7320.Midterm.RoboVac

Charles Bryan
49350684

Level	Pct Clean	Cycles	File
0	100.0%	186	RoboVac0.py
1	100.0%	196	RoboVac1.py
2	100.0%	189	RoboVac2.py
3	100.0%	250	RoboVac3.py
4	100.0%	197	RoboVac4.py
5	100.0%	207	RoboVac5.py

RoboVac5.py

```
import random
import numpy as np

class RoboVac:
    def __init__(self, config_list):
        self.room_width, self.room_height = config_list[0]
        self.pos = config_list[1]  # starting position of vacuum
        self.block_list = config_list[2]  # blocks list (x, y, width, height)

        self.last_pos = None
        self.last_chosen_random_unclean_tile = None
        self.is_move_towards_tile_x_first = False
        self.is_move_towards_tile_y_first = False
        self.coin_flip = False

        # fill in with your info
        self.name = "Charles Bryan"
        self.id = "49350684"
        # Initialize block_tiles_set based on the block_list
        self.block_tiles_set = set()
        for block in self.block_list:
            for x in range(block[0], block[0] + block[2]):
                for y in range(block[1], block[1] + block[3]):
                    self.block_tiles_set.add((x, y))

        # Initialize variables for DFS
        self.max_depth = self.room_width * self.room_height
        self.dfs_stack = [(self.pos, [self.pos])]
        self.visited = set()


        # Initialize free_tiles_set
        self.uncleaned_tiles = set((x, y) for x in
range(self.room_width) for y in range(self.room_height))
        self.uncleaned_tiles -= self.block_tiles_set

        # find clean title variables
        self.next_clean_tile = None
        self.moving_toward_clean_title = False
        self.x_iteration_towards_tile = 0
        self.x_iterations_towards_tile = 0
        self.y_iteration_towards_tile = 0
        self.y_iterations_towards_tile = 0
        self.total_iterations_towards_tile =
abs(self.x_iterations_towards_tile) +
abs(self.y_iterations_towards_tile)


    def get_next_move(self, current_pos):
        self.pos = current_pos # Update the current position
        self.uncleaned_tiles.discard(self.pos)
        if self.pos == self.last_pos:
            self.reset_moving_toward_clean_tile_variables()

        if self.moving_toward_clean_title:
            self.last_pos = self.pos
            move_direction = self.move_torward_clean_title()
            return move_direction

        move_direction = self.dfs_iter(self.max_depth)
        if move_direction is not None:
            self.last_pos = self.pos
            self.visited.add(self.pos)
            return move_direction
        else:
            # since DFS returned NONE, we want to restart it on
restart DFS on next tile
            self.set_moving_toward_clean_tile_variables()
            move_direction = self.move_torward_clean_title()

        if move_direction is None:
            move_direction = random.choice([0, 1, 2, 3])

        self.last_pos = self.pos
        new_pos = self.get_new_pos(move_direction, self.pos)
        self.dfs_stack = [(new_pos, [new_pos])]

        return move_direction


    def dfs_iter(self, limit):
        while self.dfs_stack:
            (vertex, path) = self.dfs_stack.pop()

            if len(path) > limit:
                continue  # Ignore paths longer than limit

            if vertex not in self.visited:
                self.visited.add(vertex)

                directions = [(0, -1), (1, 0), (0, 1), (-1, 0)]
                for i, (dx, dy) in enumerate(directions):
                    move_direction = i
                    self.next_pos = (vertex[0] + dx, vertex[1] + dy)

                    is_next_pos_x_valid = 0 <= self.next_pos[0] <
self.room_width
                    is_next_pos_y_valid = 0 <= self.next_pos[1] <
self.room_height
```

```
                    is_next_pos_valid = is_next_pos_x_valid and 
is_next_pos_y_valid and self.next_pos not in self.block_tiles_set and 
self.next_pos not in self.visited
                    if is_next_pos_valid:
                        self.dfs_stack.append((self.next_pos, path + 
[self.next_pos]))
                        return move_direction

        return None


    def get_new_pos(self, dir, curr_pos):
        x, y = curr_pos
        if dir == 0:  # Up
            return (x, y - 1)
        elif dir == 1:  # Right
            return (x + 1, y)
        elif dir == 2:  # Down
            return (x, y + 1)
        elif dir == 3:  # Left
            return (x - 1, y)
        else:
            return 'Invalid direction'


    def set_moving_toward_clean_tile_variables(self):
        self.moving_toward_clean_title = True
        self.next_clean_tile = self.find_next_clean_tile(self.pos)
        self.x_iterations_towards_tile, self.y_iterations_towards_tile 
= self.calculate_moves_to_tile(self.pos, self.next_clean_tile)
        self.total_iterations_towards_tile = 
abs(self.x_iterations_towards_tile) + 
abs(self.y_iterations_towards_tile)


    def find_next_clean_tile(self, current_pos):
        if not self.uncleaned_tiles:
            print('uncleaned_tiles is NONE!')
            return None

