Name : Meet Manoj Maratha

Roll No : 19190

Assignment 1 - ECS 418

In [1]:
```python
# Import modules
import numpy as np
```

Important Functions

In [2]:

```python
def _distanceBetweenLineAndPoint(point : list | tuple, line_points : list[list] | list[tuple] | tuple[tuple] | tuple[list]) -> float:
    """Returns distance between a point and a line

    Args:
        point (list | tuple): point from which we need the distance
        line_points (list[list] | list[tuple] | tuple[tuple] | tuple[list]): two points which makes the line

    Returns:
        float: distance between the point and the line
    """
    assert(len(line_points)) == 2
    assert(len(point)) == 2
    for p in line_points : assert(len(p)) == 2
    p1, p2 = line_points
    m = (p2[1] - p1[1])/(p2[0] - p1[0])
    b = (p1[0] * p2[1] - p2[0] * p1[1])/(p1[0] - p2[0])
    return np.abs(- m*point[0] + point[1] - b)/np.sqrt(m**2+1)

def _distanceBetweenPolygonAndPoint(point : list | tuple, polygon_vertices : list[list] | list[tuple] | tuple[tuple] | tuple[list]) -> float:
    """Returns minimum distance between a point and polygon

    Args:
        point (list | tuple): point from which we need the minimum distance
        polygon_vertices (list[list] | list[tuple] | tuple[tuple] | tuple[list]): vertices of polygon

    Returns:
        float: Minumum distance between the point and the polygon
    """
    line_point = None
    min_distance = 1.7976931348623157e+308 # Maximum float value
    for points in polygon_vertices:
        p1, p2 = points
        r = np.dot(p2 - p1, point - p1)
        r /= np.linalg.norm(p2 - p1)**2
        if r < 0:
            min_dist = np.linalg.norm(point - p1)
            line_point = points
        elif r > 1:
            min_dist = np.linalg.norm(p2 - point)
            line_point = points
        else:
            min_dist = np.sqrt(np.linalg.norm(point - p1)**2 - (r * np.linalg.norm(p2 - p1))**2)
            line_point = points

        if min_dist < min_distance:
            min_distance = min_dist
    return min_distance

def polygonTangentLines(polygon : list[list] | list[tuple] | tuple[tuple] | tuple[list]) -> list[list]:
    """Returns slopes and y-intercenpt for the tangent lines

    Args:
        polygon (list[list] | list[tuple] | tuple[tuple] | tuple[list]): Polygon for which we want tangent lines

    Returns:
        list[list]: List of slopes and y-intercept, each lines data is stored as [...,[slope, y-intercept], ....]
    """
    tangents = []
    for edge in polygon:
        x1, y1 = edge[0]
```

```
        x2, y2 = edge[1]
        m = (y2 - y1)/(x2 - x1)
        b = (x1*y2 - x2*y1)/(x1-x2)
        tangents.append([m, b])
    return tangents

def polygonIntersectionPoint(poly1 : list[list] | list[tuple] | tuple[tuple] | tuple[list], poly2 : list[list] | list[tuple] | tuple[tuple] | tuple[list]) -> dict:
    """Returns the intersecting points of two polygons

    Args:
        poly1 (list[list] | list[tuple] | tuple[tuple] | tuple[list]): Polygon 1
        poly2 (list[list] | list[tuple] | tuple[tuple] | tuple[list]): Polygon 2

    Returns:
        float | bool: Intersection points
    """
    intersection_points = {}
    for line1 in poly1:
        for line2 in poly2:
            A, B = line1
            C, D = line2
            # If there exists a point of intersection than it satisfies equation P = C + CD.t1 = A + AB.t2
            # Simplyfying the above equation gives us t
            t = np.cross(C - A, D - C)/np.cross(B - A, D - C)

            # If t is NAN then the lines are collinear
            # If t is INF then the lines are parallel
            # If t is > 1 then the lines intersect outside their end points
            # If t is between 0 and 1, the lines intersect somewhere in their end points
            if 0 <= t and t <= 1 and not np.isnan(t):
                idx = f'({line1[0]}, {line1[1]}),({line2[0]}, {line2[1]})'
                intersection_points[idx] = A + (B - A).dot(t)
    return intersection_points
```

