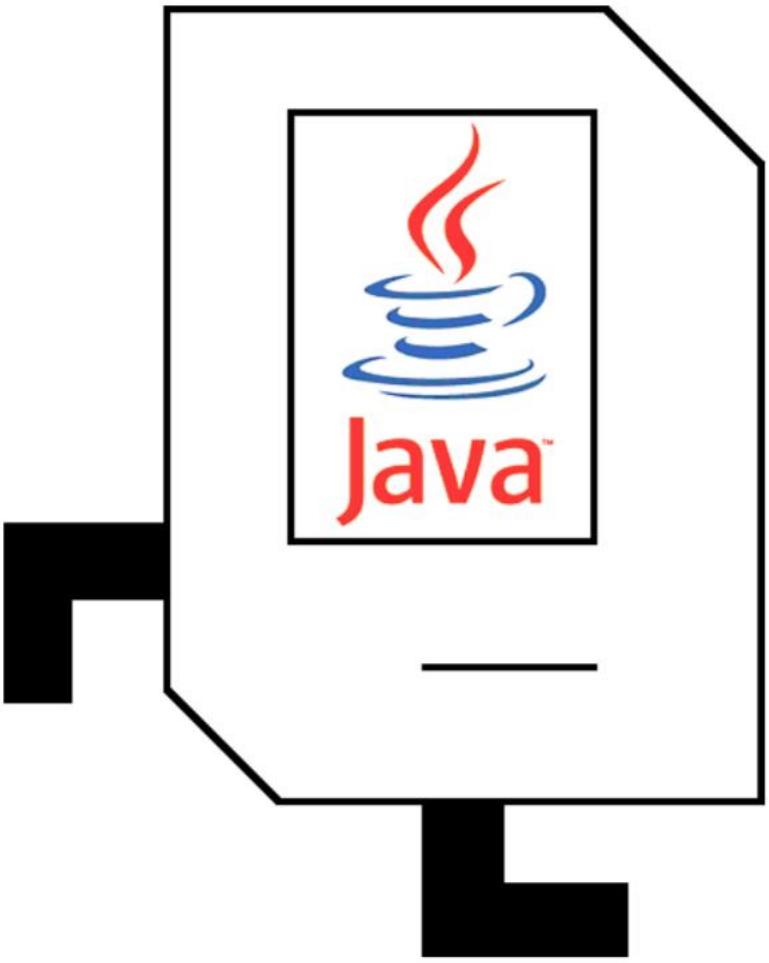


Report

Assignment (Karel the Robot)

Name: Motaz Bataineh

Table of Contents:

• Introduction

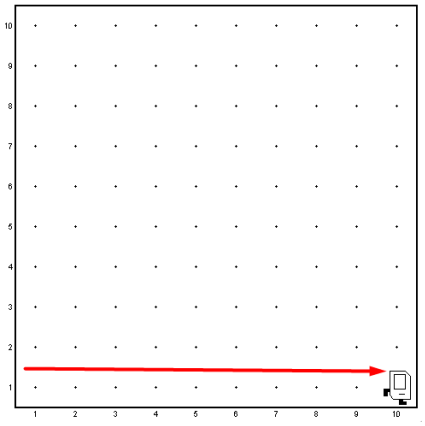
The problem was to create an optimized algorithms in Java to automate Karel to divide any possible grid into four equal areas. The problem has multiple cases, so I wrote four algorithms to ensure my solution covers any map with the least number of beepers and steps possible.

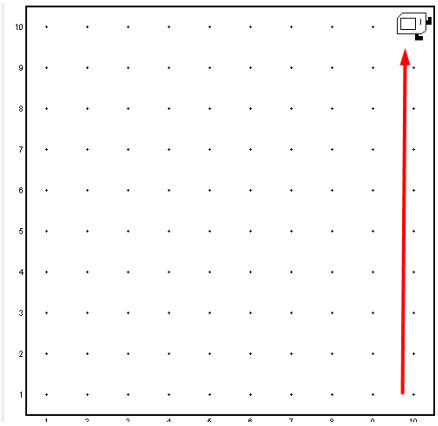
• Explanation before start

let's consider the process of dividing a map into four equal parts, and how these parts can further be divided. The goal is to find the maximum number of divisions possible whether that results in 4, 3, 2, or 1 chamber(s). To achieve an accurate answer we need to explore all possible scenarios.

