CS 7320
Midterm Coding Assignment - RoboVac
(see MidtrmFiles.zip)

- Use all your knowledge about search to implement a vacuum agent that cleans an entire floor.
- Run the simulator PygameRoboVac.py which calls your code (RoboVac0), passing the current position of the robot and room info.
- Each game cycle, the simulator PygameRoboVac , will call your function , get_next_move. You will be passed the location of your RoboVac as an x,y coordinate (0,0 is upper left corner). Return your choice of directions to move (0=up, 1=right, 2=down, 3=left)
- The code for RoboVac0.py implements an agent that performs a random walk around the room.
- Improve this code so that your agent clean the level 0 room with no furniture
- There are 6 levels (0-5) with rooms of varying degrees of difficulty. Start with Level 0, RoboVac0 (no blocks) and modify the code provided to get your RoboVac to visit all the squares in the room. Add your name and id to the code and save it as RoboVac0.py.
- When satisfied with your code for level 0, make a copy and call it RoboVac1.py. Add code to get around the one block. When satisfied, make a copy and call it RoboVac2.py and repeat. Do this for each level up to RoboVac5.py. Include all the 6 code files in yoru zip file submission.
- Max Grade: Level 5 with 100% clean
- Prepare a table of your results:

Level	Pct Clean	Cycles	File
0			RoboVac0.py
1			RoboVac1.py
2			RoboVac2.py
3			RoboVac3.py
4			RoboVac4.py
5			RoboVac5.py

- Don't panic if you cannot clean the entire room at different levels. Work your way up to level 5. Display your best results for each level in the table. Partial credit will be given. Be sure and document your code so a reader can understand your logic. Points will be given for readability and understandability of your code.

Important!!!

- Copy all the files in the zip file to one directory
- You may need to import the **pygame** package into your Python environment to run Pygame
- The *empty* file labeled: **__init__.py** must be in same directory for Pygame to draw!

Details:

- The file **PygameRoboVac.py** is the simulator that contains the main function that will launch the game. It requires the package ***pygame***
- **Run PygameRoboVac.py** with the default RoboVac0.py. Study this code. It defines a class called RoboVac that is instantiated by the simulator.

- Details of the room (size, blocks) are passed to the RoboVac constructor (__init__) when your instance is created by the PyGame game engine.
- Each game cycle your `get_next_move` method is called with your current position passed as a parameter. If the position remains the same from one call to the next, your Robot is blocked either by a wall or by a block. Figure out how to get around the blocks.
- Simulation will quit at 400 cycles
- Read the code for PygameRoboVac for ideas. Feel free copy and reuse any code. Document any reuse of Pygame code.

Conquering New Levels

- There are 6 games levels, each a bit more complex. Start with game_level = 0 and RoboVac0
- To change the game levels, navigate to the next to last code line in PygameRoboVac where you will see:
  ```
  game_level = 0  # change this to change levels
  ```
- When you move to a new level with a new RoboVacX.py .. file you must tell Pygame where to find your RoboVac class definition.
- On line 13 of the program find:
  ```
     from RoboVac0 import RoboVac
  ```
- change the name `RoboVac0` to the name of your file, e.g.RoboVac1, RoboVac2, .. etc.

Note:

- Each time you run the program, the blocks will be in slightly different positions. Therefore, do not try and solve the problem by relying on fixed positions of the blocks.
- TIP: To clean the entire room, keep track of where the robot has been and try to move to positions not yet visited. Think child nodes. Each square has at most 4 squares that it can move to. Use any technique to solve the problem.
- There is no one approach to the problem and I doubt you can find solutions for the problem online. This is unique to our class.
- To speed or slow down the game, modify the delay_time (line 241 in PyGameRoboVac)

RoboVac0.py

```
'''
create robot vacuum that cleans all the floors of a grid.
main creates an instance of RoboVac (your code) and provides:
- grid size
- loc of robovac
- list of x,y,w,h tuples are instance of rectangular blocks

goal: visit all tiles
exec will : create instance and in game loop call : nextMove()  ??
'''
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,ht)

        # fill in with your info
        self.name = "Zarkon Zeeblebrock"
        self.id = "66666666"


    def get_next_move(self, current_pos):  # called by PyGame code
        # Return a direction for the vacuum to move
        # random walk 0=north # 1=east 2=south 3=west
        return random.choice([0,1,2,3])
```

Submit:

- A **Pdf** with:
  - Your results table (shown above)
  - Your code for RoboVac5 with line numbers and 11 point courier or monospaced font.
- A **zip** file with your all versions of your code: RoboVac0,RoboVac1, 2, 3, etc.