Question 1

Write functions to

- Compute a line between two points.
- Compute distance between two points.
- Compute perpendicular distance between a point and a line segment
- Compute distance between a point and a Polygon
- Compute tangent vector to a polygon
- Find intersection of two polygons

In [3]:
```
x1, y1, x2, y2 = 5, 3, 7, 9
print(f"Let the two points given be (5, 3) and (7, 9).")
m = (y2 - y1)/(x2 - x1)
b = (x1*y2 - x2*y1)/(x1-x2)
print(f"The equation of line between these two points is given as y = ({m})x + ({b}).")
```

```
Let the two points given be (5, 3) and (7, 9).
The equation of line between these two points is given as y = (3.0)x + (-12.0).
```

In [4]:
```
x1, y1, x2, y2 = 5, 3, 7, 9
m = (y2 - y1)/(x2 - x1)
b = (x1*y2 - x2*y1)/(x1-x2)
m,b
```

Out[4]: (3.0, -12.0)

In [5]:
```
x1, y1, x2, y2 = 1, 1, 5, 5
m = (y2 - y1)/(x2 - x1)
b = (x1*y2 - x2*y1)/(x1-x2)
m,b
```

Out[5]: (1.0, -0.0)

In [6]:
```
x1, y1, x2, y2 = 5, 3, 7, 9
print(f"Let the two points given be (5, 3) and (7, 9).")
dist = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
print(f"The distance between these two points is {dist}.")
```

```
Let the two points given be (5, 3) and (7, 9).
The distance between these two points is 6.324555320336759.
```

In [7]:
```
p1, p2, p3 = np.array([35, 65]), np.array([65, 20]), np.array([1, 1])
m = (p2[1] - p1[1])/(p2[0] - p1[0])
b = (p1[0] * p2[1] - p2[0] * p1[1])/(p1[0] - p2[0])
print(f"Let the line be formed by points {p1} and {p2}. The equation of this line is given as y = ({m})x + ({b}).")
print(f"The distance of point {p3} from this line is {_distanceBetweenLineAndPoint(p3, (p1, p2))}.")
```

```
Let the line be formed by points [35 65] and [65 20]. The equation of this line is given as y = (-1.5)x + (117.5).
The distance of point [1 1] from this line is 63.790522565901355.
```

In [8]:
```
polygon_vertices = np.array([
    [[10, 20], [20, 35]],
    [[20, 35], [35, 65]],
    [[35, 65], [65, 20]],
    [[65, 20], [15, 15]],
    [[15, 15], [10, 20]]
])
point = np.array([5, 5])
print(f"Minumum distance between the point {point} and the given polygon is {_distanceBetweenPolygonAndPoint(point, polygon_vertices)}")
```

```
Minumum distance between the point [5 5] and the given polygon is 14.142135623730951
```

In [9]:
```python
polygon_vertices = np.array([
    [[10, 20], [20, 35]],
    [[20, 35], [35, 65]],
    [[35, 65], [65, 20]],
    [[65, 20], [15, 15]],
    [[15, 15], [10, 20]]
])
tangents = polygonTangentLines(polygon_vertices)
print(f"Tangent lines for the given polygon are : ")
for line in tangents:
    print(f"y = ({line[0]})x + ({line[1]})")
```

```
Tangent lines for the given polygon are :
y = (1.5)x + (5.0)
y = (2.0)x + (-5.0)
y = (-1.5)x + (117.5)
y = (0.1)x + (13.5)
y = (-1.0)x + (30.0)
```

