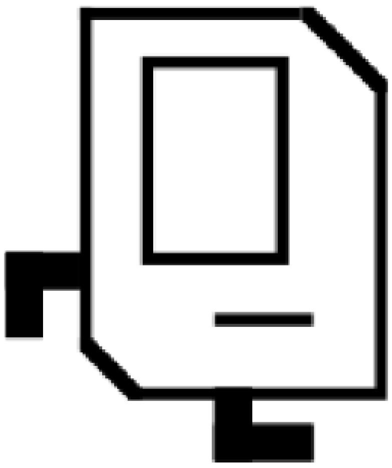

Karel The Robot

Done By:

Lana Alnimreen

Supervisor:

Dr.Motasem Aldiab

Abstract

In this report, a Java-based approach for dividing maps in the Karel programming environment is presented and examined. The offered code includes an algorithm that systematically divides maps into equal chambers according to their dimensions. The algorithm will be briefly described and examined in the sections that follow.

Table of Contents

List Of Figures

Karel Robot

Introduction

Karel is a very simple robot living in a very simple world. By giving Karel a set of commands, you can direct it to perform certain tasks within its world. In this document I perform dividing a given map into 4 equal chambers or divide them into the maximum possible number of equal chambers (1, 2, or 3, with minimum number of movement and number of beepers).

Getting The Dimensions of Map

getDim ( )

I invoke this method twice it returns an integer that express the length of line, by counting Karel's steps on the x-axis and the y-axis.

Dividing The Map

divideMap (row , column)

It takes the map dimensions as a parameter and contains many if and else statements that specify the even and odd dimensions with some special cases, each conditional statement calls a specific method to divide either x-axis or y-axis.

If both x and y are greater than or equal to three, the map will sure be divided into four equal chambers by cross boundary shape.

Case 1: at least one of the axes is less or equal two

Firstly, if the map is (1,1) it will show the message " The map can't be divided. ", otherwise it will invoke *handleSpecialCases*( ) method which in its turn check if the axes one or two and If the other axis is more or equal eight or less than eight.

I split the case where the other axis is more or equal eight, because I can divide the map into four champers easily. In *divideMoreEight*( ) method I checked the divisibility of the axis by four if it's pass It will put beeper and move steps equal to the result of dividing the axis by four then put beeper and so on. If it’s not pass it will fill the beginning of the line with beepers until it got to where the left of the line is divisible by four.

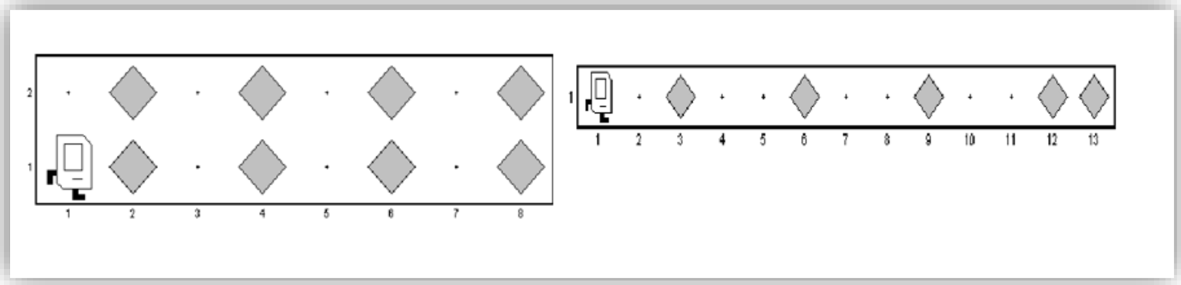

Figure 1: (2,8) and (1,13)

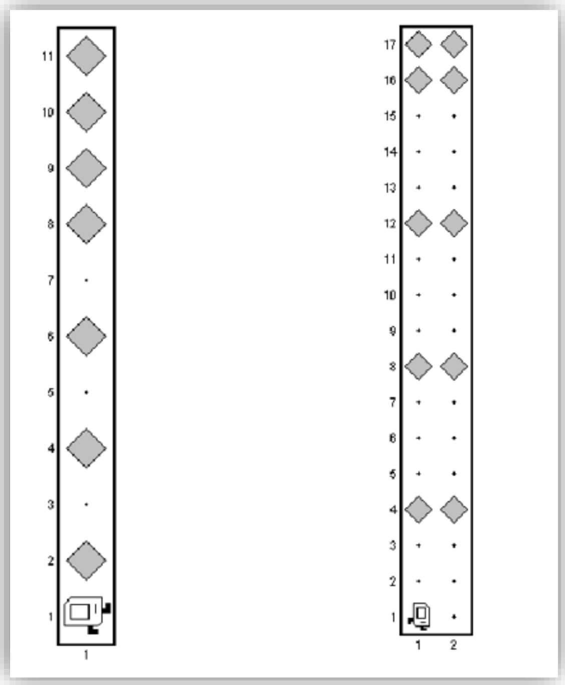

Figure 2:(11,1) and (17,2)