First, before it is calculated whether it is divided into the largest possible number, I had to calculate the number of rows and columns, because the solutions depend on this, so my method was as follows:

o I made Karel walk to the far right and then made her go to the highest point, which allowed me to calculate the length and width without using the existing methods. I defined the length and width variables as global variables, so I could use them anywhere I needed them, that cost me [(width + height) – 2] step's. I used the frontIsClear() function to help me bypass any error or problem in the program.





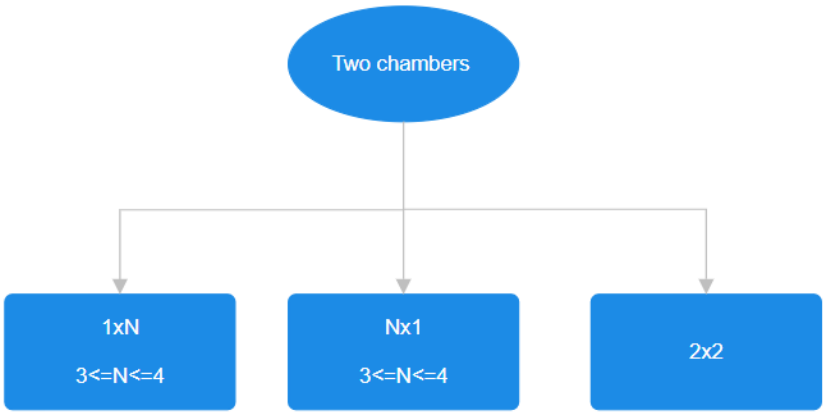
- ## Cant Divided to any chamber

  Any map can be divided into one part, but in this assignment, I have to avoid as much as possible that it is divided into one chamber, to divide any map into one equal part, I only have two cases to be divided into one part ,If the map contains one row and one column, then this will certainly not be divided into four, three, or two parts, so it will be one chamber, and there is the other case that if the row is equal to one and the column is equal to two or vice versa.

