```python
# Path calculator for Waterloo Engineering Expeller of Dominoes

# Andor Siegers

# v1.2

import math
import sys
import pygame
from pygame.locals import *

class Point:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f'({self.x}, {self.y})'

class Instr:
    def __init__(self, if_ang, val):
        self.if_ang = if_ang
        self.val = val

    def __str__(self):
        return f'{self.if_ang}, {self.val}'

# Finds if 2 given line segments intersect or not
# From: https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/

# Given three collinear points p, q, r, the function checks if
# point q lies on line segment 'pr'
def onSegment(p, q, r):
        if ( (q.x <= max(p.x, r.x)) and (q.x >= min(p.x, r.x)) and
                (q.y <= max(p.y, r.y)) and (q.y >= min(p.y, r.y))):
                return True
        return False

def orientation(p, q, r):
        # to find the orientation of an ordered triplet (p,q,r)
        # function returns the following values:
        # 0 : Collinear points
        # 1 : Clockwise points
        # 2 : Counterclockwise

        # See https://www.geeksforgeeks.org/orientation-3-ordered-points/amp/
        # for details of below formula.

        val = (float(q.y - p.y) * (r.x - q.x)) - (float(q.x - p.x) * (r.y - q.y))
        if (val > 0):

                # Clockwise orientation
                return 1
        elif (val < 0):

                # Counterclockwise orientation
                return 2
        else:

                # Collinear orientation
                return 0

# returns true if the line segment 'p1q1' and 'p2q2' intersect
def doIntersect(p1,q1,p2,q2):

        # Find the 4 orientations required for
        # the general and special cases
        o1 = orientation(p1, q1, p2)
        o2 = orientation(p1, q1, q2)
        o3 = orientation(p2, q2, p1)
        o4 = orientation(p2, q2, q1)

        # General case
        if ((o1 != o2) and (o3 != o4)):
                return True

        # Special Cases

        # p1 , q1 and p2 are collinear and p2 lies on segment p1q1
        if ((o1 == 0) and onSegment(p1, p2, q1)):
                return True

        # p1 , q1 and q2 are collinear and q2 lies on segment p1q1
        if ((o2 == 0) and onSegment(p1, q2, q1)):
                return True

        # p2 , q2 and p1 are collinear and p1 lies on segment p2q2
        if ((o3 == 0) and onSegment(p2, p1, q2)):
                return True

        # p2 , q2 and q1 are collinear and q1 lies on segment p2q2
        if ((o4 == 0) and onSegment(p2, q1, q2)):
                return True

        # If none of the cases
        return False

# returns dot product
def dot(vA, vB):
    return vA[0]*vB[0]+vA[1]*vB[1]

# returns line length
def calcLength(p1, p2):
    return math.sqrt((p1.x-p2.x)**2 + (p1.y-p2.y)**2)

# get angle between two vectors
def getAngle(p1,p2,p3,p4):
    # https://stackoverflow.com/questions/28260962/calculating-angles-between-line-segments-
python-with-math-atan2

    # Get nicer vector form
    lineA = ((p1.x,p1.y),(p2.x,p2.y))
    lineB = ((p3.x,p3.y),(p4.x,p4.y))
    vA = [(lineA[0][0]-lineA[1][0]), (lineA[0][1]-lineA[1][1])]
```

```python
    vB = [(lineB[0][0]-lineB[1][0]), (lineB[0][1]-lineB[1][1])]
    # Get dot prod
    dot_prod = dot(vA, vB)
    # Get magnitudes
    magA = dot(vA, vA)**0.5
    magB = dot(vB, vB)**0.5
    # Get cosine value
    cos_ = dot_prod/magA/magB
    # Get angle in radians and then convert to degrees
    angle = math.acos(dot_prod/magB/magA)
    # Basically doing angle <- angle mod 360
    ang_deg = math.degrees(angle)%360
    return ang_deg

# calculate the center point of a circle tangent to 2 lines forming an angle
def calcCenterPoint(new_point, rad, coords):
    # from:
    # https://stackoverflow.com/questions/51223685/create-circle-tangent-to-two-lines-with-radius-r-geometry

    p1 = coords[len(coords) - 2]
    p2 = coords[len(coords) - 1]
    p3 = new_point

    le1 = math.sqrt((p2.x-p1.x)**2 + (p2.y-p1.y)**2) # length of A1-B1 segment
    v1x = (p2.x-p1.x) / le1
    v1y = (p2.y-p1.y) / le1

    le2 = math.sqrt((p3.x-p2.x)**2 + (p3.y-p2.y)**2) # length of A1-B1 segment
    v2x = (p3.x-p2.x) / le2
    v2y = (p3.y-p2.y) / le2

    R = rad
    px1 = p1.x - v1y*R
    py1 = p1.y + v1x*R
    px2 = p2.x - v2y*R
    py2 = p2.y + v2x*R

    px1u = p1.x + v1y*R
    py1u = p1.y - v1x*R
    px2u = p2.x + v2y*R
    py2u = p2.y - v2x*R

    den = v1x*v2y - v2x*v1y

    k1 = (v2y*(px2-px1) - v2x*(py2-py1)) / den
    # k2 = (v1y*(px2-px1) - v1x*(py2-py1)) / den

    k1u = (v2y*(px2u-px1u) - v2x*(py2u-py1u)) / den
    # k2u = (v1y*(px2u-px1u) - v1x*(py2u-py1u)) / den

    tx1 = p1.x + k1*v1x
    ty1 = p1.y + k1*v1y
    # tx2 = p2.x + k2*v2x
    # ty2 = p2.y + k2*v2x

    if(onSegment(p1,Point(tx1,ty1),p2)):
        cx =  px1 + k1*v1x
        cy =  py1 + k1*v1y
        left_turn = False
    else:
        cx =  px1u + k1u*v1x
        cy =  py1u + k1u*v1y
        left_turn = True

    # subtracts length taken from the arc from line lengths
    len_to_sub = calcLength(p2, Point(tx1,ty1))

    return Point(cx,cy), left_turn, len_to_sub

def main():
    # pygame specific instructions from:
    # https://stackoverflow.com/questions/19780411/pygame-drawing-a-rectangle
    pygame.init()

    DISPLAY = pygame.display.set_mode((700,500),0,32)

    WHITE = (255,255,255)
    BLUE = (0,0,255)
    prev_point = Point(0,0)
    prev_len_to_sub = 0
    ang1 = 0
    line_count = -1
    ANGLE_TOLERANCE = 20
    RADIUS_IN_CM = 20