In *divideLessEight*( ) method, in this case I tried to get the maximum number of equal champers. So, in axis equal four or six I did the zigzag thing and I got four equal champers. The method is just a moves step then put beeper.

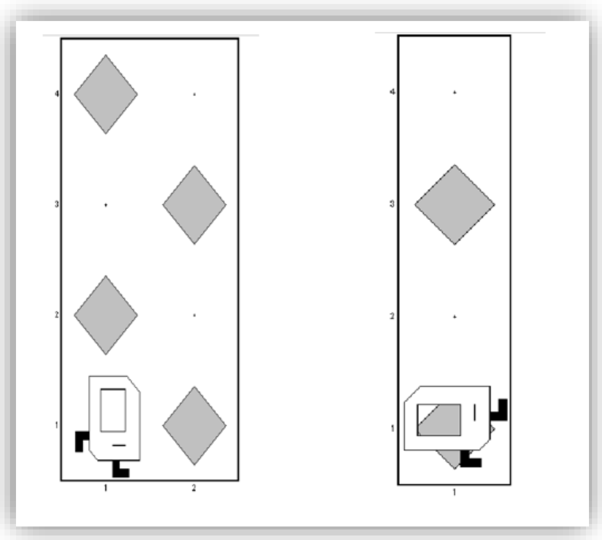

Figure 3: (4,2) and (4,1)

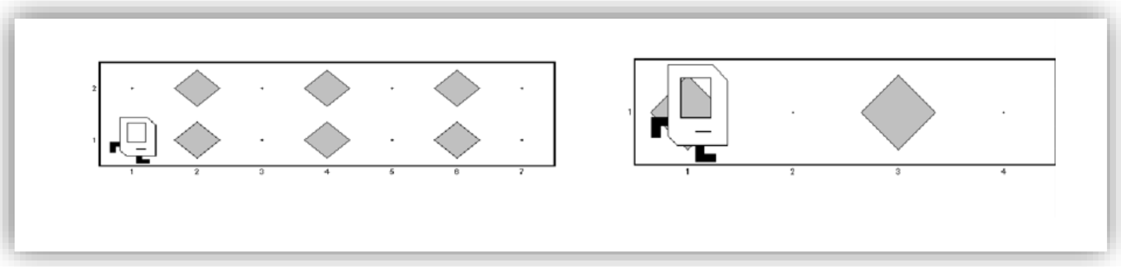

Figure 4: (2,7) and (1,4)

Here below, I have special case according to my algorithm when the map is (2,3) and vice versa. In the first line it will move and put beepers but that doesn't work in the second line so I create *lineNum* variable and assign it to one at the beginning the two in the second line. In the other hand, for (2,6) and vice versa I fill the corner of the map to get four equally champers, also for (2,5) and vice versa I tried to get four equally champers.

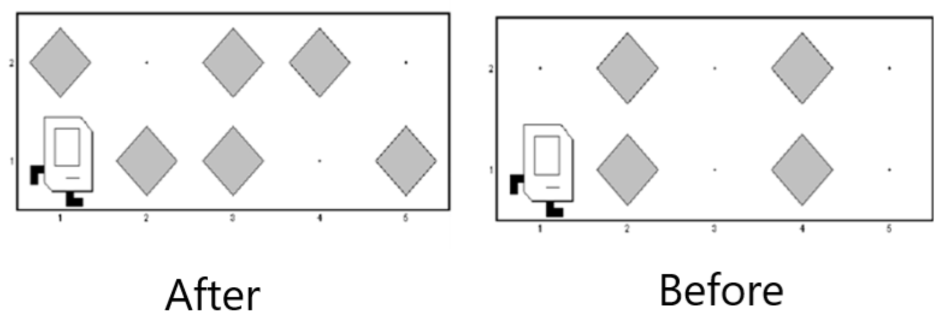

Figure 5: (2,5)