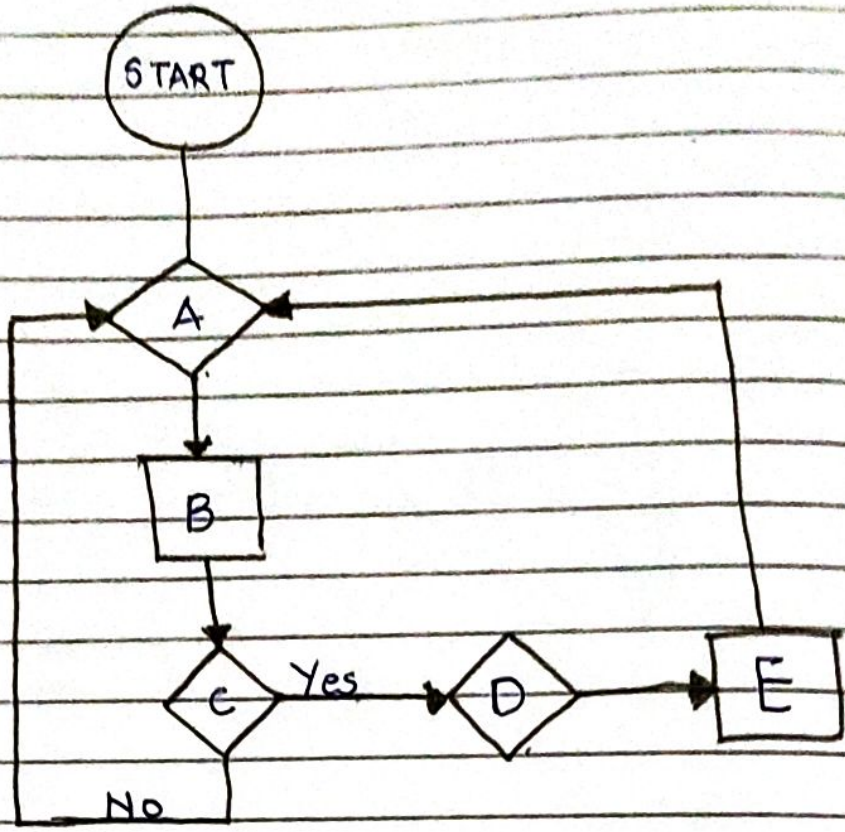
In [10]:
```python
polygon_vertices1 = np.array([
    [[10, 20], [20, 35]],
    [[20, 35], [35, 65]],
    [[35, 65], [65, 20]],
    [[65, 20], [15, 15]],
    [[15, 15], [10, 20]]
])
polygon_vertices2 = np.array([
    [[10, 20], [15, 45]],
    [[15, 45], [25, 65]],
    [[25, 65], [13, 20]],
    [[13, 20], [3, 15]],
    [[3, 15], [10, 20]]
])
intersecting_points = polygonIntersectionPoint(polygon_vertices1, polygon_vertices2)
print(f"The intersecting points for the two given polygons is :")
for id, vals in intersecting_points.items():
    print(f"{id} : {vals}")
```

```
The intersecting points for the two given polygons is :
([10 20], [20 35]),([10 20], [15 45]) : [10. 20.]
([10 20], [20 35]),([25 65], [13 20]) : [15.  27.5]
([10 20], [20 35]),([ 3 15], [10 20]) : [10. 20.]
([35 65], [65 20]),([13 20], [ 3 15]) : [52.  39.5]
([35 65], [65 20]),([ 3 15], [10 20]) : [47.25806452 46.61290323]
([15 15], [10 20]),([10 20], [15 45]) : [10. 20.]
([15 15], [10 20]),([25 65], [13 20]) : [12.36842105 17.63157895]
([15 15], [10 20]),([13 20], [ 3 15]) : [11. 19.]
([15 15], [10 20]),([ 3 15], [10 20]) : [10. 20.]
```

```
/tmp/ipykernel_1512/3559817411.py:84: RuntimeWarning: divide by zero encountered in divide
  t = np.cross(C - A, D - C)/np.cross(B - A, D - C)
```

Q.3.

→



A: While not at goal location.

