

```

1 # Authored By: Jayant Sogikar
2 import tkinter as tk
3 import math
4 import random
5
6
7 root = tk.Tk()
8
9
10 # root.geometry("300x300-1000-1000")
11 # root.state('zoomed')
12
13
14 root.title("Lines In Circle")
15
16 width = 500
17 height = 500
18 win = tk.Canvas(root,width=width,height=height)
19 win.pack()
20
21 # Experiment with flag at line 66, set it to any value between 0 and 2, can be random too, i like setting it to 0
22
23 class CustomLine:
24     def __init__(self, fromPoint, toPoint,theta, angle, dist = float('inf')):
25         self.fromPoint = fromPoint
26         self.toPoint = toPoint
27         self.theta = theta
28         self.angle = angle
29         self.dist = dist
30
31
32 # Return true if line segments AB and CD intersect
33 def intersect(self,line1,line2):
34     A,B,C,D = line1[0],line1[1],line2[0],line2[1]
35     def ccw(A,B,C):
36         return (C[1]-A[1]) * (B[0]-A[0]) > (B[1]-A[1]) * (C[0]-A[0])
37     return ccw(A,C,D) != ccw(B,C,D) and ccw(A,B,C) != ccw(A,B,D)
38
39
40 def slope(self,P1, P2):
41     # dy/dx
42     # (y2 - y1) / (x2 - x1)
43     if P2[0] - P1[0] == 0: return 0
44     return(P2[1] - P1[1]) / (P2[0] - P1[0])
45
46 def y_intercept(self,P1, slope):
47     # y = mx + b
48     # b = y - mx
49     # b = P1[1] - slope * P1[0]
50     return P1[1] - slope * P1[0]
51 # Return the point of intersection
52 def line_intersect(self,m1, b1, m2, b2):
53     if m1 == m2:
54         print ("These lines are parallel!!!")
55         return None
56     # y = mx + b
57     # Set both lines equal to find the intersection point in the x direction
58     # m1 * x + b1 = m2 * x + b2
59     # m1 * x - m2 * x = b2 - b1
60     # x * (m1 - m2) = b2 - b1
61     # x = (b2 - b1) / (m1 - m2)
62     x = (b2 - b1) / (m1 - m2)
63     # Now solve for y -- use either line, because they are equal here
64     # y = mx + b
65     y = m1 * x + b1
66     return x,y
67
68
69 def update_ToPoint(self, radius):
70     a = self.toPoint
71
72     flag = 1
73     flag = random.randint(0,1)
74
75
76     x = a[0]
77     y = a[1]
78
79     if flag == 0: x += random.randint(radius - 50 , radius )
80     elif flag == 1: y += random.randint(radius - 50 , radius )
81     # else:
82     #     x += new_dir
83     #     y += new_dir
84
85
86
87
88     self.toPoint = [x,y]
89
90
91
92
93 def getDetails(self):
94     return (self.fromPoint,self.toPoint)
95
96 def draw():
97     # Set the radius
98     radius = min(width,height) // 2 - 50
99     # Create a circle
100     win.create_oval((width//2) - radius,(height//2) - radius, (width//2) + radius,(height//2) + radius)
101
102     # Specify lines to be drawn
103     nLines = random.randint(300,600)
104     angle = (2 * math.pi) / float(nLines)
105     points = [x for x in range(nLines)]
106     random.shuffle(points)
107     lineAngles = points
108     allLines = []
109     for theta in lineAngles:
110         # Point on the circle at an angle
111         pos1 = [(math.cos(theta * angle) * (radius)) + width // 2, (math.sin(theta* angle) * (radius) ) + height // 2]
112         # Point at the center of the circle
113         d = [(width // 2),(height // 2) ]
114         # Create an object with the above vals
115         line = CustomLine(pos1,d,theta, angle)
116         # Radomly change the point
117         line.update_ToPoint(radius)
118         # Append the object to a list
119         allLines.append(line)
120     drewLines = []
121
122     def checkIfOutCircle(x,center_x,y,center_y,radius):
123         return ((x - center_x)**2) + ((y - center_y)**2) > radius**2
124     # We now have to check if the line to be drawn intersects any other pre-drawn lines or not
125     # If it does, then we reset to the toPoint in it to the point of intersection
126     for i in range(len(allLines)):
127
128         line1 = [[allLines[i].fromPoint[0],allLines[i].fromPoint[1]],[allLines[i].toPoint[0],allLines[i].toPoint[1]]]
129         for j in range(len(drewLines)):
130             line2 = [[drewLines[j].fromPoint[0],drewLines[j].fromPoint[1]],[drewLines[j].toPoint[0],drewLines[j].toPoint[1]]]
131             if allLines[i].intersect(line1,line2):
132                 slope1 = allLines[i].slope(line1[0],line1[1])
133                 slope2 = allLines[j].slope(line2[0],line2[1])
134                 yA = allLines[i].y_intercept(line1[0],slope1)
135                 yB = allLines[i].y_intercept(line2[0],slope2)
136                 a = allLines[i].line_intersect(slope1,yA,slope2,yB)
137                 if a:
138                     x , y = a[0], a[1]
139                     dist = (((line1[0][0] - x)**2) + ((line1[0][1] - y)**2)
140                     if dist < allLines[i].dist:
141                         allLines[i].toPoint = [x,y]
142                         allLines[i].dist = dist

