HW4 Project Details Document

1. ***Question 1*** Exact Inference Observation:

- **Code Details:**

```
def observe(self, observation, gameState):
    """
    Updates beliefs based on the distance observation and Pacman's position.

    The noisyDistance is the estimated Manhattan distance to the ghost you
    are tracking.

    The emissionModel below stores the probability of the noisyDistance for
    any true distance you supply. That is, it stores P(noisyDistance |
    TrueDistance).

    self.legalPositions is a list of the possible ghost positions (you
    should only consider positions that are in self.legalPositions).

    A correct implementation will handle the following special case:
      *  When a ghost is captured by Pacman, all beliefs should be updated
         so that the ghost appears in its prison cell, position
         self.getJailPosition()

         You can check if a ghost has been captured by Pacman by
         checking if it has a noisyDistance of None (a noisy distance
         of None will be returned if, and only if, the ghost is
         captured).
    """
    noisyDistance = observation
    emissionModel = busters.getObservationDistribution(noisyDistance)
    pacmanPosition = gameState.getPacmanPosition()

    #print "noisyDistance", noisyDistance
    #print "emissionModel", emissionModel
    #print "pacmanPosition", pacmanPosition

    "*** YOUR CODE HERE ***"
    #util.raiseNotDefined()

    # Replace this code with a correct observation update
    # Be sure to handle the "jail" edge case where the ghost is eaten
    # and noisyDistance is None

    allPossible = {}
    allPossible = util.Counter()            #initialize the all possible observations

    if noisyDistance is not None:           #if sonar data available
        for p in self.legalPositions:       #iterate over all the legal positions
            trueDistance = util.manhattanDistance(p, pacmanPosition)    #calculate the distance between pacman and ghost position
            if emissionModel[trueDistance] > 0:
                allPossible[p] = emissionModel[trueDistance] * self.beliefs[p]      #set the calculated observations


    else:       #handle jail edge case
        jailP = self.getJailPosition()
        allPossible[jailP] = 1.0

    "*** END YOUR CODE HERE ***"

    allPossible.normalize()
    self.beliefs = allPossible
```

- I have updated the code in for observer() method inside ExactInference class in inference.py file.
- We were provided with noisy distance, emission model and Pacman position to calculate the observation.
- Get all the existing Pacman beliefs in allPossible dictionary.
- If noisyDistance is available then:
    - Loop over the legal positions:
        - Calculate the Manhattan distance between Pacman's position and ghost's position

        - If any positive value exist for emission model for the current distance then:
            - Calculate the new observation using the emission model and old Pacman's belief.
    - Handle the jail edge case if noisy distance not available. This means that the ghost is already eaten by Pacman.
        - Get the position of the eaten ghost
        - Mark that in the Pacman's belief system.

    - Normalize the calculated observations
    - Update the old belief system with the new one for Pacman.

2. **Question 2** Exact Inference with Time Elapse:

- **Code Details:**

```
def elapseTime(self, gameState):
    """
    Update self.beliefs in response to a time step passing from the current
    state.

    The transition model is not entirely stationary: it may depend on
    Pacman's current position (e.g., for DirectionalGhost).  However, this
    is not a problem, as Pacman's current position is known.

    In order to obtain the distribution over new positions for the ghost,
    given its previous position (oldPos) as well as Pacman's current
    position, use this line of code:

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

    Note that you may need to replace "oldPos" with the correct name of the
    variable that you have used to refer to the previous ghost position for
    which you are computing this distribution. You will need to compute
    multiple position distributions for a single update.

    newPosDist is a util.Counter object, where for each position p in
    self.legalPositions,

    newPostDist[p] = Pr( ghost is at position p at time t + 1 | ghost is at position oldPos at time t )

    (and also given Pacman's current position).  You may also find it useful
    to loop over key, value pairs in newPosDist, like:

      for newPos, prob in newPosDist.items():
        ...

    *** GORY DETAIL AHEAD ***

    As an implementation detail (with which you need not concern yourself),
    the line of code at the top of this comment block for obtaining
    newPosDist makes use of two helper methods provided in InferenceModule
    above:

      1) self.setGhostPosition(gameState, ghostPosition)
          This method alters the gameState by placing the ghost we're
          tracking in a particular position.  This altered gameState can be
          used to query what the ghost would do in this position.

      2) self.getPositionDistribution(gameState)
          This method uses the ghost agent to determine what positions the
          ghost will move to from the provided gameState.  The ghost must be
          placed in the gameState with a call to self.setGhostPosition
          above.
```

```
        It is worthwhile, however, to understand why these two helper methods
        are used and how they combine to give us a belief distribution over new
        positions after a time update from a particular position.
        """
        "*** YOUR CODE HERE ***"

        allPossible = {}
        allPossible = util.Counter()        #initialize the all possible ghost locations

        for oldPos in self.legalPositions:
            newPosDist = self.getPositionDistribution(self.setGhostPosition(gameState, oldPos))   #get the new position distribution for the ghosts

            for newPos, possibility in newPosDist.items():

                newPossibility = possibility * self.beliefs[oldPos]     #calculate the new probalility of ghosts position
                allPossible[newPos] = allPossible[newPos] + newPossibility      #Finalize the ghosts location


        self.beliefs = allPossible          #update the beliefs

        #util.raiseNotDefined()
```

