1 # Path calculator for Waterloo Engineering Expeller of Dominoes 2

3 # Andor Siegers 4

5 # v1.2

6

1. **import** math
2. **import** sys
3. **import** pygame
4. **from** pygame.locals **import** \* 11

12 **class Point**:

13

1. **def init** (self, x, y):
2. self.x = x
3. self.y = y 17
4. **def str** (self):
5. **return** f'({self.x}, {self.y})' 20

# class Instr:

1. **def init** (self, if\_ang, val):
2. self.if\_ang = if\_ang
3. self.val = val 25
4. **def str** (self):
5. **return** f'{self.if\_ang}, {self.val}' 28
6. # Finds if 2 given line segments intersect or not
7. # From: https://[www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/](http://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/) 31
8. # Given three collinear points p, q, r, the function checks if
9. # point q lies on line segment 'pr'
10. **def onSegment**(p, q, r):
11. **if** ( (q.x <= max(p.x, r.x)) **and** (q.x >= min(p.x, r.x)) **and**

36 (q.y <= max(p.y, r.y)) **and** (q.y >= min(p.y, r.y))):

1. **return** True
2. **return** False 39
3. **def orientation**(p, q, r):
4. # to find the orientation of an ordered triplet (p,q,r)
5. # function returns the following values:
6. # 0 : Collinear points
7. # 1 : Clockwise points
8. # 2 : Counterclockwise 46
9. # See https://[www.geeksforgeeks.org/orientation-3-ordered-points/amp/](http://www.geeksforgeeks.org/orientation-3-ordered-points/amp/)
10. # for details of below formula. 49
11. val = (float(q.y - p.y) \* (r.x - q.x)) - (float(q.x - p.x) \* (r.y - q.y))
12. **if** (val > 0):

52

1. # Clockwise orientation

# return 1

1. **elif** (val < 0):

56

57 # Counterclockwise orientation

58

59 **else**:

60

61

62

63

**return** 2

# Collinear orientation

# return 0

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

66

1. # Find the 4 orientations required for
2. # the general and special cases
3. o1 = orientation(p1, q1, p2)
4. o2 = orientation(p1, q1, q2)
5. o3 = orientation(p2, q2, p1)
6. o4 = orientation(p2, q2, q1) 73

74 # General case

75 **if** ((o1 != o2) **and** (o3 != o4)):

76 **return** True

77

78 # Special Cases 79

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

83

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

87

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

91

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

95

1. # If none of the cases
2. **return** False 98
3. # returns dot product
4. **def dot**(vA, vB):

101 **return** vA[0]\*vB[0]+vA[1]\*vB[1] 102

1. # returns line length
2. **def calcLength**(p1, p2):

105 **return** math.sqrt((p1.x-p2.x)\*\*2 + (p1.y-p2.y)\*\*2) 106

1. # get angle between two vectors
2. **def getAngle**(p1,p2,p3,p4):
3. # https://stackoverflow.com/questions/28260962/calculating-angles-between-line-segments- python-with-math-atan2
4. # Get nicer vector form

112 lineA = ((p1.x,p1.y),(p2.x,p2.y))

113 lineB = ((p3.x,p3.y),(p4.x,p4.y))

114 vA = [(lineA[0][0]-lineA[1][0]), (lineA[0][1]-lineA[1][1])]

115 vB = [(lineB[0][0]-lineB[1][0]), (lineB[0][1]-lineB[1][1])]

1. # Get dot prod
2. dot\_prod = dot(vA, vB)
3. # Get magnitudes
4. magA = dot(vA, vA)\*\*0.5
5. magB = dot(vB, vB)\*\*0.5
6. # Get cosine value
7. cos\_ = dot\_prod/magA/magB
8. # Get angle in radians and then convert to degrees
9. angle = math.acos(dot\_prod/magB/magA)
10. # Basically doing angle <- angle mod 360
11. ang\_deg = math.degrees(angle)%360
12. **return** ang\_deg 128
13. # calculate the center point of a circle tangent to 2 lines forming an angle
14. **def calcCenterPoint**(new\_point, rad, coords):
15. # from:
16. # https://stackoverflow.com/questions/51223685/create-circle-tangent-to-two-lines-with- radius-r-geometry
17. p1 = coords[len(coords) - 2]
18. p2 = coords[len(coords) - 1]
19. p3 = new\_point 137