```

```

143         if checkIfOutCircle(allLines[i].fromPoint[0], width // 2, allLines[i].fromPoint[1], height // 2, radius) or checkIfOutCircle(allLines[i].toPoint[0], width // 2, allLines[i].toPoint[1], height // 2, radius):
144             drawLines.append(allLines[i])
145             s,d = allLines[i].getDetails()
146             win.create_line(s[0],s[1],d[0],d[1])
147             win.update()
148
149 draw()
150
151 # Can save using this
152
153 win.pack()
154 win.update()
155 win.postscript(file="file_name.eps", colormode='color')
156 img = Image.open("file_name.eps")
157 img.save('filename.png')
158
159 root.mainloop()

```

```

1 // Authored By: Jayant Sogikar
2 #include<omp.h>
3 #include<stdio.h>
4 #include<stdlib.h>
5
6 int main(){
7
8     // Init a parallel section
9     #pragma omp parallel
10    {
11        // Finding thread number
12        int t = omp_get_thread_num();
13
14        // output
15        printf("Hello world from thread number %d \n",t);
16    }
17
18    return 0;
19 }
20
21 // gcc hw.c -fopenmp
22 // ./a.out

```

```

1 # Authored By: Jayant Sogikar
2 from collections import defaultdict
3
4
5 def solution(map):
6
7     fromA = defaultdict(lambda : float('inf'))
8     fromB = defaultdict(lambda : float('inf'))
9
10    r, c = len(map) , len(map[0])
11    fromA[(0,0)] = 1
12
13    q = [(0,0,1)]
14
15    while q:
16        x, y, score = q.pop(0)
17
18
19
20        for (i,j) in [(0,1), (1,0),(-1,0), (0, -1)]:
21            if 0 <= x + i < r and 0 <= y + j < c and fromA[(x + i,y + j)] == float('inf'):
22
23                fromA[(x + i,y + j)] = score + 1
24
25                if map[x + i][y + j] == 0 : q.append((x + i, y + j, score + 1))
26
27
28    q = [(r - 1, c - 1, 1)]
29    fromB[(r-1,c-1)] = 1
30
31    while q:
32        x, y, score = q.pop(0)
33
34
35        for (i,j) in [(0,1), (1,0),(-1,0), (0, -1)]:
36            if 0 <= x + i < r and 0 <= y + j < c and fromB[(x + i,y + j)] == float('inf'):
37
38                fromB[(x + i,y + j)] = score + 1
39
40                if map[x + i][y + j] == 0 : q.append((x + i, y + j, score + 1))
41
42
43    minVal = float('inf')
44
45
46
47    for i in range(r):
48        for j in range(c):
49            # print(fromA[(i,j)], fromB[(i,j)], fromA[(i,j)] + fromB[(i,j)])
50            minVal = min(minVal, fromA[(i,j)] + fromB[(i,j)])
51
52
53    return minVal - 1
54
55
56
57
58
59 print(solution([[0, 1, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [1, 1, 1, 0]]))

```