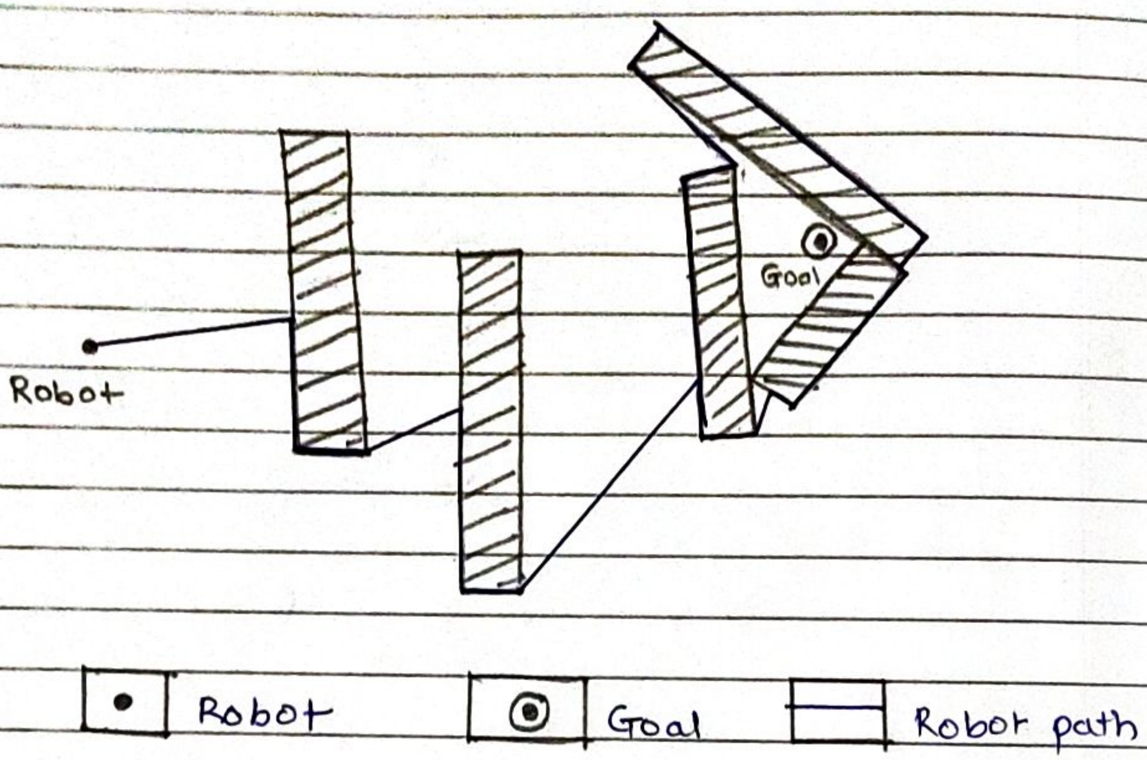
B: Move towards goal

C: If hit an obstacle

D: While not able to move towards goal

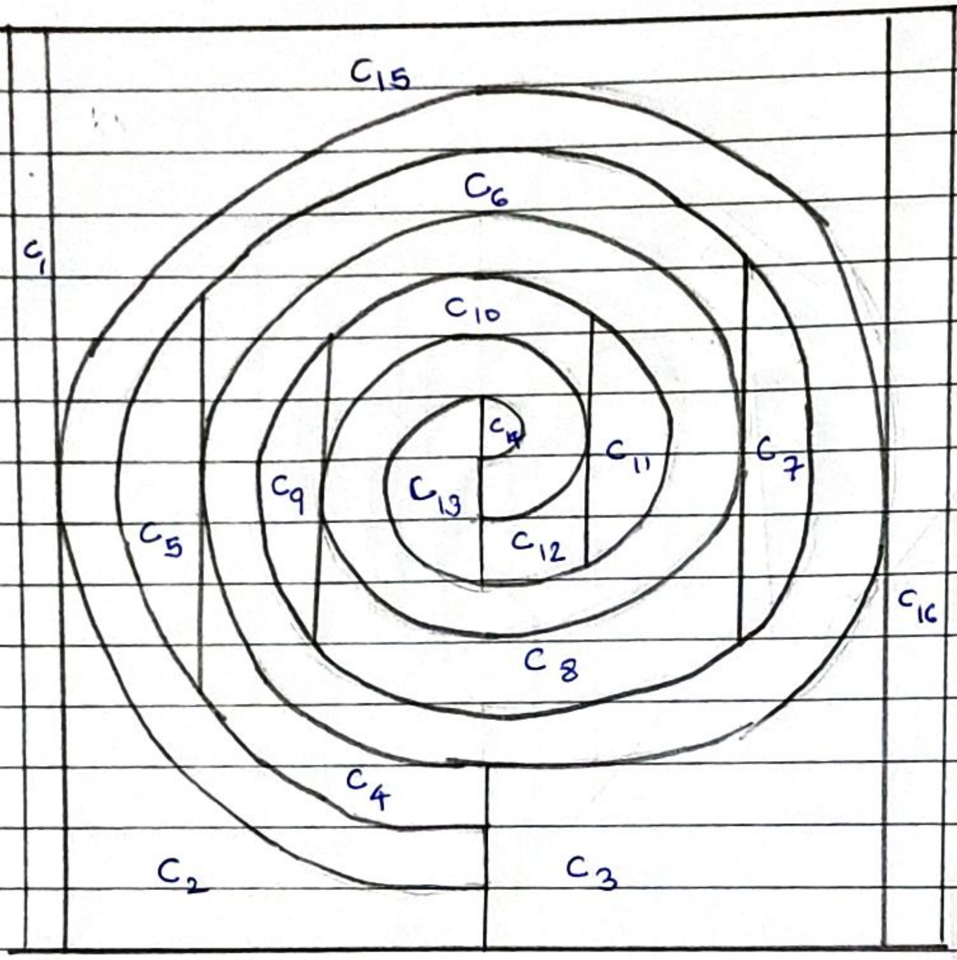
E: Follow obstacle moving to the right until can head towards goal.