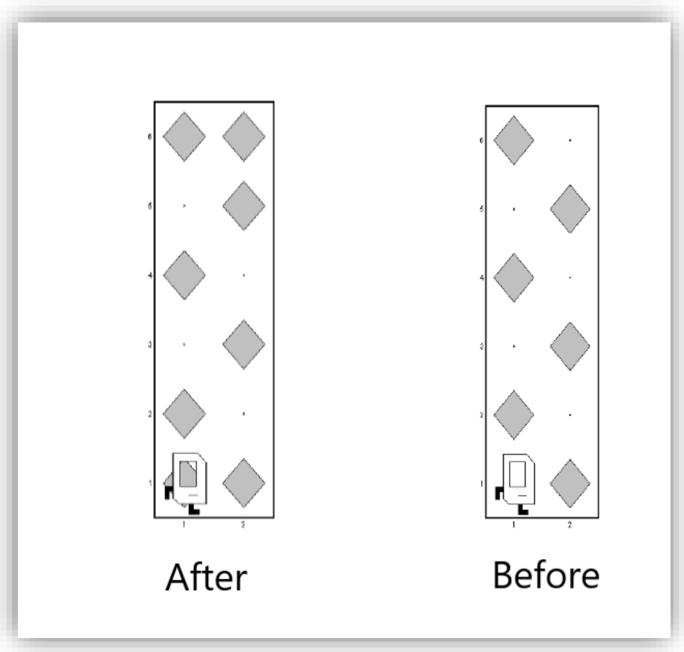

Figure 6: (6,2)

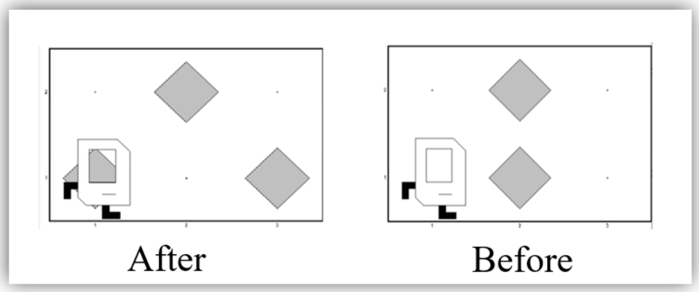

Figure 7:(2,3)

Case 2: square map (x==y) for even axes

I found that in this case if I divide it diagonally it will reduce the number of moves and beepers used.

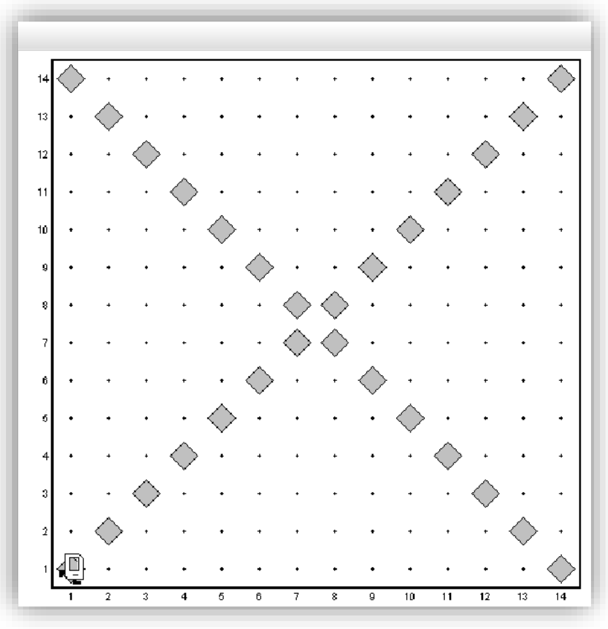

Figure 8: (14,14)

Other four Cases: for axes more than three

Using a single or double line, divide the map in half vertically and horizontally based on whether the axis is even or odd. This gives exactly four champers. If both axis is even, divide using double lines of beepers but I filled them together instead of fill first line then move to the second line this reduce the number of steps to around 10 steps this improvement came at the cost of increasing the code lines by 25 lines.

