CS440 – Ghostbusters

Purpose: 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 lab, 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.

Getting Started: Download the ghostbusters.zip files and complete each of the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies or if you get an error message that you do not understand.

In this 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 you can run `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 task in this project is to implement inference to track the ghosts.

Assignment 1 – Observation Probability:

Throughout this project, we will be using the DiscreteDistribution class defined in `inference.py` to model belief distributions. This class is an extension of the built-in Python dictionary class, where the keys are the different discrete elements of our distribution, and the corresponding values are proportional to the belief that the distribution assigns that element. First, fill in the `normalize` method, which normalizes the values in the distribution to sum to one, but keeps the proportions of the values the same. Use the total method to find the sum of the values in the distribution. For an empty distribution or a distribution where all of the values are zero, do nothing. Note that this method modifies the distribution directly, rather than returning a new distribution.

Next, you will implement the `getObservationProb` method in the `InferenceModule` base class in `inference.py`. This method takes in an observation (which is a noisy reading of the distance to the ghost), Pacman's position, the ghost's position, and the position of the ghost's jail, and returns the probability of the noisy distance reading given Pacman's position and the ghost's position. In other words, we want to return P(noisyDistance | pacmanPosition, ghostPosition).

The distance sensor has a probability distribution over distance readings given the true distance from Pacman to the ghost. This distribution is modeled by the function `busters.getObservationProbability(noisyDistance, trueDistance)`, which returns P(noisyDistance | trueDistance) and is provided for you. You should use this function to help you solve the problem, and use the provided `manhattanDistance` function to find the distance between Pacman's location and the ghost's location.

However, there is the special case of jail that you have to handle as well. Specifically, when we capture a ghost and send it to the jail location, the distance sensor deterministically returns `None`, and nothing else. So, if the ghost's position is the jail position, then the observation is `None` with probability 1, and everything else with probability 0. Conversely, if the distance reading is not `None`, then the ghost is in jail with probability 0. If the distance reading is `None`, then the ghost is in jail with probability 1. Make sure you handle this special case in your implementation.

Criteria for Success:

You may use the autograder with q1 to verify everything is working.

Assignment 2 – Exact Inference Observation:

In this assignment, you will implement the `observeUpdate` method in `ExactInference` class of `inference.py` to correctly update the agent's belief distribution over ghost positions given an observation from Pacman's sensors. You are implementing the online belief update for observing new evidence. The observe method should, for this problem, update the belief at every position on the map after receiving a sensor reading. You should iterate your updates over the variable `self.allPositions` which includes all legal positions plus the special jail position. Beliefs represent the probability that the ghost is at a particular location, and are stored as a `DiscreteDistribution` object in a field called `self.beliefs`, which you should update.

Before typing any code, write down the equation of the inference problem you are trying to solve. You should use the function `self.getObservationProb` that you wrote in the last assignment, which returns the probability of an observation given Pacman's position, a potential ghost position, and the jail position. You can obtain Pacman's position using `gameState.getPacmanPosition()`, and the jail position using `self.getJailPosition()`.

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. As you watch the test cases, be sure that you understand how the squares converge to their final coloring.

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 update function, you'll only see a single number even though there may be multiple ghosts on the board.

Criteria for Success:

You may use the autograder with q2 to verify everything is working.

Assignment 3 – Exact Inference with Time Elapse:

In the previous assignment you implemented belief updates for Pacman based on his observations. Fortunately, Pacman's observations are not his only source of knowledge about where a ghost may be. Pacman also has knowledge about the ways that a ghost may move; namely that the ghost can not move through a wall or more than one space in one time step.

To understand why this is useful to Pacman, consider the following scenario in which there is Pacman and one Ghost. Pacman receives many observations which indicate the ghost is very near, but then one which indicates the ghost is very far. The reading indicating the ghost is very far is likely to be the result of a buggy sensor. Pacman's prior knowledge of how the ghost may move will decrease the impact of this reading since Pacman knows the ghost could not move so far in only one move.

In this assignment, you will implement the `elapseTime` method in `ExactInference`. The `elapseTime` step should, for this problem, update the belief at every position on the map after one time step elapsing. Your agent has access to the action distribution for the ghost through `self.getPositionDistribution`. In order to obtain the distribution over new positions for the ghost, given its previous position, use this line of code:

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

Where `oldPos` refers to the previous ghost position. `newPosDist` is a DiscreteDistribution object, where for each position p in `self.allPositions`, `newPosDist[p]` is the probability that the ghost is at position p at time t + 1, given that the ghost is at position `oldPos` at time t. Note that this call can be fairly expensive, so think about whether or not you can reduce the number of calls to `self.getPositionDistribution`.

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

Since Pacman is not observing the ghost, this means the ghost's actions will not impact Pacman's beliefs. Over time, Pacman's beliefs will come to reflect places on the board where he believes ghosts are most likely to be given the geometry of the board and what Pacman already knows about their valid movements.

Criteria for Success:
The tests for this assignment will sometimes use a ghost with random movements and other times use the GoSouthGhost. This ghost tends to move south so over time, and without any observations, Pacman's belief distribution should begin to focus around the bottom of the board. To see which ghost is used for each test case you can look in the .test files.

You may use the autograder with q3 to verify everything is working.

Assignment 4 – Exact Inference Full Test:

Now that Pacman knows how to use both his prior knowledge and his observations when figuring out where a ghost is, he is ready to hunt down ghosts on his own. This assignment will use your `observeUpdate` and `elapseTime` implementations together, along with a simple greedy hunting strategy which you will implement for this question. In the simple greedy strategy, Pacman assumes that each ghost is in its most likely position according to his beliefs, then moves toward the closest ghost. Up to this point, Pacman has moved by randomly selecting a valid action.

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 maze distance to the closest ghost.

To find the maze distance between any two positions pos1 and pos2, use `self.distancer.getDistance(pos1, pos2)`. To find the successor position of a position after an action:

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

You are provided with `livingGhostPositionDistributions`, a list of DiscreteDistribution objects representing the position belief distributions for each of the ghosts that are still uncaptured.

Criteria for Success:

You may use the autograder with q4 to verify everything is working.

Submit your code in Canvas for grading.