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

# Name: treasurehunt

# Purpose: Homework 7

#

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

"""

A Treasure hunt under the sea

Use noisy sonar to sense the treasure first then dive to get it

"""

**import** argparse

**import** tkinter

**import** random

**import** utils

**from** beliefs **import** Belief

**class** **Problem**(object):

"""

Problem class

Argument:

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

Attribute:

treasure (tuple): the location of the treasure (x, y)

"""

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

# Pick a random location for the treasure

x = random.randint(0, size - 1)

y = random.randint(0, size - 1)

self.treasure = x, y

**def** **treasure\_found**(self, pos):

"""

Determine if the dive is successful

:param pos (tuple): diving position

:return: boolean - True if the treasure is at that position

and False otherwise.

"""

**return** pos == self.treasure

**class** **Model**(object):

"""

Model class used to describe our sensor model

Argument:

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

"""

# Class variable pcolorgivendist

# The initial probabilities for distance <= 7

# This dictionary is extended in \_\_init\_\_ to include all values allowed

# for a game of the given size

# This is how to interpret the dictionary below:

# P(red|d=0) = 0.7, P(red|d=1) = 0.3, P(red|d=2) = 0.1,

# P(red|d>=3) = 0

# P(yellow|d=0) = 0.3, P(yellow|d=1) = 0.6, P(yellow|d=2) = 0.6, ...

sonar\_model = {'red': [0.7, 0.3, 0.1, 0],

'yellow': [0.3, 0.6, 0.6, 0.4, 0.3, 0.2, 0.1, 0],

'green': [0, 0.1, 0.3, 0.6, 0.7, 0.8, 0.9, 1]}

**def** **observe**(self, pos, problem):

"""

Return the color returned when the sonar is aimed at the given location

:param pos: (tuple) sonar position

:param problem: (Problem object)

:return:

color (string) : sensor reading

"""

dist = utils.manhattan\_distance(pos, problem.treasure)

color = self.sample(dist)

**return** color

**def** **sample**(self, distance):

"""

Sample a color reading based on the sensor model we have

:param distance: (integer) Manhattan distance to the treasure

:return: color (string) sensor reading

"""

dist = []

**for** each\_color **in** self.sonar\_model:

farthest = len(self.sonar\_model[each\_color]) - 1

**if** distance > farthest:

d = farthest

**else**:

d = distance

**for** i **in** range(int(self.sonar\_model[each\_color][d] \* 100)):

dist.append(each\_color)

**return** random.choice(dist)

**def** **pcolorgivendist**(self, color, dist):

"""

Return the conditional probability of the given color given the

distance

:param color: (string) sensor reading

:param dist: distance from the sensor location to a given position

:return: float probability

"""

farthest = len(self.sonar\_model[color]) - 1

**if** dist <= farthest:

**return** self.sonar\_model[color][dist]

**else**:

**return** self.sonar\_model[color][farthest]

**class** **Game**(object):

"""

Game class for the treasure hunt

Arguments:

parent: the root window object

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

mode (string): discovery or guided? No probabilities are shown in

discovery mode and no sensor recommendations are shown.

Attributes

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

mode (string): discovery or guided?

canvas (Canvas): tkinter widget

model (Model object) our world model of how the location of the treasure

affects the sensor readings

problem (Problem object) the specific problem (with random treasure

location that we are trying to solve

belief (Belief object) belief distribution based on the sensing

evidence we have so far

text (canvas text object): used to display the probability of each location

message (Label) tkinter widget

"""

square\_size = 70 # length in pixels of the side of a grid square

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

self.size = size

self.mode = mode

parent.title('Under the Sea')

sensor\_button = tkinter.Button(parent, text='SONAR',

command=self.sensor\_mode)

sensor\_button.grid()

action\_button = tkinter.Button(parent, text='DIVE',

command=self.diving\_mode)

action\_button.grid()

self.canvas = tkinter.Canvas(parent,

width=self.size \* self.square\_size,

height=self.size \* self.square\_size)

self.canvas.grid()

self.text = [[ None **for** row **in** range(self.size)]

**for** col **in** range(self.size)]

# create the squares on the canvas

**for** row **in** range(self.size):

**for** column **in** range(self.size):

self.canvas.create\_rectangle(column \* self.square\_size,

row \* self.square\_size,

(column + 1) \* self.square\_size,

(row + 1) \* self.square\_size,

fill = '#83DFFD',

outline = 'white')

self.text[column][row] = self.canvas.create\_text(

(column + 0.5) \* self.square\_size,

(row+ 0.5) \* self.square\_size,

text='', fill='black')

self.message = tkinter.Label(parent)

self.message.grid()

self.treasure\_img = tkinter.PhotoImage(file="treasure.gif")

self.oct = tkinter.PhotoImage(file="octopus.gif")

self.model = Model()

self.problem = Problem(size)

self.belief = Belief(size)

**if** self.mode == 'guided':

self.showbeliefs()

**def** **sensor\_mode**(self):

"""

Set the game to sensor mode - the method is invoked when the "SONAR"

button is clicked.