- I have updated the code in for elapseTime() method inside ExactInference class in inference.py file.
- Get all the beliefs of Pacman in allPossible dictionary.
- Loop over all the positions in legal positions:
  - Get the new position distribution for ghosts
  - Loop over new positions and probabilities for this new distribution:
    - Calculate the new possibility using the old belief system and new probability value
    - Finalize the new position for the ghosts
- Update the belief system to new values for Pacman.

3. **Question 3** Exact Inference Full Test:

- **Code Details:**

```
    def chooseAction(self, gameState):
        """
        First computes the most likely position of each ghost that has
        not yet been captured, then chooses an action that brings
        Pacman closer to the closest ghost (according to mazeDistance!).

        To find the mazeDistance between any two positions, use:
          self.distancer.getDistance(pos1, pos2)

        To find the successor position of a position after an action:
          successorPosition = Actions.getSuccessor(position, action)

        livingGhostPositionDistributions, defined below, is a list of
        util.Counter objects equal to the position belief
        distributions for each of the ghosts that are still alive.  It
        is defined based on (these are implementation details about
        which you need not be concerned):

          1) gameState.getLivingGhosts(), a list of booleans, one for each
             agent, indicating whether or not the agent is alive.  Note
             that pacman is always agent 0, so the ghosts are agents 1,
             onwards (just as before).

          2) self.ghostBeliefs, the list of belief distributions for each
             of the ghosts (including ghosts that are not alive).  The
             indices into this list should be 1 less than indices into the
             gameState.getLivingGhosts() list.
        """
```

```
pacmanPosition = gameState.getPacmanPosition()          #get pacman's current position
legal = [a for a in gameState.getLegalPacmanActions()]      #get all the legal actions
livingGhosts = gameState.getLivingGhosts()              #get all the living ghosts
livingGhostPositionDistributions = [beliefs for i, beliefs in enumerate(self.ghostBeliefs) if livingGhosts[i+1]]      #get all the living ghosts as per pacman's beliefs
"*** YOUR CODE HERE ***"

bestAction = None               #set best action
bestActionDist = None
mostLikelyGhostPostition = []

for dist in livingGhostPositionDistributions:       #iterate over living ghosts distribution
    mostLikelyProb = None
    mostLikelyPos = None

    for position, probability in dist.items():
        if mostLikelyProb == None or probability > mostLikelyProb:
            mostLikelyProb = probability            #find the mostly likely probability of ghost
            mostLikelyPos = position                #find the mostly likely position of ghost

    mostLikelyGhostPostition.append(mostLikelyPos)      #accumulate all the positions


for ghostPosition in mostLikelyGhostPostition:          #iterate over mostly likely ghost positions
    for action in legal:                            #iterate over valid actions
        successorPosition = Actions.getSuccessor(pacmanPosition, action)        #apply action and get next location of pacman
        successorDist = self.distancer.getDistance(successorPosition, ghostPosition)    #find distance from pacman to ghost

        if bestAction == None or successorDist < bestActionDist:        #choose the minimum distance
            bestActionDist = successorDist
            bestAction = action                                         #choose the best action


#util.raiseNotDefined()
return bestAction                       #return the best action calculated
```

- I have updated chooseAction() method in GreedyBusterAgent class in bustersAgents.py
- We were provided with Pacman's current position in the game.
- Get all the legal actions that are allowed for Pacman.
- Get all the living ghost's positions and validate them as per the Pacman's actual belief system.
- Define bestAction, bestActionDist and mostLikelyGhost positions.
- Loop over the distribution of living ghosts:
    - Define mostlikelyprob and mostlikelypos
    - For each position and probability over each distribution:
        - If mostlikelyprob is none or less than the distribution probability then:
            - Set this as new mostlikely prob
            - Set mostlikelypos as distribution position
    - Append the new position to the ghost position list

- For each ghost positions in ghost position list:
    - For each legal actions:
        - Get the successor position for Pacman's current position
        - Find the distance from Pacman's successor to ghost
        - If best action is none or successor's distance is less than best action distance then:
            - Set this as new best distance action
            - Set this action as new best action
- At the end of the execution return the best action from this greedy buster agent.