ghostAgents.py (original)

```
# ghostAgents.py
# --------------
# Licensing Information: Please do not distribute or publish solutions to this
# project. You are free to use and extend these projects for educational
# purposes. The Pacman AI projects were developed at UC Berkeley, primarily by
# John DeNero (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# For more info, see http://inst.eecs.berkeley.edu/~cs188/sp09/pacman.html

from game import Agent
from game import Actions
from game import Directions
import random
from util import manhattanDistance
import util

class GhostAgent( Agent ):
  def __init__( self, index ):
    self.index = index

  def getAction( self, state ):
    dist = self.getDistribution(state)
    if len(dist) == 0:
      return Directions.STOP
    else:
      return util.chooseFromDistribution( dist )
```

```
  def getDistribution(self, state):
vided state." "Returns a Cou
    util.raiseNotD
```

```
class RandomGhost(
  "A ghost that chooses a legal action uniformly at random."
  def getDistribution( self, state ):
    dist = util.Counter()
    for a in state.getLegalActions( self.ind
```

```
    dist.normalize()
    return dist

class DirectionalGhost( GhostAgent ):
  "A ghost that prefers to rush Pacman, or flee when scared."
  def __init__( self, index, prob_attack=0.8, prob_scaredFlee=0.8 ):
    self.index = index
    self.prob_attack = prob_attack
    self.prob_scaredFlee = prob_scaredFlee

  def getDistribution( self, state ):
    # Read variables from state
    ghostState = state.getGhostState( self.index )
    legalActions = state.getLegalActions( self.index )
    pos = state.getGhostPosition( self.index )
    isScared = ghostState.scaredTimer > 0

    speed = 1
    if isScared: speed = 0.5

    actionVectors = [Actions.directionToVector( a, speed ) for a in legalActions]
    newPositions = [( pos[0]+a[0], pos[1]+a[1] ) for a in actionVectors]
    pacmanPosition = state.getPacmanPosition()

    # Select best actions given the state
    distancesToPacman = [manhattanDistance( pos, pacmanPosition ) for pos in
newPositions]
    if isScared:
      bestScore = max( distancesToPacman )
```

```
        bestProb = self.prob_scaredFlee
      else:
        bestScore = min( distancesToPacman )
        bestProb = self.prob_attack
      bestActions = [action for action, distance in zip( legalActions,
distancesToPacman ) if distance == bestScore]

      # Construct distribution
      dist = util.Counter()
      for a in bestActions: dist[a] = bestProb / len(bestActions)
      for a in legalActions: dist[a] += ( 1-bestProb ) / len(legalActions)
      dist.normalize()
      return dist
```