138 le1 = math.sqrt((p2.x-p1.x)\*\*2 + (p2.y-p1.y)\*\*2) # length of A1-B1 segment 139 v1x = (p2.x-p1.x) / le1

140 v1y = (p2.y-p1.y) / le1 141

142 le2 = math.sqrt((p3.x-p2.x)\*\*2 + (p3.y-p2.y)\*\*2) # length of A1-B1 segment 143 v2x = (p3.x-p2.x) / le2

144 v2y = (p3.y-p2.y) / le2 145

1. R = rad
2. px1 = p1.x - v1y\*R
3. py1 = p1.y + v1x\*R
4. px2 = p2.x - v2y\*R
5. py2 = p2.y + v2x\*R 151
6. px1u = p1.x + v1y\*R
7. py1u = p1.y - v1x\*R
8. px2u = p2.x + v2y\*R
9. py2u = p2.y - v2x\*R 156

157 den = v1x\*v2y - v2x\*v1y 158

159 k1 = (v2y\*(px2-px1) - v2x\*(py2-py1)) / den

160 # k2 = (v1y\*(px2-px1) - v1x\*(py2-py1)) / den 161

1. k1u = (v2y\*(px2u-px1u) - v2x\*(py2u-py1u)) / den
2. # k2u = (v1y\*(px2u-px1u) - v1x\*(py2u-py1u)) / den 164

165 tx1 = p1.x + k1\*v1x

166 ty1 = p1.y + k1\*v1y

167 # tx2 = p2.x + k2\*v2x

168 # ty2 = p2.y + k2\*v2x 169

170 **if**(onSegment(p1,Point(tx1,ty1),p2)):

|  |  |  |
| --- | --- | --- |
| 171  172  173  174 | cx = px1 + k1\*v1x  cy = py1 + k1\*v1y left\_turn = False  **else**: | |
| 175 |  | cx = px1u + k1u\*v1x |
| 176 |  | cy = py1u + k1u\*v1y |
| 177 |  | left\_turn = True |
| 178 |  |  |
| 179 |  | # subtracts length taken from the arc from line lengths |
| 180 |  | len\_to\_sub = calcLength(p2, Point(tx1,ty1)) |
| 181 |  |  |
| 182 |  | **return** Point(cx,cy), left\_turn, len\_to\_sub |
| 183 |  |  |
| 184 | **def** | **main**(): |
| 185 |  | # pygame specific instructions from: |
| 186 |  | # https://stackoverflow.com/questions/19780411/pygame-drawing-a-rectangle |
| 187 |  | pygame.init() |
| 188 |  |  |
| 189 |  | DISPLAY = pygame.display.set\_mode((700,500),0,32) |
| 190 |  |  |
| 191 |  | WHITE = (255,255,255) |
| 192 |  | BLUE = (0,0,255) |
| 193 |  | prev\_point = Point(0,0) |
| 194 |  | prev\_len\_to\_sub = 0 |
| 195 |  | ang1 = 0 |
| 196 |  | line\_count = -1 |
| 197 |  | ANGLE\_TOLERANCE = 20 |
| 198 |  | RADIUS\_IN\_CM = 20 |
| 199 |  | PIXELS\_PER\_CM = 5 |
| 200 |  | RADIUS\_IN\_PIXELS = RADIUS\_IN\_CM\*PIXELS\_PER\_CM |
| 201 |  | coords = [] # stores coordinates as point values |
| 202 |  | instructs = [] # stores instructions for robot |
| 203 |  |  |
| 204 |  | DISPLAY.fill(WHITE) |
| 205 |  |  |
| 206 |  | **while** True: |
| 207 |  |  |
| 208 |  | **for** event **in** pygame.event.get(): |

209

210

211

212

213

214

215

== QUIT:

**if** (event.type == pygame.KEYDOWN **and** event.key == pygame.K\_ESCAPE) **or** event.type

# before program ends

file = open('instr.txt', 'w')