1x1	1x2	2x1

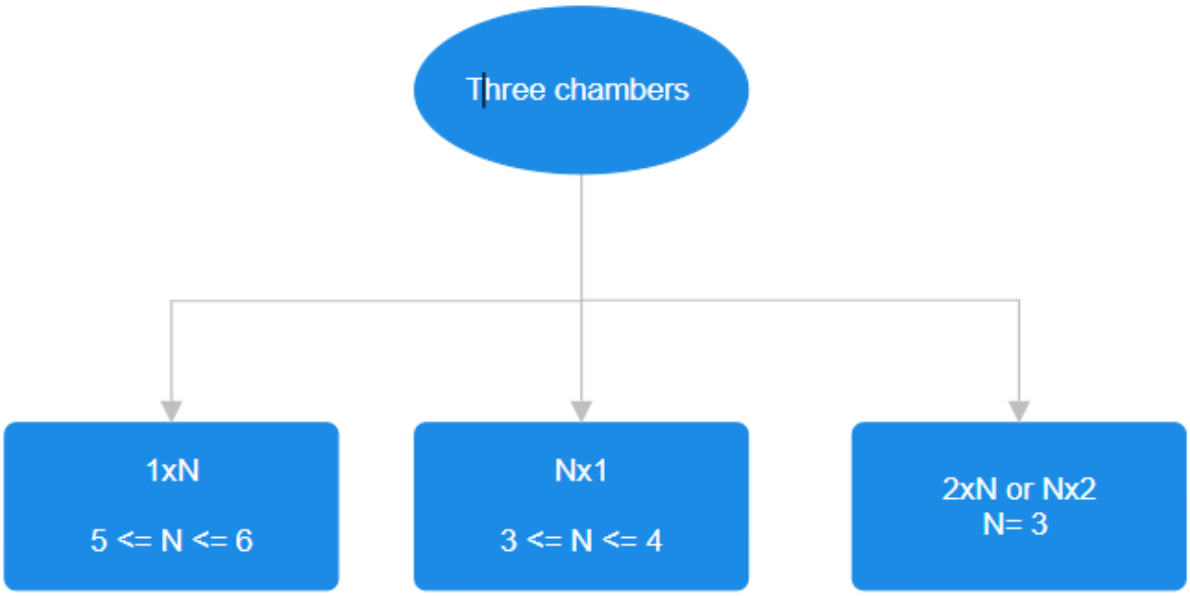
- ## Divided to Two chambers

  Now we will come to the case that the map is divided into two equal parts, and there are a few cases that will allow us to divide it into two equal parts, the following cases:



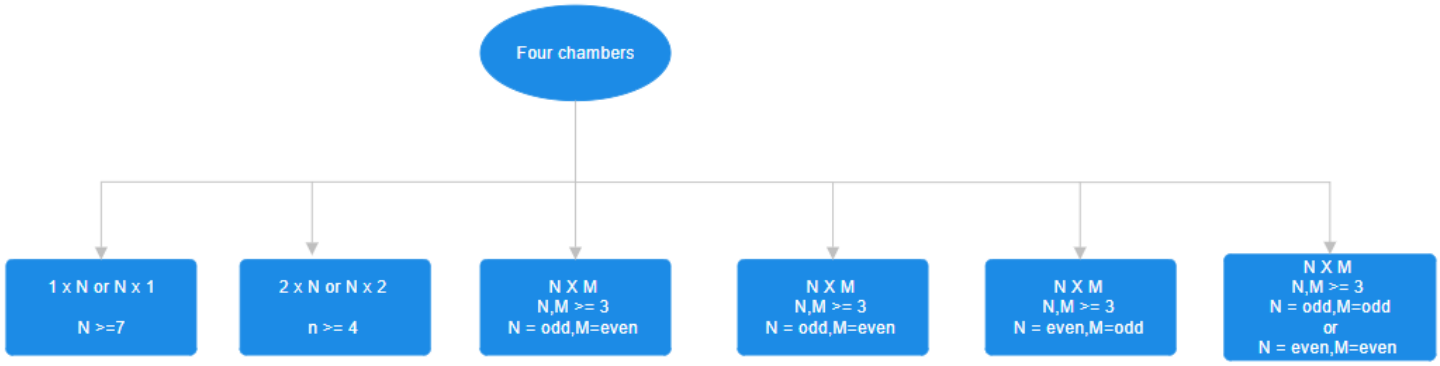
• Divided to Three chambers

As we have seen in the previous cases, the cases of dividing chambers into one or two were completely covered. Now, at the time of dividing the map into three chambers, the chambers were divided in similar cases to divide them into two, with a difference of some N value , To divide the map into three equal parts, I have only three cases, (1 x N | N x 1) which are that N is equal to 5 or 6, or that it is (2 x N | N x 2) which are that N is equal to 3 .



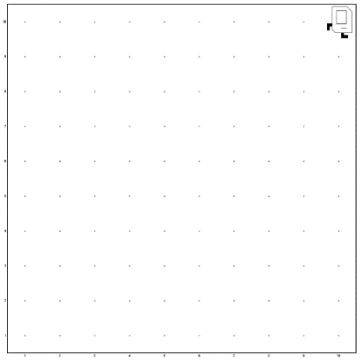
- # Divided to Four chambers

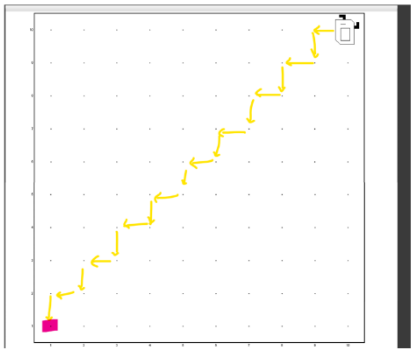
  This is the most comprehensive case, which has many cases, and that required a lot of thinking from me to reach the best suitable solution for dividing the chambers, in some cases I did not reach a more appropriate solution, and in others I found a suitable solution that matches the conditions required in the duty, so after counting the cases I divided them into several cases, namely As shown in the picture