Q.4
→

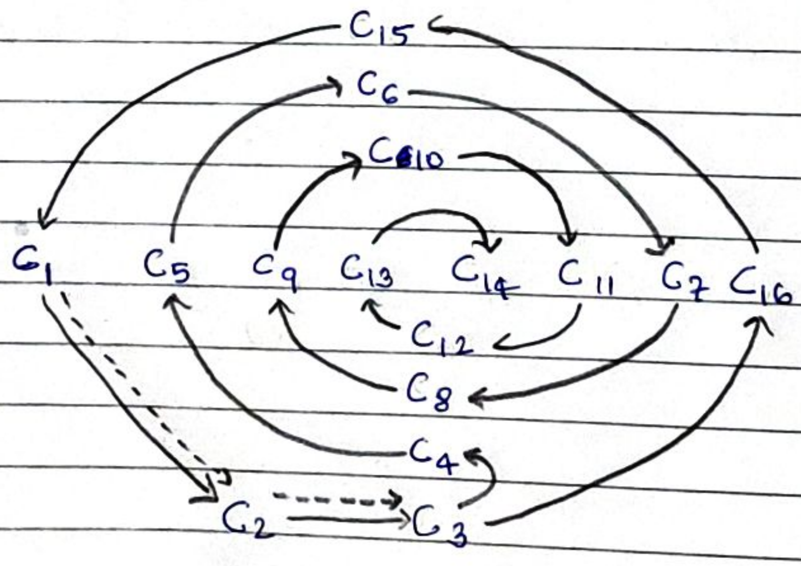


As this (above figure) is a case where bug 0 which always turns right fails, bug 0 is not a complete algorithm.