Once the game is in sensor mode, clicks on the grid result in sensor

readings.

"""

self.canvas.bind("<Button-1>", self.sense)

self.message.configure(text='SONAR MODE')

**def** **diving\_mode**(self):

"""

Set the game to diving mode - the method is invoked when the "DIVER"

button is clicked.

Once the game is in diving mode, clicks on the grid reveal whether the

treasure is found at that location.

"""

self.canvas.bind("<Button-1>", self.dive)

self.message.configure(text='DIVING MODE')

**def** **dive**(self, event):

"""

A click at a given location represents a dive.

Check if the dive is successful and update the GUI

"""

y = event.y // self.square\_size

x = event.x // self.square\_size

square = self.canvas.find\_closest((x + 0.1) \* self.square\_size,

(y + 0.1)\* self.square\_size)[0]

self.canvas.itemconfigure(square, fill="black")

**if** self.problem.treasure\_found((x, y)):

self.canvas.create\_image(

(x + 0.5) \* self.square\_size,

(y + 0.5) \* self.square\_size,

image=self.treasure\_img)

self.message.configure(text='Treasure chest found!!!')

**else**:

self.canvas.create\_image(

(x + 0.5) \* self.square\_size,

(y + 0.5) \* self.square\_size,

image=self.oct)

self.message.configure(text='Nope. Try again')

**def** **sense**(self, event):

"""

A click at a given location represents a sonar reading.

"""

x = event.x // self.square\_size

y = event.y // self.square\_size

square = self.canvas.find\_closest((x + 0.1) \* self.square\_size,

(y + 0.1)\* self.square\_size)[0]

sensing\_position = (x, y)

color = self.model.observe(sensing\_position, self.problem)

self.belief.update(color, sensing\_position, self.model)

self.mark(square, color)

**if** self.mode == 'guided':

self.showbeliefs()

self.show\_recommendation()

**def** **showbeliefs**(self):

"""

Show the current belief distribution on the GUI

"""

b = self.belief.get\_beliefs()

**for** i **in** range(self.size):

**for** j **in** range(self.size):

message = f'{b[(i, j)]:4.1}'

self.canvas.itemconfigure(self.text[i][j],

text=message,

fill='black')

**def** **show\_recommendation**(self):

"""

Show the recommendation returned by the belief module in purple

on the GUI.

"""

next\_position = self.belief.recommend\_sensing()

**if** next\_position != NotImplemented:

nx, ny = next\_position

square = self.canvas.find\_closest((nx + 0.1) \* self.square\_size,

(ny + 0.1) \* self.square\_size)[0]

self.canvas.itemconfigure(square, fill='purple')

**def** **mark**(self, square, color):

"""

Mark a given square with the specified color.

:param square: tkinter id of the square to be marked

:param color: (string) 'red', 'green' or 'yellow'

"""

self.canvas.itemconfigure(square, fill=color)

**def** **get\_arguments**():

"""

Parse and validate the command line arguments

:return: (tuple containing the size of the game and the algorithm

"""

parser = argparse.ArgumentParser()

parser.add\_argument('size',

help='How many columns/rows in the grid?',

nargs = '?',

type=int,

default=8)

parser.add\_argument('mode',

help='discovery or guided?',

nargs = '?',

type=str,

choices=['discovery', 'guided'],

default='guided')

arguments = parser.parse\_args()

size = arguments.size

mode = arguments.mode

**return** size, mode

**def** **main**():

# TODO remove for final submission

# remove randomness to get predictable results

# random.seed(1)

size, mode = get\_arguments()

# Instantiate a root window

root = tkinter.Tk()

# Instantiate a Game object

my\_game = Game(root, size, mode)

# Enter the main event loop

root.mainloop()

**if** \_\_name\_\_ == '\_\_main\_\_':

main()