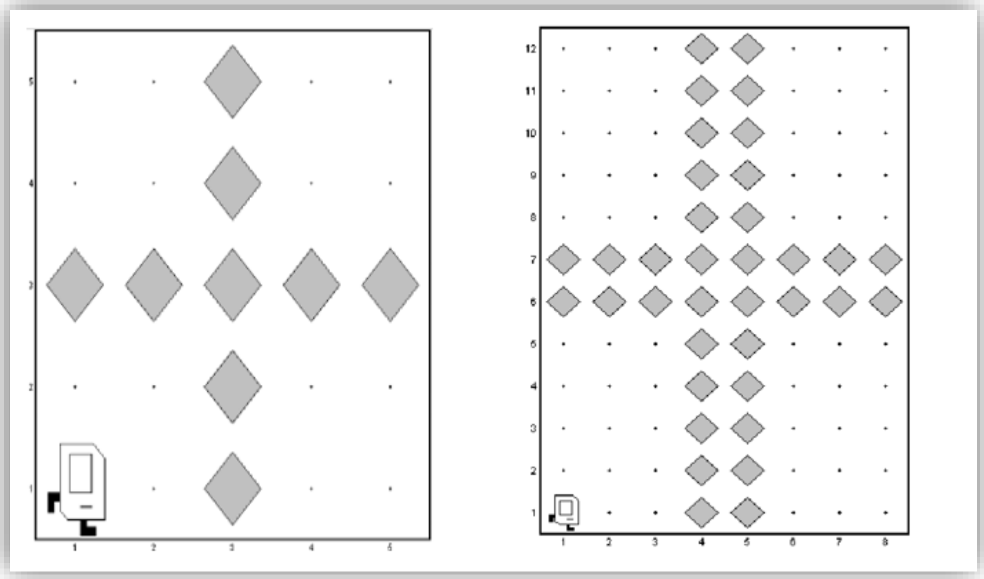

Figure 9: (5,5) and (12,8)

If one of the axes is even, I didn't place double-lined beepers. Instead, I place beepers equal to the result of dividing the odd axis by 4. Then, I check if the odd axis mod 4 is more than one to determine if I should place beepers beside each other by this I reduce number of beepers.

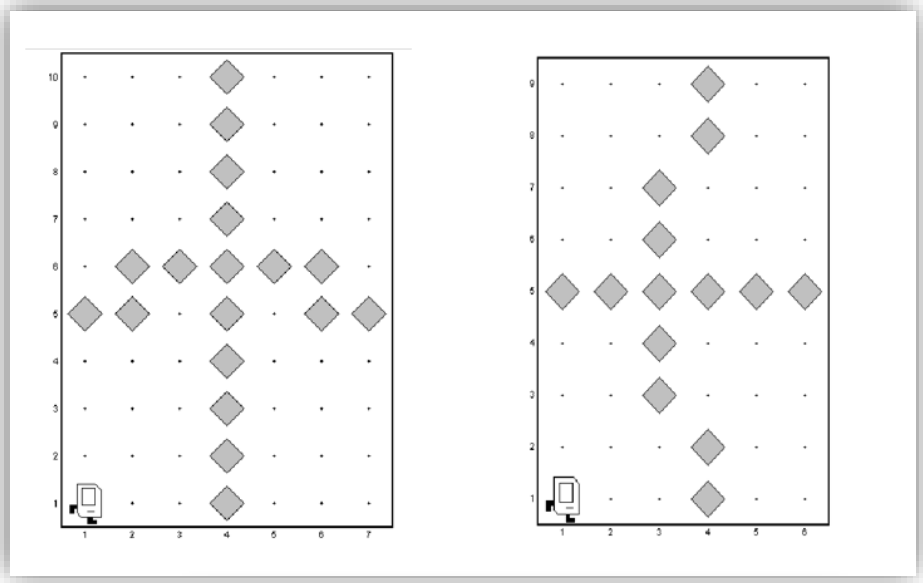

Figure 10:(10,7) and (9,6)

Returning To Origin

After dividing the map Karel returns to its origin point, by calling *returnToOrigin*() method.

Optimizations

In the code that is provided, the main goal of optimization is to divide a map into four equal champers as efficiently as possible while decreasing Karel's total moves, which is the most important goal. Then, the code lines and the beepers count take precedence.

- Reusable functions were written, to reduce the number of code lines.
- I immediately start divide the map after I calculate the dimensions without return to origin to reduce number of steps.
- I choose to divide the even square axes like 10x10 diagonally to reduce number of steps and beepers.
- In case where one of the axes is one or two, I filled the beginning of the line with beepers until I get to the point where the left of the line is divisible by four so I can divide the map into four champers, this method reduces the number of steps.
- In case where one of the axes is two and the other is more than eight, I put beepers for the two lines together instead of fill first one then moves to the second one and fill it, this method reduces number of steps.
- When one of the axes is even, I didn’t put double-lined beepers.