Q.7.
→

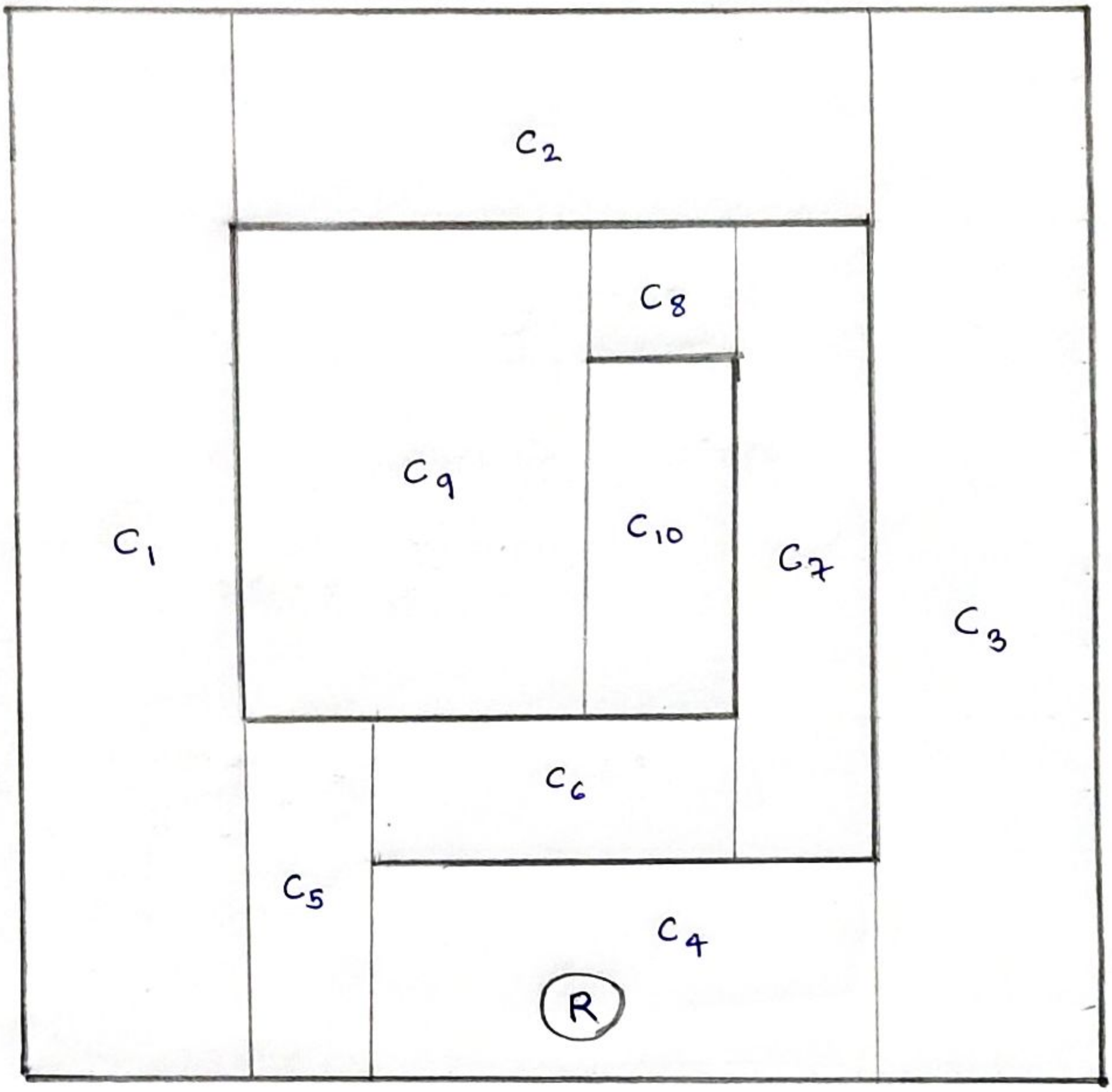


C_i : i^{th} cell.



Q. 7

$\longrightarrow$ <u>Trapezoidal decomposition</u>:



