Karel Assignment

Mohannad Al-Tamimi

Overview

The objective of this assignment was to make the Karel bot, given any map, divide said map into 4 equal chambers if possible. If it is not possible divide into 4 then divide the greatest number of equal chambers (3 or 2). This had to be done using the least number of beepers and steps.

Thought Process

So, the first thing I did was think of the different types of maps and the special cases we could encounter. We have maps that have either width or height equal to 1, maps with height or width equal to 2, maps where the height and width are odd, maps where the height and width are even, and finally maps with mixed height and width between odd and even.

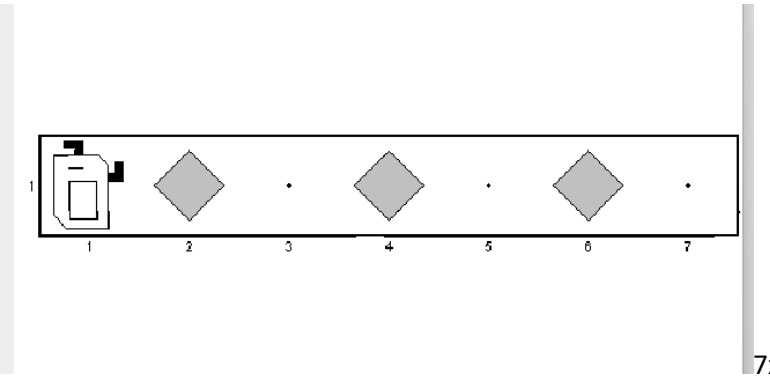
Since we have so many cases, I used 3 algorithms, a case-based algorithm based on the dimensions of the map, a pattern- based algorithm that follows up on the cases. Where, depending on the dimensions we place the beepers in a different way, S-shape and plus shape.

First, we need to get the dimensions of the map. I used a function to get the height and a function to get the width by traversing horizontally first then vertically.

One dimensional map:

Now we have the dimensions of the map and can start thinking how to handle each case. The first case is the one-dimensional row or column 1xN and Nx1, in this case there are 2 maps that can't be split 1x1 and 1x2 so we handle those separately in the beginning. Now for any 1xN or Nx1 map we always make sure that the height is the longer dimension by swapping before we start drawing to standardize the approach for both types of maps so we swap the width with the height if the height is one, we need to find the chamber size and determine the number of beepers needed and where to place them. Then we place the beepers.

Example:



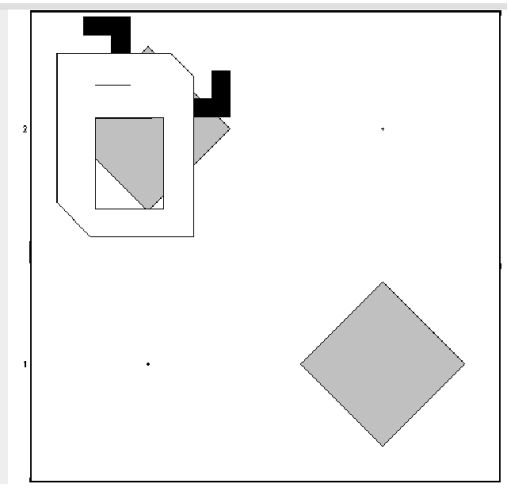
7x1

A map like this one which is 7x1 took 6 steps to measure and another 6 steps to place the beepers. As is obvious the beepers used are three.

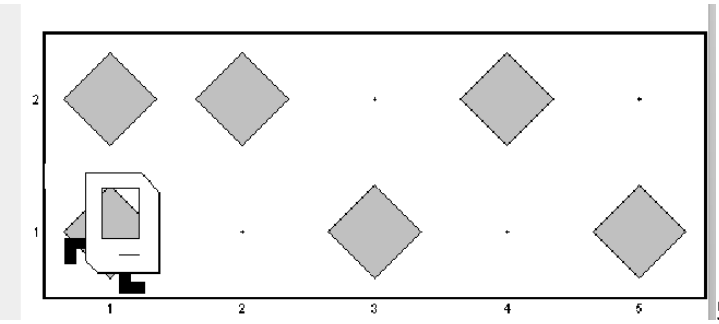
Two dimensional map:

In a two dimensional map that is 2xN or Nx2 we start by standraizing the approach the same way we did in the method before, swapping the height and width if the height is equal to 2. Next in these two dimensions we also have 2 cases, one where N<7 and one where N >=7. If N<7 we use a pattern placed approach to blace the beepers, you can say it's almost a zigzag pattern until we have four equal chambers then we fill the rest with beepers to ensure we stay on target. This also ensures we can handle maps that can't be divided into 4.

