#----------------------------------------------------------------------------

# Name: beliefs

# Purpose: Homework 7

#

# Author:

#

# ---------------------------------------------------------------------------

"""

Module to track the belief distribution over all possible grid positions

Your task for homework 7 is to implement:

1. update

2. recommend\_sensing

"""

**import** utils

**class** **Belief**(object):

"""

Belief class used to track the belief distribution based on the sensing

evidence we have so far.

Arguments:

size (int): the number of rows/columns in the grid

Attributes:

open (set of tuples): set containing all the positions that have not

been observed so far.

current\_distribution (dictionary): probability distribution based on

the evidence acquired so far.

The keys of the dictionary are the possible grid positions

The values represent the (conditional) probability that the

treasure is found at that position given the evidence (sensor data)

accumulated so far.

"""

**def** **\_\_init\_\_**(self, size):

# Initially all positions are open - have not been observed

self.open = {(x, y) **for** x **in** range(size)

**for** y **in** range(size)}

# Initialize to a uniform distribution

self.current\_distribution = {pos: 1 / (size \*\* 2) **for** pos **in** self.open}

**def** **get\_beliefs**(self):

"""

Accessor method for the belief distribution

Note: to be used in treasurehunt.py only for a cleaner interface

You do not need to use it inside the Belief class.

:return:

dictionary representing the belief distribution given the evidence

accumulated so far.

"""

**return** self.current\_distribution

**def** **update**(self, color, sensor\_position, model):

"""

Update the belief distribution based on new evidence: our agent

detected the given color at sensor location: sensor\_position.

:param color: (string) color detected

:param sensor\_position: (tuple) position of the sensor

:param model (Model object) models the relationship between the

treasure location and the sensor data

:return: None

"""

for position in sensor\_position:

model.pcolorgivendist()

self.open

# Iterate over ALL positions in the grid and update the probability

# of finding the treasure at that position - given the new evidence

# The probability of the evidence given the Manhattan distance to the

# treasure may be accessed by calling model.pcolorgivendist.

# Don't forget to normalize.

# Don't forget to update self.open since sensor\_position has now been

# observed

# Iterate over All positions in the grid and update the probability of finding treasure

# Normalize

# Remove sensor\_position from the open set

**def** **recommend\_sensing**(self):

"""

Recommend where we should take the next measurement in the grid.

The position should be the most promising unobserved location.

If all remaining unobserved locations have a probability of 0, return

the unobserved location that is closest to the (observed) location with

the highest probability.

:return: tuple representing the position where we should take the

next measurement

"""

# Enter your code and remove the statement below

# Create dictionary with keys as the open and value as the probability

# Get the open position with the best probability

# check if all open probabilities are 0

# Find the position with the highest probability from the visited

# Get the location from open that is closest to the observed max above