C_i : i^{th} cell

R : Robot

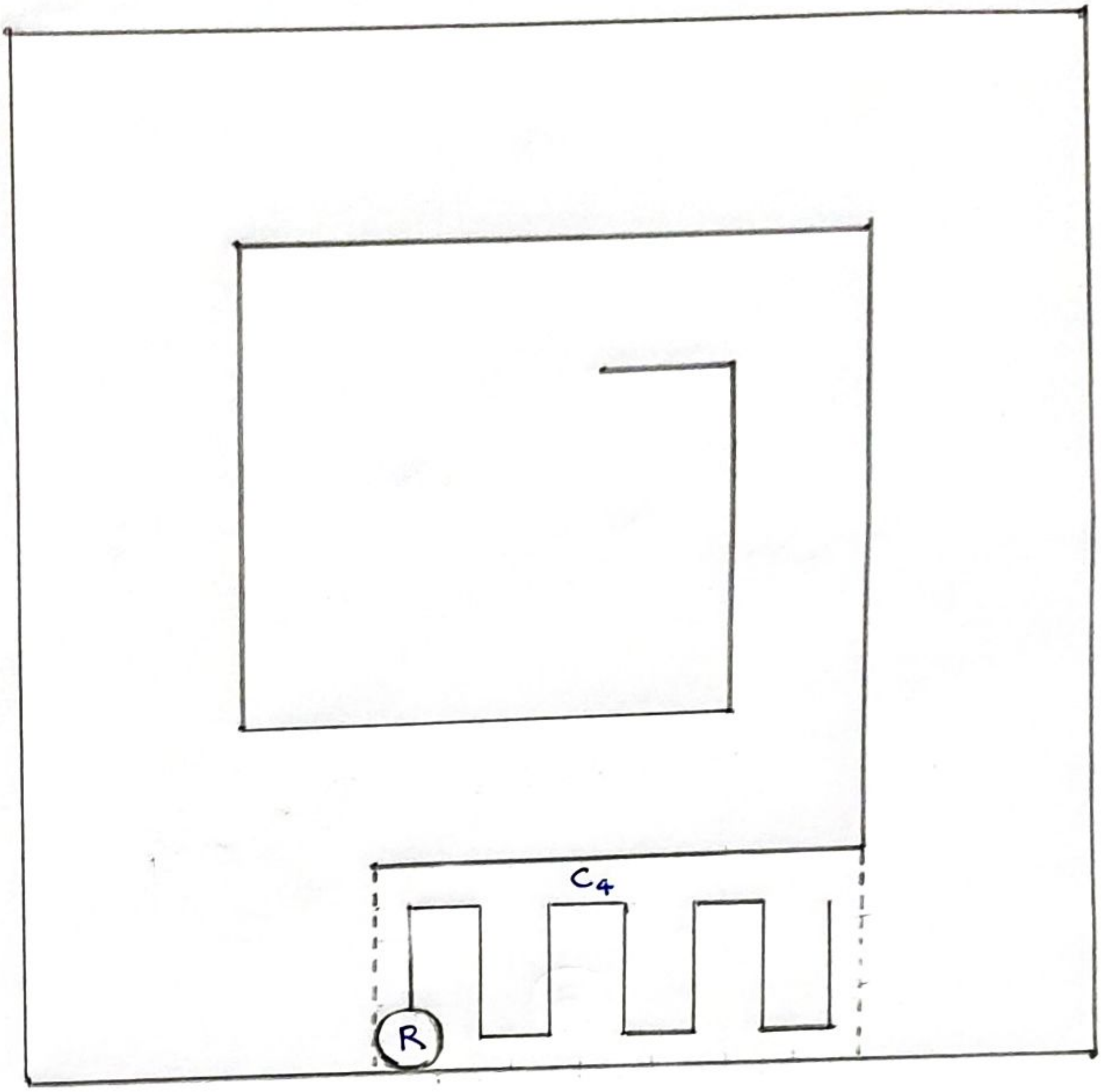
<u>Rebb graph</u>:

$$C_4 \rightarrow C_3 \rightarrow C_2 \rightarrow C_1 \rightarrow C_5$$

$$\downarrow$$

$$C_{10} \leftarrow C_9 \leftarrow C_8 \leftarrow C_7 \leftarrow C_6$$

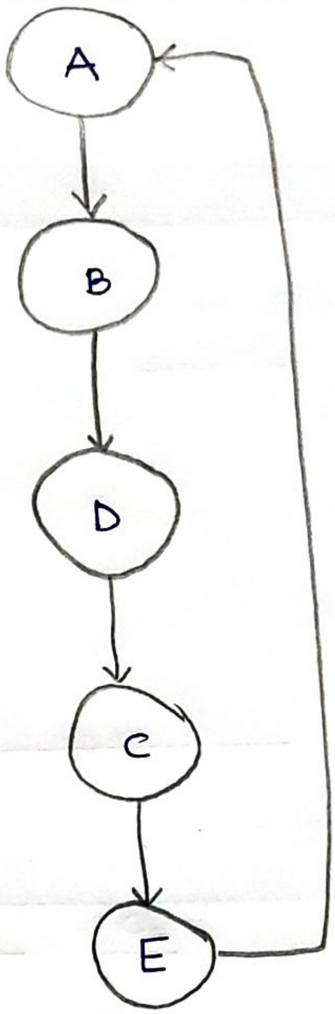
Q.8.
→



The above consists of lawn mower pattern cell C_4 in the previous environment.

Q.3

→



A: While not at goal location.

B: Move towards goal.

C: If hit an obstacle.

D: While not able to move towards goal

E: Follow obstacle moving to right until can head towards goal location.