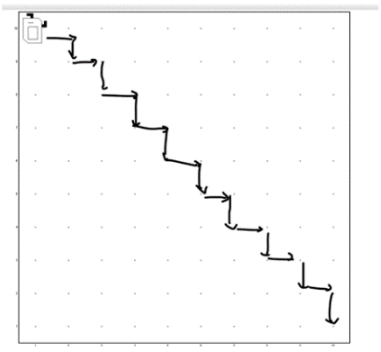


- # Diagonal algorithm(NXM and both even)

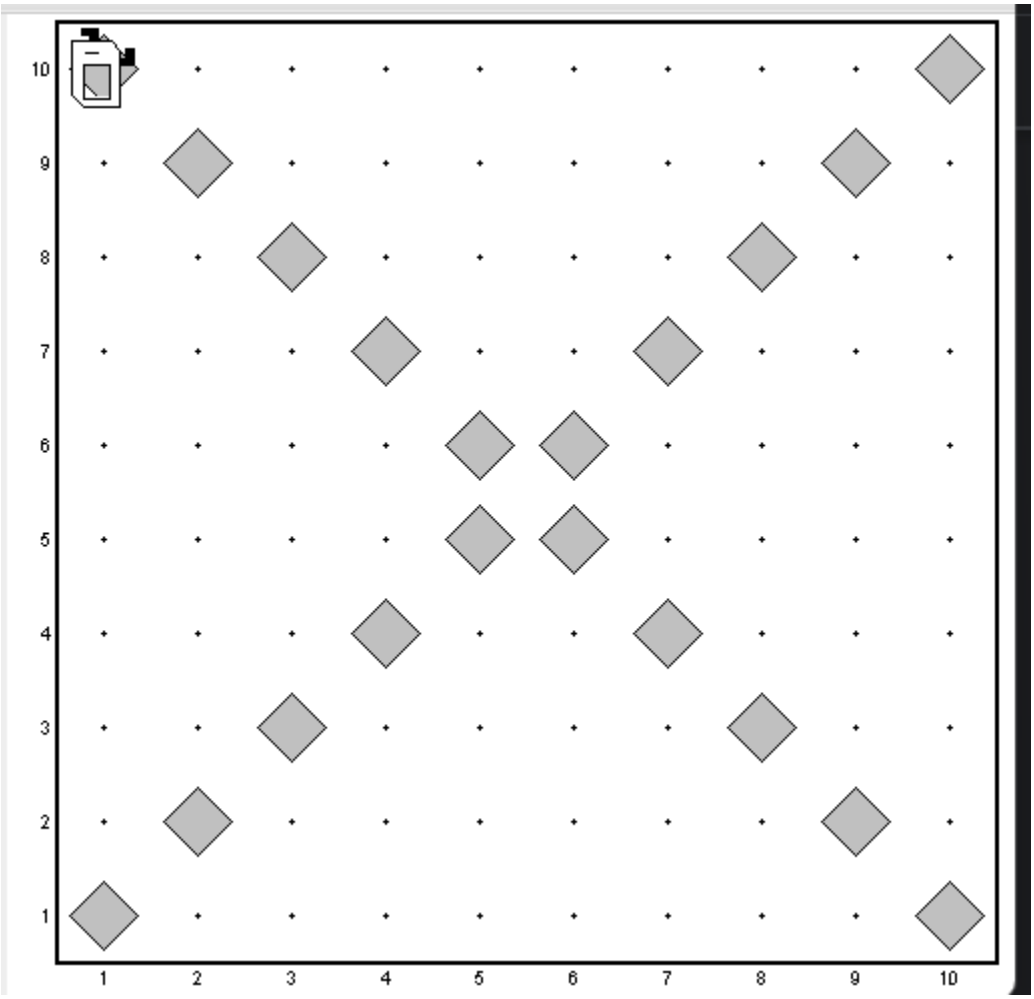
  the Karel will check if the number of rows and columns are even and equal . If the condition was true , then the Karle will use the diagonal algorithm to solve the problem .







