Alexander Giang

REDID: 818285579

CS550 Assignment 01

1. Agents
   1. For preventing E.coli O157:H7 while monitoring production of soup, a possible precept would be temperature. If meats are cooked to a certain temperature, it would be E.coli-free, so keeping soup at a high enough temperature would keep the soup safe from E.coli. <https://www.foodsafety.gov/blog/2016/05/e-coli.html>
   2. If temperature is below the optimal temperature, raise the temperature of the soup, otherwise do nothing.
2. 3.6b
   1. Initial State: A 3-foot-tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. The room contains two stackable, movable, climbable 3-foot-high crates.
   2. Goal test: Monkey can grab the bananas
   3. Successors: stack on crate, stack off crate, move crate from current to desired spot, climb on crate, climb off crate, check height away from bananas, grab bananas (if height <= 0), move monkey from current to desired spot
   4. Cost Functions: number of successors

*'''*

*file: driver.py*

*'''*

from boardtypes import TileBoard

up = [-1,0]

down = [1,0]

left = [0,-1]

right = [0,1]

none = [0,0]

tileBoard = TileBoard(8);

tileBoard.get\_actions() # need this for location of empty tile

print(*"Tile Game Start: "*)

print(*"To move the blank tile: Type 'left', 'right', 'up', or 'down'"*)

print(*"To leave game: Type 'stop' or 'exit'"*)

print(tileBoard.\_\_repr\_\_())

while not tileBoard.solved():

actions = tileBoard.get\_actions()

actionList = []

for i in range(len(actions)):

if actions[i] == left:

actionList.append(*"left"*)

if actions[i] == right:

actionList.append(*"right"*)

if actions[i] == up:

actionList.append(*"up"*)

if actions[i] == down:

actionList.append(*"down"*)

print(*"Available Actions: "*)

print(actionList)

count = 0

while not count == 1:

user = input()

if user in actionList and user == *"left"*:

temp = tileBoard.move(left)

count += 1

if user in actionList and user == *"right"*:

temp = tileBoard.move(right)

count += 1

if user in actionList and user == *"up"*:

temp = tileBoard.move(up)

count += 1

if user in actionList and user == *"down"*:

temp = tileBoard.move(down)

count += 1

if user == *"stop"* or *"exit"*:

print(*"Better luck next time"*)

exit()

else:

print(*"Please try another move."*)

print(temp)

tileBoard = temp

print(*"Congrats! You've solved it!"*)

*'''*

*file: boardtypes.py*

*'''*

import math

import random

import copy

from board import Board

class **TileBoard**(Board):

def **\_\_init\_\_**(*self*, n, force\_state=None):

*"""TileBoard(n, force\_state)*

*Creates an n-puzzle of size n. Example TileBoard(8) creates the 8-*

*puzzle shown below. (Note that we are using your book’s notation which*

*refers to the number of tiles as opposed to the number of squares.)*

*"""*

*self*.boardsize = size = int(math.sqrt(n+1))

if isinstance(size, float):

raise ValueError(*"Board size must be 1 less than a perfect square, "*

+ *"such as 8, 15, 24, etc"*)

# Create the initial state

tilesList = [x+1 for x in range(n)]

tilesList.append(None)

solvable = False

while not solvable:

random.shuffle(tilesList)

# Compute inversion order length defined in A01

length = 0

for i in range(len(tilesList)):

perm = tilesList[i]

for j in range(i,len(tilesList)):

if tilesList[j] != None and perm != None and \

tilesList[j] < perm:

length += 1

# Deals with empty tile condition

if size % 2 == 0:

length += math.floor(tilesList.index(None) / size)+1

solvable = length % 2 == 0

# initialize the board after shuffle

super(TileBoard, *self*).\_\_init\_\_(size, size)

for row in range(size):

for col in range(size):

tile = tilesList[row\*size + col]

if tile:

*self*.place(row, col, tile)

else:

*self*.empty = (row, col)

def **\_\_eq\_\_**(*self*, other):

*"Check if two tile boards are in the same state. "*

pair = ()

eq = True

for (this\_board, other\_board) in pair(*self*.state\_tuple(),

other.state\_tuple()):

eq = this\_board == other\_board

if not eq:

break

return eq

def **state\_tuple**(*self*):

*"Flatten the list of list representation of the board & cast it to a \*

*tuple."*

flatList = []

# Merge the items in each list of lists

for tupList in *self*.board:

for x in tupList:

flatList.append(x)

return tuple(flatList)

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

*"Return list of possible moves in [row\_delta,col\_delta] format"*

moves = []

up = [-1,0]

down = [1,0]

left = [0,-1]

right = [0,1]

none = [0,0]

# find Blank Tile to know the row and column

for i in range(*self*.get\_rows()):

for j in range(*self*.get\_cols()):

if *self*.get(i,j) == None:

row = i

col = j

*self*.blank = (i,j)

# Knowing row and column make it easy to check for available actions

if row-1 >= 0:

moves.append(up)

if row+1 <= *self*.get\_rows()-1:

moves.append(down)

if col-1 >= 0:

moves.append(left)

if col+1 <= *self*.get\_cols()-1:

moves.append(right)

else:

moves.append(none)

return moves

def **move**(*self*, offset):

*"move - Move the empty space by [delta\_row, col\_delta] and return new \*

*board"*

# Current row and column of empty space "blank"

(row, col) = *self*.blank

[delta\_row, col\_delta] = offset

# calculate move if possible

row\_move = row + delta\_row

col\_move = col + col\_delta

if offset not in *self*.get\_actions():

raise ValueError(*"Can't do this move"*)

input()

# Copy board with desired move

newboard = copy.deepcopy(*self*)

newboard.place(row, col, *self*.get(row\_move, col\_move))

newboard.place(row\_move, col\_move, None)

newboard.blank = (row\_move, col\_move)

return newboard

def **solved**(*self*):

*"solved - Is the puzzle solved?"*

temp = 0

# 1st condition: empty space must be in center

center = (*self*.boardsize-1) / 2

solved = *self*.blank == (center, center)

# 2nd condition: check if tiles are in order

if solved:

for tiles in *self*.state\_tuple():

if tiles:

solved = tiles == temp + 1

if not solved:

break

temp = tiles

return solved

----------------------------------------Output----------------------------------------------

Tile Game Start:

To move the blank tile: Type 'left', 'right', 'up', or 'down'

To leave game: Type 'stop' or 'exit'

0 1 2

0 4 3 5

1 . 2 1

2 6 7 8

Available Actions:

['up', 'down', 'right']

up

0 1 2

0 . 3 5

1 4 2 1

2 6 7 8

Available Actions:

['down', 'right']

up

Please try another move.

stop

Better luck next time