Examples:



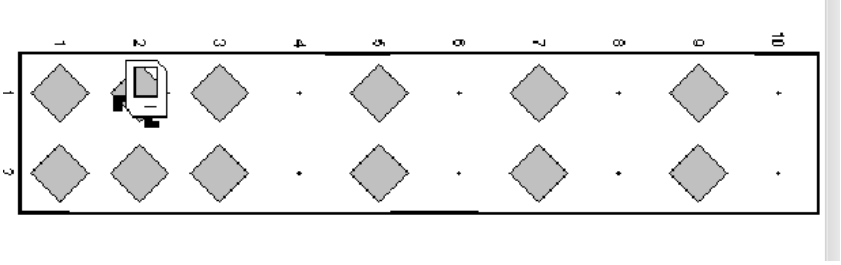
2x2



5x2

Now we are done with case 1, for case 2 we use an S-shape movement to place the beepers

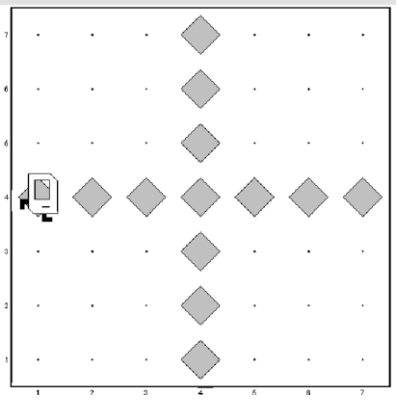
Example:


2x10

Odd dimensional map:

The next case we have to handle is the Odd x Odd map. The way I solved this is by first calculating the step count for plus shaped and S-shaped traversal and comparing which is more efficient based on the map size we have. Then based on the scores and dimension we call either S-shape or plus shaped. If the room is small we go with plus shaped pattern.

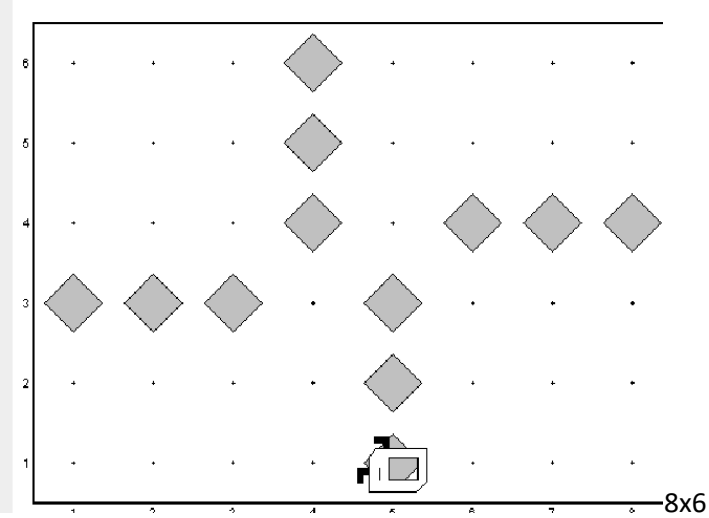
Example:


7x7

Even dimensional map:

We swap to standirize, calculate half dimensions to detirmine the area of each half of the room. Calculate the area difference. Now based on the difference is how we place the beepers. we try to place more beepers in the larger half of the room.

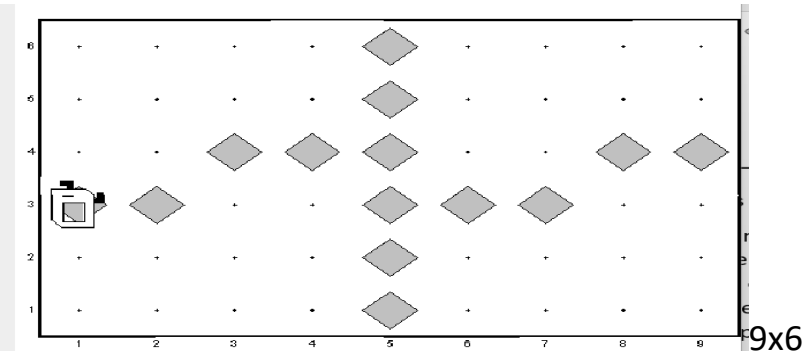
Example:



Mixed dimensions map:

We check which dimension is which. Ensure the height is always the odd dimension. We use a combination of the odd and even handling methods here, like the method to calculate half of the even dimension and then create a bridge, you could say, of beepers in the middle if there is a big difference between the two halfs. We use plus shaped movement similar to the odd handling method as well.

Example:



Finally let me explain some of the helper methods used to optimize the code and make it more modular.

CalculatePlusCount and calculateSCount: these were used to calculate the number of steps needed for plus and s-shpaed traversals in a map.

MoveAndCountSteps: this is the main method that draws divides the chambers and counts the number of steps taken to divide the chambers.

putBeeperIfNotPresent: this method puts a beeper on the map.

sShapeMovement: this method moves Karel in an S-shape

executePlusShapeMovement: this makes karel move in a plus shape.

fillRemainingSteps: this method moves karel until it encounters a wall and then turns around and moves back the same sumber of steps minus 1 while placing beepers.

turnDirection: this method takes a swap flag (boolean) and a number(direction) and determines the diection based on the value of the swap. If swap is false, 1 is left and 2 is right. If swap is true, it is the opposite