After applying the diagonal algorithm

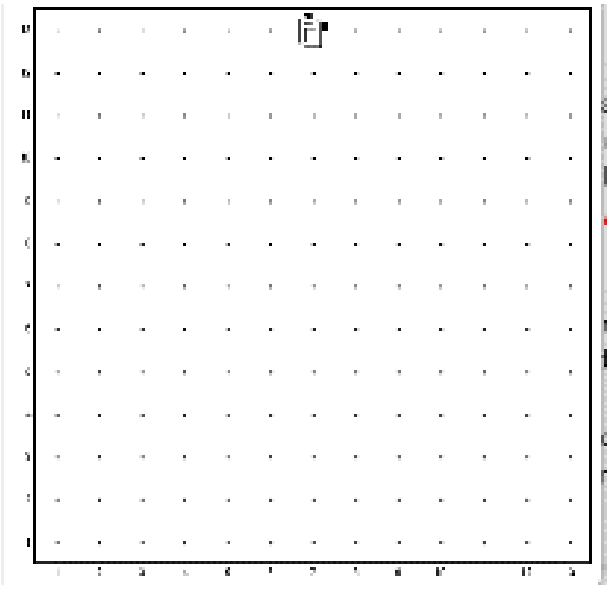


• OddOdd Algorithm (N x M and M x N , N, M >= 3 ,both odd)

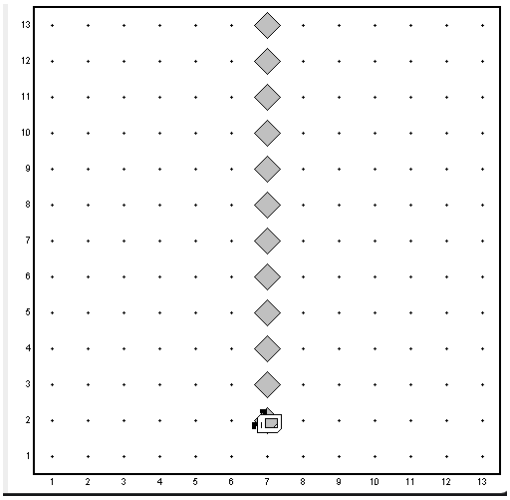
In case the column and row are greater than or equal to three, we will have the simplest case I encountered, which is to just divide the map for half of the column and half of the row, or in other words I put all the beeps on the longitude of column/2 + 1 and also the same along the line row/2 + 1 .

For example : If the map contains 12 Rows and 12 column , what are the exact steps the Karel will do to solve the problem?

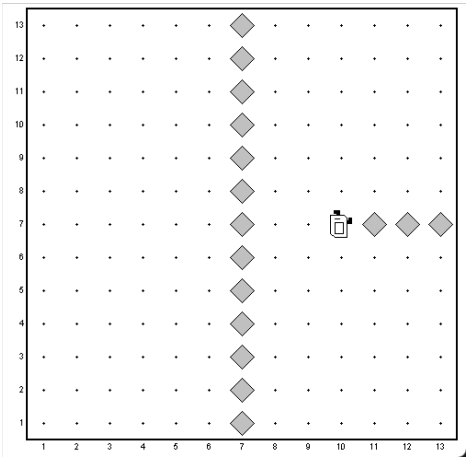
1) Karel will move to the middle of the column number, which is from the most right corner to the middle of the same row and put a beeper



2) Karle will move all through to the point (7,1), will putting a beeper with every move