PygameRoboVac.py

```
'''
    RoboVac - clean the room
    note: PyGame needs an empty file __init__.py in directory to draw!
    v. 0.90
'''

import random

import pygame
import sys

# modify RoboVac0 or copy and create your own file
from RoboVac0 import RoboVac

BLACK = (0, 0, 0)
GOLD = (200,0, 0)
WHITE = (200, 200, 200)
GREEN = (0,200, 0)
BROWN = (165, 42, 42)
ORANGE = (255,140,0)
YELLOW = (240, 240, 130)

# load images

RoboVacPic = pygame.image.load("robovac.png")

class Room:
    def __init__(self, level):
        pygame.init()
        self.game_level = level

        # sets up room (size) with blocks and vacuum start position
        # -Robovac is initialized with data structure containing
        #   all room data


        # window  - varies in size
        window_size_list = [360, 390, 420]
        self.window_width = random.choice(window_size_list)
        self.window_height = random.choice(window_size_list)

        # grid max values for width and height
        self.room_blocksize = 30
        self.max_width =  (int)(self.window_width / self.room_blocksize)
        self.max_height = (int) (self.window_height /
                                self.room_blocksize)
        # for grid logic
        self.max_x = (self.window_width / self.room_blocksize) -1
        self.max_y = (self.window_height / self.room_blocksize) -1

        # blocks - number of blocks depends on game_level
        self.block_list = []

        if self.game_level >= 1:
            self.block_list.append((1, 2, 4, 1))

        if self.game_level >= 2:
            self.block_list.append((1, 2, 4, 1))
            self.block_list.append((3, 2, 1, 4))
            self.block_list.append((1, 2, 4, 2))

        if self.game_level >= 3:
            self.block_list.append((1, 2, 4, 1))
            self.block_list.append((6, 6, 4, 1))
            self.block_list.append((9, 6, 1, 3))
            self.block_list.append((6, 8, 4, 1))

        if self.game_level >= 4:
            self.block_list.append((0, 8, 4, 1))

        if self.game_level >= 5:
            self.block_list.append((10, 10, 4,1))


        # vacuum random positioning
        x=0; y=0
        dx = 3; dy = 3   #dist from edge

        intersect = True
        while intersect:
            x = random.randrange(dx, self.max_x)
            y = random.randrange(dy, self.max_y)
            intersect = self.does_pos_intersect_blocks((x, y))
        self.vac_pos = (x,y)  # starting location for vacuum

        # define sets with positions as tuples; useful utilities
        self.clean_set = set()
        self.clean_set.add(self.vac_pos)

        # create set with all tiles
        self.free_tiles_set = set()
        for x in range(self.max_width):
            for y in range(self.max_height):
                self.free_tiles_set.add( (x,y))

        # BLOCKS
        # create set of all block positions from block list
        self.block_tiles_set = set()

        for b in self.block_list:
            for x in range(b[0], b[0] + b[2]):
                for y in range(b[1], b[1] + b[3]):
                    self.block_tiles_set.add((x, y))

        self.free_tiles_set = self.free_tiles_set - self.block_tiles_set

        # easily get max number of tiles that need cleaning
        self.max_tiles = len(self.free_tiles_set)

        # other  for display
        self.font = pygame.font.SysFont('Arial', 20)
```

```

    def get_room_config(self):
        '''
        Returns LIST with all the info RoboVac needs;
              passed to RoboVac constructor
        [ (room_width, room_height), (vac_x, vac_y) [list-of-blocks] ]
         note: list-of-blocks is list of tuples (x,y,width, height)
        '''
        room_config_list = [(self.max_width, self.max_height), \
                                  self.vac_pos, \
                                  self.block_list
                            ]
        return room_config_list

    # Utility Methods ------
    def add_clean_pos (self, xytuple):
        self.clean_set.add(xytuple)

    def rect_intersect(self, pos, rect):
        rx, ry, width, height = rect
        x,y = pos
        is_intersect =   x >= rx and x < (rx + width) and \
           y >= ry and y < (ry + height)
        return is_intersect

    def does_pos_intersect_blocks(self, pos):
        #  check all blocks
        for rect in self.block_list:
            if self.rect_intersect(pos, rect):
                return True
        return False

    def is_ok_next_pos(self, xytuple):
        x,y = xytuple
        if x < 0 & x > self.max_x & \
            y < 0 & y > self.max_y:
            return False
        # check intersect with blocks
        for rect in self.block_list:
            rx,ry,width,height = rect
            if x > rx & x < (rx+ width) & y > ry & y < (ry + height):
                return False
        return True

    def __str__(self):
        return f"cpos={self.vac_pos},max_x={self.max_x} \
        max_y={self.max_y}, window:({self.window_width},
{self.window_height}) \
         blocksize={self.room_blocksize}  clean={self.clean_set}"

def get_date_time():
    import datetime
    today = datetime.date.today()
    hour  = datetime.datetime.now().hour
    min   = datetime.datetime.now().minute
    hour_str = f"{hour:02d}"
    min_str  = f"{min:02d}"
    return f"{today} {hour_str}:{min_str}"

#  Utility Functions for Drawing (PyGame) -----------------------
def draw_tile(room,x,y):
    x_draw = x * room.room_blocksize
    y_draw = y * room.room_blocksize
    rect = pygame.Rect(x_draw, y_draw, room.room_blocksize,
room.room_blocksize)
    # draw rect