        # look for closest/neighbor nodes first
        directions = [(0, -1), (1, 0), (0, 1), (-1, 0)]  # up, right,
down, left
        for dx, dy in directions:
            self.next_pos = (current_pos[0] + dx, current_pos[1] + dy)
            if self.next_pos in self.uncleaned_tiles:
                return self.next_pos

        # If no neighbor clean tile was found, return a random title
        random_unclean_tile =
random.choice(list(self.uncleaned_tiles))
        while random_unclean_tile ==
self.last_chosen_random_unclean_tile and len(self.uncleaned_tiles) >
1:
                random_unclean_tile =
random.choice(list(self.uncleaned_tiles))
        else:
            # if the only unclean tile is the last chosen unclean
tile, return it
            pass

        self.last_chosen_random_unclean_tile = random_unclean_tile
        return random_unclean_tile


    def move_torward_clean_title(self):
        move_direction = None
        if not self.uncleaned_tiles:
            return move_direction

        if not self.is_move_towards_tile_x_first or not
self.is_move_towards_tile_y_first:
            if self.coin_flip:
                self.coin_flip = False
                self.is_move_towards_tile_x_first = True
            else:
                self.coin_flip = True
                self.is_move_towards_tile_y_first = True

        if self.is_move_towards_tile_x_first:
            move_direction = self.move_towards_tile_x_first()
        else:
            move_direction = self.move_towards_tile_y_first()
        return move_direction


    def move_towards_tile_x_first(self):
        if self.x_iteration_towards_tile <
abs(self.x_iterations_towards_tile):
            self.x_iteration_towards_tile += 1
            move_left = self.x_iterations_towards_tile < 0
            if move_left:
                move_direction = 3 # Left
            else: # move_right
                move_direction = 1  # Right
        elif self.y_iteration_towards_tile <
abs(self.y_iterations_towards_tile):
            self.y_iteration_towards_tile += 1
            move_down = self.y_iterations_towards_tile > 0
            if move_down:
                move_direction = 2  # Down
            else: # move_up
                move_direction = 0  # Up
        else:
            print('FOUND TARGET TILE')
            self.reset_moving_toward_clean_tile_variables()
            return
        return move_direction


    def move_towards_tile_y_first(self):
        if self.y_iteration_towards_tile <
abs(self.y_iterations_towards_tile):
            self.y_iteration_towards_tile += 1
            move_down = self.y_iterations_towards_tile > 0
            if move_down:
                move_direction = 2  # Down
            else: # move_up
                move_direction = 0  # Up
        elif self.x_iteration_towards_tile <
abs(self.x_iterations_towards_tile):
            self.x_iteration_towards_tile += 1
            move_left = self.x_iterations_towards_tile < 0
            if move_left:
                move_direction = 3 # Left
            else: # move_right
                move_direction = 1  # Right
        else:
            # FOUND TARGET TILE
            self.reset_moving_toward_clean_tile_variables()
            return
        return move_direction


    def calculate_moves_to_tile(self, pos, target):
        x_distance = target[0] - pos[0]
        y_distance = target[1] - pos[1]
        return x_distance, y_distance


    def reset_moving_toward_clean_tile_variables(self):
        self.moving_toward_clean_title = False
        self.x_iteration_towards_tile = 0
        self.x_iterations_towards_tile = 0
        self.y_iteration_towards_tile = 0
        self.y_iterations_towards_tile = 0
        self.total_iterations_towards_tile =
abs(self.x_iterations_towards_tile) +
abs(self.y_iterations_towards_tile)
        self.is_move_towards_tile_x_first = False
        self.is_move_towards_tile_y_first = False


    def new_pos(self, dir, curr_pos):
        x, y = curr_pos
        if dir == 0:  # Up
            return (x, y - 1)
        elif dir == 1:  # Right
            return (x + 1, y)
        elif dir == 2:  # Down
            return (x, y + 1)
        elif dir == 3:  # Left
            return (x - 1, y)
        else:
            return 'Invalid direction'
    def get_neighbors(current_pos):
        directions = [(0, -1), (1, 0), (0, 1), (-1, 0)]  # up, right,
down, left
        neighbors = []
        for dx, dy in directions:
            next_pos = (current_pos[0] + dx, current_pos[1] + dy)
            neighbors.append(next_pos)
        return neighbors


    def get_random_excluding(input_number):
        if input_number not in [0, 1, 2, 3]:
            raise ValueError("Input number must be 0, 1, 2, or 3")
        numbers = [0, 1, 2, 3]
        numbers.remove(input_number)
        return random.choice(numbers)


    def reset_random_dir(self):
        self.random_dir = {1, 2, 3, 4}
        self.random_dir = set(random.sample(list(self.random_dir),
len(self.random_dir)))
        return None
```