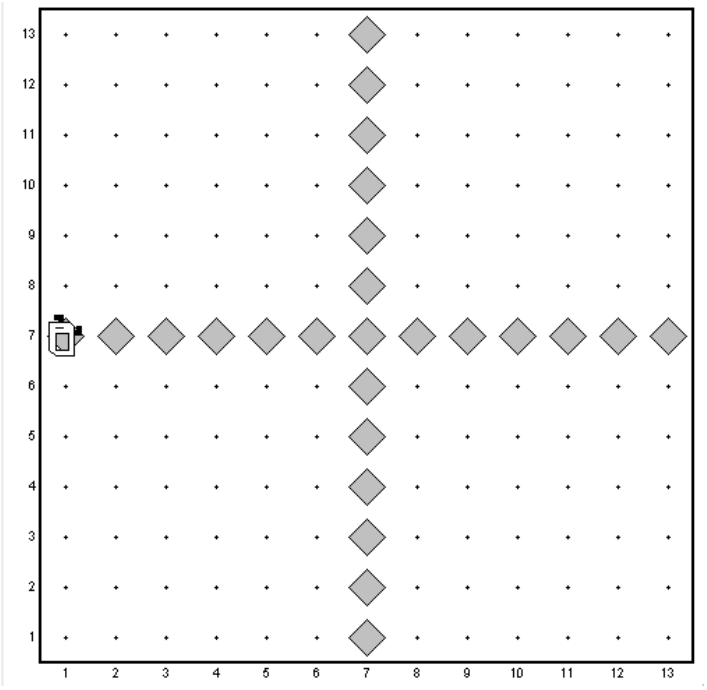


3) Karel will turn left and move until found wall and turn left and walk half the rows number and turn left



4) Move until found wall and put a beeper while moving

After applying the general algorithm



- # SingleDimension algorithm

is designed to handle cases where either the width or height of the map is 1 which means the map is either a single row or a single column Here's a detailed explanation of the algorithm

1) The function starts by initializing swap to false, respectively.
   If height is 1 it means the map is a single row. The code then swaps width and height to handle the single row as a column and sets swap to true.If the swap flag is not set meaning the map is a single column Karel turns right to face the correct direction.

2) **Setting Chunk Size**:

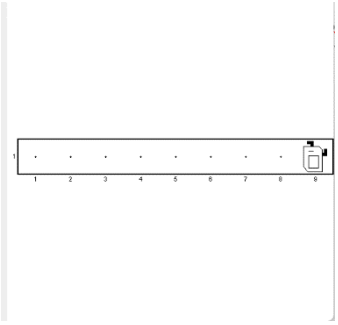
- chmprSize is calculated based on the height If height is greater than 8 it's divided by 4 and reduced by 1 otherwise it's set to 1
- This value helps to determine the size of the chunks in which beepers will be placed.

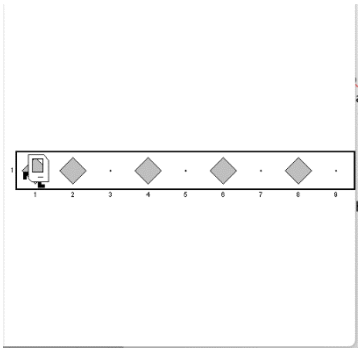
3) **First Loop**:

- The first while loop moves Karel by chmprSize steps, placing a beeper every chmprSize steps, and stops after placing 4 beepers or hitting a wall The first move is adjusted by subtracting 1 from chmprSize to account for the initial position

4) **Second Loop**:

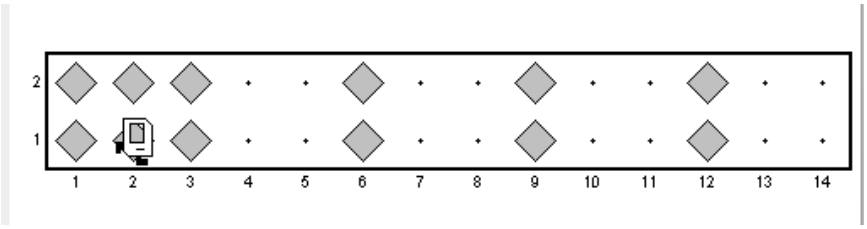
- If there are still clear spaces and the height is not 2 Karel continues to move forward by 1 step placing beepers in each cell





- DoubleDimension algorithm

  It is the same as the previous case, but the only difference is that it will fill every two lines or two columns together, the logic is the same as the previous case.