    PIXELS_PER_CM = 5
    RADIUS_IN_PIXELS = RADIUS_IN_CM*PIXELS_PER_CM
    coords = [] # stores coordinates as point values
    instructs = [] # stores instructions for robot

    DISPLAY.fill(WHITE)

    while True:

        for event in pygame.event.get():
            if (event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE) or event.type
== QUIT:
                # before program ends
                file = open('instr.txt', 'w')
                try:
                    # save instructions to file
                    file.write(str(len(instructs)) + "\n")

                    for i in range(len(instructs)):
                        file.write(str((int)(instructs[i].if_ang)) + " " + str((int)
(instructs[i].val)))
                        if i != len(instructs)-1:
                            file.write("\n")

                except:
                    print("Unable to open file")

                file.close()
                pygame.quit()
                sys.exit()
```

```python

            if event.type == pygame.MOUSEBUTTONDOWN:
                # when mouse is pressed
                x,y = pygame.mouse.get_pos()
                new_point = Point(x,y)
                # check for double click and continue if it is to avoid instructions with length 0
                if(new_point.x == prev_point.x and new_point.y == prev_point.y):
                    continue
                legal_line = True

                # new line
                p1 = Point(prev_point.x, prev_point.y)
                q1 = Point(new_point.x, new_point.y)

                length = calcLength(new_point, prev_point)

                if line_count == -1:
                    # calculate very first angle to turn
                    angle = math.degrees(math.atan2(new_point.y,new_point.x))
                    ang1 = angle

                elif line_count == 0:
                    # calculates second angle to turn
                    angle = 180-getAngle(new_point, prev_point, Point(0,0), prev_point)

                    # check if angle is negative
                    ang2 = math.degrees(math.atan2(new_point.y,new_point.x))
                    if ang2 < ang1:
                        angle = -angle

                else:
                    # check if new line lintersects with any other line
                    angle = getAngle(new_point, prev_point, coords[line_count-1], prev_point)

                    # check if angle between old and new line is more than 20 degrees
                    if angle < ANGLE_TOLERANCE:
                        legal_line = False
                    for i in range(line_count-1):
                        # temp line
                        p2 = coords[i]
                        q2 = coords[i+1]
                        if(doIntersect(p1, q1, p2, q2)):
                            legal_line = False

                if legal_line:
                    # if all checks are passed
                    if line_count != -1:
                        # draws line to visualize path
                        pygame.draw.aaline(DISPLAY, BLUE, (prev_point.x, prev_point.y), (new_point.x, new_point.y))

                        if line_count >= 1:
                            angle = 180-angle
                            # calculates turn direction, while getting data to draw circle(representing turning arc)
                            centCoord, left_turn, len_to_sub = calcCenterPoint(new_point, RADIUS_IN_PIXELS, coords)

                            # adjust angle depending on turn direction
                            if left_turn:
                                angle = -angle

                            # subtract len_to_sub from overall length
                            length -= (len_to_sub + prev_len_to_sub)

                            # subtracts length from previous instruction to accomodate new arc
                            if line_count == 1 and not instructs[len(instructs) - 1].if_ang:
                                instructs[len(instructs) - 1].val -= len_to_sub

                            prev_len_to_sub = len_to_sub

                            # draw circle representing robot turning arc
                            rect = Rect(centCoord.x-RADIUS_IN_PIXELS, centCoord.y-RADIUS_IN_PIXELS, RADIUS_IN_PIXELS*2, RADIUS_IN_PIXELS*2)
                            pygame.draw.arc(DISPLAY,BLUE,rect,0,2*math.pi, 1)

                        # update display
                        pygame.display.flip()
                    # add new coordinate to point list
                    coords.append(new_point)
                    prev_point = new_point
                    # add new instruction to point list
                    instructs.append(Instr(True,angle))

                    if length > 0:
                        instructs.append(Instr(False, length))

                    line_count += 1

        #  update display
        pygame.display.flip()

main()

# Resources:
# http://www.pygame.org/docs/ref/draw.html#pygame.draw.line
# https://www.geeksforgeeks.org/with-statement-in-python/
# https://www.pythontutorial.net/python-basics/python-write-text-file/
# https://www.w3schools.com/python/ref_list_extend.asp
# https://stackoverflow.com/questions/19780411/pygame-drawing-a-rectangle
# https://stackoverflow.com/questions/3838329/how-can-i-check-if-two-segments-intersect
# https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/
# https://stackoverflow.com/questions/28260962/calculating-angles-between-line-segments-python-with-math-atan2
```