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 Pacman 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 Pacman assumes that each ghost is in its most likely position according to its beliefs, then moves toward the closest ghost.

**Question 5 (10 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. *Note:* the autograder will check the correctness of your inference directly, not the outcome of games, but it’s a reasonable sanity check.

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

**Hints:**

1. When correctly implemented, your agent will thrash around a bit in order to capture a ghost.

2. The comments of `chooseAction` provide you with useful method calls for computing maze distance and successor positions.

3. Make sure to only consider the living ghosts, as described in the comments.

**Hand-in**

You should hand in:

1. A printed document containing your answers to questions 1 and 2.

2. Python files `bustersAgents.py` and `inference.py`.

To hand in the assignment, go into `/[CSlogin]/course/cs1410/GhostBusters` and run:
Please submit your printed answers with your banner ID (not your name or CS login!) to the questions to our box #3 on the second floor of the CIT, directly outside the fishbowl.