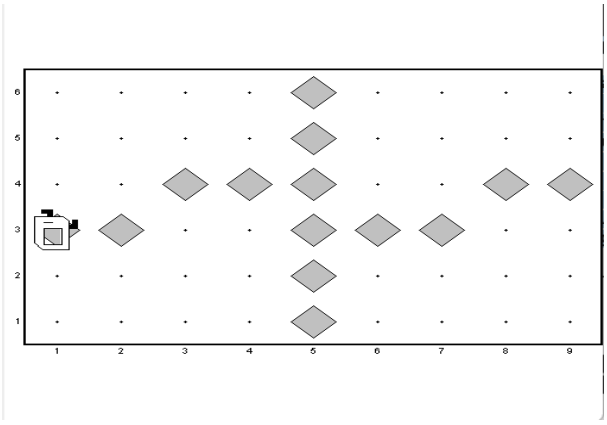


- # EvenOdd algorithm (N x M and M x N , N, M >= 3 , ODD and EVEN vice versa)

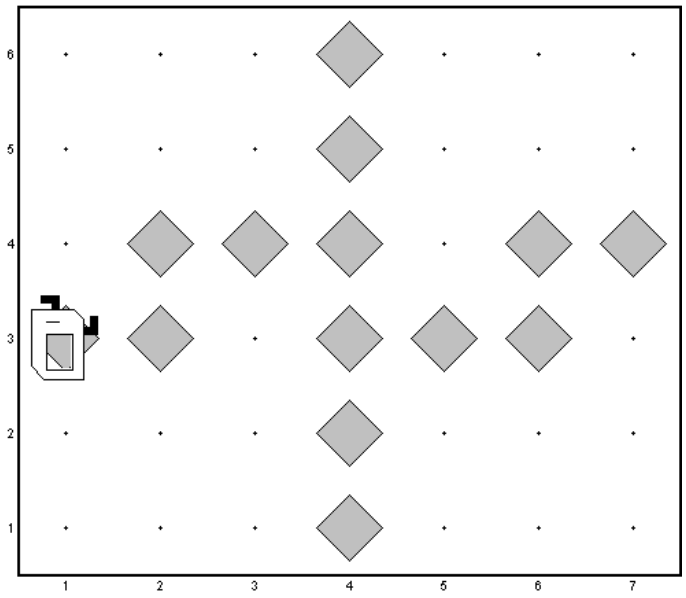
  This is one of the cases that I tried as much as possible to optimize it. I had a preliminary solution, which was to go towards the odd number, the row or column, and start filling the beepers from N/2, where N is the odd number, and complete the rest of the lines as I did in the case of the even rows and columns, so I started Thinking about how I can reach a suitable and optimize solution so that the number of steps is as few as possible and the number of beepers is as few as possible, so I started experimenting on many different cases, which I have a general perspective on the thing, which is the following:

  When I divide the map in half on the odd side, I have two equal parts, so when I tried many cases, I noticed that I might have a somewhat fork-like shape or a kind of zigzag, but there were two impediments. The first is when the curve will be completed, and how much it will walk until it reaches the curve, so the rule was as follows: first he will walk a certain distance, then he will stop, and I will come to the turn of the curve. I calculated this distance by means that since the map is divided in half, this means that the remaining distance from the right and left is N/2, to see how you walk and whether the curve is also full of a beeper or not, you have to see if the result of the division is not a multiple of the two. This means that you have to fill in the

  angle or curve. Things will become clear in the examples.



In this example, divide the map into two halves first, then it will come up with the remaining distance, which can be calculated without moving the robot, as follows: the length is divided by 2, and the distance will be found (only when it is odd), when calculating the distance here, the result was floor(9/2)= 4 which means that the distance between the far right and the middle is 4, now we will come to see if we can put a beeper or not, I will check if the 4%2?=0 the answer is yes then I will not put any beeper at the curve, so he will walk two steps and then crouch Up or down doesn't make a difference and finally goes to the wall and does the same to the opposite side.



In this example, when we calculated the distance here, the result was floor(7/2)=3, which means that the distance between the far right and the middle is 3. Now we will come to see if we can put the beeper or not. I will check if the 3%2? =0 The answer is no. So I will put a beeper at the curve, so he will walk one step + one because he did not fulfill the condition, then crouch up or down and finally turn to the wall and do the same to the opposite side.