    pygame.draw.rect(SCREEN, GREEN, rect, 0)  # 0 means fill!!
    pygame.draw.rect(SCREEN, WHITE, rect, 1)

def draw_all_tiles(room):
    for r_tuple in room.clean_set:
        x,y  = r_tuple
        draw_tile(room, x, y)
    # draw vacuum
    draw_vac(room)

def draw_blocks(room) :

    for rect in room.block_list:
        x,y,w,h = rect
        x_draw = x * room.room_blocksize
        y_draw = y * room.room_blocksize
        rect = pygame.Rect(x_draw, y_draw, room.room_blocksize*w,
                           room.room_blocksize*h)
        # draw rect
        pygame.draw.rect(SCREEN, ORANGE, rect, 0)  # 0 means fill!!

def draw_vac(room):
    # draw RoboVac at it's current location
    global SCREEN, RoboVacPic
    global GOLD
    x,y = room.vac_pos
    blocksize = room.room_blocksize
    SCREEN.blit(RoboVacPic,(x*blocksize, y*blocksize) )
    pygame.display.flip()

def drawGrid(room):
    # draw basic room grid..
    for x in range(0, room.window_width, room.room_blocksize):
        for y in range(0, room.window_height, room.room_blocksize):
            rect = pygame.Rect(x, y, room.room_blocksize, \
                                     room.room_blocksize)
            pygame.draw.rect(SCREEN, WHITE, rect, 1)

def main(game_level):
    global SCREEN

    # max # game cycles allowed robot
    max_cycles = 400

    # create the room - pass in the level
    room = Room(game_level)

    # create the Robot Vacuum AI : pass in configuration
    config_list = room.get_room_config()
    robo_vac = RoboVac(config_list)  # this creates your RoboVac !

    # set up the screen display ----------------
    SCREEN = pygame.display.set_mode((room.window_width, \
                                      room.window_height))
    SCREEN.fill(BLACK)

    drawGrid(room)  # draws grid - white outlines; black bkgnd
    draw_vac(room)  # draw vac at initial random pos
    draw_blocks(room)

    pygame.display.update()

    ## CONTROL GAME SPEED   ** OK to change  **
    delay_time = 200
    # game delay in milliseconds between cycles

    move_count = 0  # track number of moves

    # GAME LOOP ---------
    while True:
        if (move_count % 50)  == 0:
            print (f"Move Count: {move_count}")

        # check if done..
        if len(room.clean_set) == room.max_tiles  or \
                move_count > max_cycles:
            result_str = " SUCCESS!"
            if move_count > max_cycles:
                result_str = " OUT OF TIME"
            print(

                f"--------------------------------------\n"
                f"RESULTS ***** {result_str}\n"
                  f"{robo_vac.name} ID={robo_vac.id} {get_date_time()}"
                  f"\nLevel: {room.game_level}  Coverage: "
                  f"{((len(room.clean_set)/room.max_tiles)*100):.1f}%\n"
                  f"Cycles: {move_count} Tiles Cleaned: "
                  f"{len(room.clean_set)} Max Tiles: {room.max_tiles}"
                  f"Total Tiles {room.max_tiles}\n"
                  f"Efficiency: {(room.max_tiles/move_count):.2f}"
                  )
            # results to logfile
            result_str = f"{get_date_time()} {robo_vac.id}" \
                         f" {robo_vac.name} " \
              f" L{room.game_level} " \
              f"Coverage:{((len(room.clean_set)/room.max_tiles)):.2f} " \
              f"Eff:{(room.max_tiles/move_count):.2f}\n"
            print (result_str)
            f = open("log.txt", "a")
            f.write(result_str)
            f.close()


            pygame.quit()
            sys.exit()
        else: # play game  -------------------------
            move_count += 1
            pygame.display.update()

            # CALL ROBO VAC --Returns Direction based on location
            dir = robo_vac.get_next_move(room.vac_pos)
            #  ###################################################

            # Determine if direction results in legal move
            # IF YES, update robot position, else pos remains same

            x,y = room.vac_pos  # current position
            # adjust based on direction only if new pos inside room
            if dir == 0:
                if y > 0:
                    y = y - 1
            elif dir == 1:
                if x < room.max_x:
                    x = x + 1
            elif dir == 2:
                if y < room.max_y:
                    y =y+1
            elif dir == 3:
                if x > 0:
                    x = x-1

            if (x,y) == room.vac_pos:   # tried to go beyond room
                print (f"dir={dir}  BLOCKED: WALL")
            elif (x,y) in room.block_tiles_set:
                print (f"dir={dir} BLOCKED: FURNITURE")
            else:
                room.vac_pos = (x,y)   # update vacuum position
                room.add_clean_pos(room.vac_pos) # track new clean tile
                draw_all_tiles(room)

            # draw vacuum
            draw_vac(room)

            pygame.display.flip()
            pygame.display.update()

            # CONTROLS GAME SPEED
            pygame.time.delay(delay_time)

            # REQUIRED for PyGame - close window,stop game
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()


    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
```

```python

if __name__ == '__main__':
    ''' set your game level -----
            0 = no blocks
            1 = 1 block
            2 = 4 blocks
            3 = 8 blocks
            4 = 9 blocks
            5 = 10 blocks
    '''


    game_level = 0  # Change this as you move from easy to complex

    # calls main with the game level & runs the simulation
    main(game_level)
```