featureExtractors.py (original)

```python
# featureExtractors.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

"Feature extractors for Pacman game states"

from game import Directions, Actions
import util

class FeatureExtractor:
  def getFeatures(self, state, action):
    """
      Returns a dict from features to counts
      Usually, the count will just be 1.0 for
      indicator functions.
    """
    util.raiseNotDefined()

class IdentityExtractor(FeatureExtractor):
  def getFeatures(self, state, action):
    feats = util.Counter()
    feats[(state,action)] = 1.0
    return feats

def closestFood(pos, food, walls):
  """
  closestFood -- this is similar to the function that we have
  worked on in the search project; here its all in one place
  """
  fringe = [(pos[0], pos[1], 0)]
  expanded = set()
  while fringe:
    pos_x, pos_y, dist = fringe.pop(0)
    if (pos_x, pos_y) in expanded:
      continue
    expanded.add((pos_x, pos_y))
    # if we find a food at this location then exit
    if food[pos_x][pos_y]:
      return dist
    # otherwise spread out from the location to its neighbours
    nbrs = Actions.getLegalNeighbors((pos_x, pos_y), walls)
    for nbr_x, nbr_y in nbrs:
      fringe.append((nbr_x, nbr_y, dist+1))
  # no food found
  return None

class SimpleExtractor(FeatureExtractor):
  """
  Returns simple features for a basic reflex Pacman:
  - whether food will be eaten
  - how far away the next food is
  - whether a ghost collision is imminent
  - whether a ghost is one step away
  """

  def getFeatures(self, state, action):
    # extract the grid of food and wall locations and get the ghost locations
    food = state.getFood()
    walls = state.getWalls()
    ghosts = state.getGhostPositions()
    features = util.Counter()

    features["bias"] = 1.0

    # compute the location of pacman after he takes the action
    x, y = state.getPacmanPosition()
    dx, dy = Actions.directionToVector(action)
    next_x, next_y = int(x + dx), int(y + dy)

    # count the number of ghosts 1-step away
    features["#-of-ghosts-1-step-away"] = sum((next_x, next_y) in
Actions.getLegalNeighbors(g, walls) for g in ghosts)

    # if there is no danger of ghosts then add the food feature
    if not features["#-of-ghosts-1-step-away"] and food[next_x][next_y]:
      features["eats-food"] = 1.0

    dist = closestFood((next_x, next_y), food, walls)
    if dist is not None:
      # make the distance a number less than one otherwise the update
      # will diverge wildly
      features["closest-food"] = float(dist) / (walls.width * walls.height)
    features.divideAll(10.0)
    return features
```