**try**:

# save instructions to file file.write(str(len(instructs)) + "\n")

|  |  |  |
| --- | --- | --- |
| 216 | **for** | i **in** range(len(instructs)): |
| 217 | (instructs[i].val))) | file.write(str((int)(instructs[i].if\_ang)) + " " + str((int) |
| 218 |  | **if** i != len(instructs)-1: |
| 219 |  | file.write("\n") |
| 220 |  |  |
| 221 | **except**: |  |
| 222 | print("Unable to open file") | |
| 223 |  | |
| 224 | file.close() | |
| 225 | pygame.quit() | |
| 226 | sys.exit() | |

|  |  |  |
| --- | --- | --- |
| 228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246 | **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 | |
| 247 |  |  |
| 248 |  | **elif** line\_count == 0: |
| 249 |  | # calculates second angle to turn |
| 250 |  | angle = 180-getAngle(new\_point, prev\_point, Point(0,0), prev\_point) |
| 252 |  | # check if angle is negative |
| 253 |  | ang2 = math.degrees(math.atan2(new\_point.y,new\_point.x)) |
| 254 |  | **if** ang2 < ang1: |
| 255 |  | angle = -angle |
| 256 |  |  |
| 257 |  | **else**: |
| 258 |  | # check if new line lintersects with any other line |
| 259 |  | angle = getAngle(new\_point, prev\_point, coords[line\_count-1], prev\_point) |
| 260 |  |  |
| 261 |  | # check if angle between old and new line is more than 20 degrees |
| 262 |  | **if** angle < ANGLE\_TOLERANCE: |
| 263 |  | legal\_line = False |
| 264 |  | **for** i **in** range(line\_count-1): |
| 265 |  | # temp line |
| 266 |  | p2 = coords[i] |
| 267 |  | q2 = coords[i+1] |
| 268 |  | **if**(doIntersect(p1, q1, p2, q2)): |
| 269 |  | legal\_line = False |
| 270 |  |  |
| 271 |  | **if** legal\_line: |
| 272 |  | # if all checks are passed |
| 273 |  | **if** line\_count != -1: |
| 274 |  | # draws line to visualize path |
| 275 | (new\_point.x, | pygame.draw.aaline(DISPLAY, BLUE, (prev\_point.x, prev\_point.y), new\_point.y)) |
| 276 |  |  |
| 277 |  | **if** line\_count >= 1: |
| 278 |  | angle = 180-angle |
| 279 |  | # calculates turn direction, while getting data to draw |
| 280 | circle(representing turning arc)  centCoord, left\_turn, len\_to\_sub = calcCenterPoint(new\_point,  RADIUS\_IN\_PIXELS, coords) | |

251

|  |  |  |  |
| --- | --- | --- | --- |
| 282 |  | # adjust angle depending on turn direction | |
| 283 |  | **if** left\_turn: | |
| 284 |  | angle = -angle | |
| 285 |  |  | |
| 286 |  | # subtract len\_to\_sub from overall length | |
| 287 |  | length -= (len\_to\_sub + prev\_len\_to\_sub) | |
| 288 |  |  | |
| 289 | arc | # subtracts length from previous instruction to accomodate new | |
| 290 |  | **if** line\_count == 1 **and not** instructs[len(instructs) - 1].if\_ang: | |
| 291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308 | 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)) | | |
| 309 |  | |  |
| 310 |  | | line\_count += 1 |
| 311 |  | |  |
| 312 | # update | | display |

313

314

315

316

main()

pygame.display.flip()

1. # Resources:
2. # <http://www.pygame.org/docs/ref/draw.html#pygame.draw.line>
3. # https://[www.geeksforgeeks.org/with-statement-in-python/](http://www.geeksforgeeks.org/with-statement-in-python/)
4. # https://[www.pythontutorial.net/python-basics/python-write-text-file/](http://www.pythontutorial.net/python-basics/python-write-text-file/)
5. # https://[www.w3schools.com/python/ref\_list\_extend.asp](http://www.w3schools.com/python/ref_list_extend.asp)
6. # https://stackoverflow.com/questions/19780411/pygame-drawing-a-rectangle
7. # https://stackoverflow.com/questions/3838329/how-can-i-check-if-two-segments-intersect
8. # https://[www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/](http://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/)
9. # https://stackoverflow.com/questions/28260962/calculating-angles-between-line-segments- python-with-math-atan2