layout.py (original)

```
# layout.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 util import manhattanDistance
from game import Grid
import os
import random

VISIBILITY_MATRIX_CACHE = {}

class Layout:
  """
  A Layout manages the static information about the game board.
  """

  def __init__(self, layoutText):
    self.width = len(layoutText[0])
    self.height= len(layoutText)
    self.walls = Grid(self.width, self.height, False)
    self.food = Grid(self.width, self.height, False)
    self.capsules = []
    self.agentPositions = []
    self.numGhosts = 0
    self.processLayoutText(layoutText)
    self.layoutText = layoutText
    # self.initializeVisibilityMatrix()

  def getNumGhosts(self):
    return self.numGhosts

  def initializeVisibilityMatrix(self):
    global VISIBILITY_MATRIX_CACHE
    if reduce(str.__add__, self.layoutText) not in VISIBILITY_MATRIX_CACHE:
      from game import Directions
      vecs = [(-0.5,0), (0.5,0),(0,-0.5),(0,0.5)]
      dirs = [Directions.NORTH, Directions.SOUTH, Directions.WEST, Directions.EAST]
      vis = Grid(self.width, self.height, {Directions.NORTH:set(),
Directions.SOUTH:set(), Directions.EAST:set(), Directions.WEST:set(),
Directions.STOP:set()})
      for x in range(self.width):
        for y in range(self.height):
          if self.walls[x][y] == False:
            for vec, direction in zip(vecs, dirs):
              dx, dy = vec
              nextx, nexty = x + dx, y + dy
              while (nextx + nexty) != int(nextx) + int(nexty) or not
self.walls[int(nextx)][int(nexty)] :
                vis[x][y][direction].add((nextx, nexty))
                nextx, nexty = x + dx, y + dy
      self.visibility = vis
      VISIBILITY_MATRIX_CACHE[reduce(str.__add__, self.layoutText)] = vis
    else:
      self.visibility = VISIBILITY_MATRIX_CACHE[reduce(str.__add__, self.layoutText)]

  def isWall(self, pos):
    x, col = pos
    return self.walls[x][col]

  def getRandomLegalPosition(self):
    x = random.choice(range(self.width))
    y = random.choice(range(self.height))
    while self.isWall( (x, y) ):
      x = random.choice(range(self.width))
      y = random.choice(range(self.height))
    return (x,y)

  def getRandomCorner(self):
    poses = [(1,1), (1, self.height - 2), (self.width - 2, 1), (self.width - 2,
self.height - 2)]
    return random.choice(poses)

  def getFurthestCorner(self, pacPos):
    poses = [(1,1), (1, self.height - 2), (self.width - 2, 1), (self.width - 2,
self.height - 2)]
    dist, pos = max([(manhattanDistance(p, pacPos), p) for p in poses])
    return pos

  def isVisibleFrom(self, ghostPos, pacPos, pacDirection):
    row, col = [int(x) for x in pacPos]
    return ghostPos in self.visibility[row][col][pacDirection]

  def __str__(self):
    return "\n".join(self.layoutText)

  def deepCopy(self):
    return Layout(self.layoutText[:])

  def processLayoutText(self, layoutText):
    """
    Coordinates are flipped from the input format to the (x,y) convention here

    The shape of the maze.  Each character
    represents a different type of object.
     % - Wall
     . - Food
     o - Capsule
     G - Ghost
     P - Pacman
    Other characters are ignored.
    """
    maxY = self.height - 1
    for y in range(self.height):
      for x in range(self.width):
        layoutChar = layoutText[maxY - y][x]
        self.processLayoutChar(x, y, layoutChar)
    self.agentPositions.sort()
    self.agentPositions = [ ( i == 0, pos) for i, pos in self.agentPositions]

  def processLayoutChar(self, x, y, layoutChar):
    if layoutChar == '%':
      self.walls[x][y] = True
    elif layoutChar == '.':
      self.food[x][y] = True
    elif layoutChar == 'o':
      self.capsules.append((x, y))
    elif layoutChar == 'P':
      self.agentPositions.append( (0, (x, y) ) )
    elif layoutChar in ['G']:
      self.agentPositions.append( (1, (x, y) ) )
      self.numGhosts += 1
    elif layoutChar in  ['1', '2', '3', '4']:
      self.agentPositions.append( (int(layoutChar), (x,y)))
      self.numGhosts += 1
def getLayout(name, back = 2):
  if name.endswith('.lay'):
    layout = tryToLoad('layouts/' + name)
    if layout == None: layout = tryToLoad(name)
  else:
    layout = tryToLoad('layouts/' + name + '.lay')
    if layout == None: layout = tryToLoad(name + '.lay')
  if layout == None and back >= 0:
    curdir = os.path.abspath('.')
    os.chdir('..')
    layout = getLayout(name, back -1)
    os.chdir(curdir)
  return layout

def tryToLoad(fullname):
  if(not os.path.exists(fullname)): return None
  f = open(fullname)
  try: return Layout([line.strip() for line in f])
  finally: f.close()
```