KAREL ASSIGNMENT

Hashem Altabbaa

JULY 30, 2022

ATYPON TRAINING

Supervisors:
Dr. Motasem Aldiab
Dr. Fahed Jubair

Table of Contents

1. Introduction

I was asked to write a program that uses Karel API to divide a given map into 4 equal chambers covering all edges cases such as maps that cannot be divided into 4 equal chambers and divide them into the maximum number of equal chambers. Taking into consideration the complexity of the program that can be assessed depending on the **number of moves** to complete the given task. So, the program should be **optimized** so that Karel the robot can divide the map by the lowest number of moves. Additionally, the written program should be readable, reusable, and number of lines is **minimized**.

2. The approach of solving the problem

Dividing a map into 4 equal chambers depends mainly on the map dimensions. So, it is important to classify the types of maps and how each type will be divided. In this section I will discuss and criticize the approach of solving the problem, for each map type.

I have classified the maps into 4 different categories:

1. Regular maps
2. Vertical & Horizontal maps
    a. Cannot be divided into 4 chambers maps
3. Cannot be divided to any number of chambers.

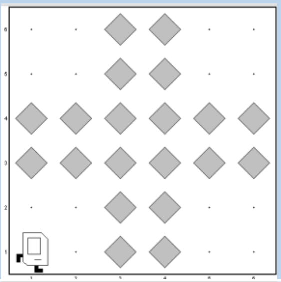
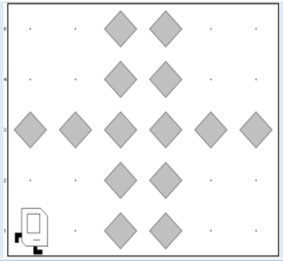
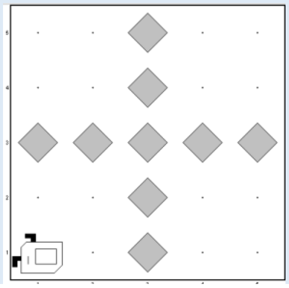
In the following sections I will identify each map and discuss the approach of solving them.

2.1 Regular maps

2.1.1 Regular Maps Identification

Regular maps are those maps that can be divided into 4 equal chambers by drawing a **vertical and horizontal** lines that cut the middle of the map. Regular maps height and width’s **must be greater than 2.** They are divided into 4 sub-categories depending on the dimensions if they are even or odd.

Table 1 Regular maps comparison

Width	Height	How it will be divided?
Even	Even	 Double vertical and horizontal lines
Even	Odd	 Double vertical line and single horizontal line
Odd	Even	Flipped Even-Odd map
Odd	Odd	 Single vertical and horizontal lines

As we can notice the regular maps will be divided by drawing a **single** line from the **odd** dimension side and a **double line** from the **even** dimension side.

2.1.2 Approach of solving regular maps

As an **earlier thought** that came to my mind:

To be able to divide the map into 4 equal chambers, I must calculate both the map’s width **and** height and to do so, I need to move from the point (1,1) to (Width, Height), which is the opposite angel of map. The problem with this process is that I am wasting (Width + Height) steps before doing any progress in the actual mission, which is divide the map.

The optimization of my approach depends on the idea of that **I do not have to calculate both the map’s width and height to start dividing the map.** To start dividing the map by a vertical line, I need only two information.

1. What is the width of the map?
2. Is this map considered a regular map according to my definition (Width and Height > 2) ?

If I knew what the width of a map is, I already know where I am going to cut the map vertically. For instance, if you have been told that the width of a map is 10, assuming that the height of the map is greater than 2 (can be divided by a horizontal and vertical line). Then you can conclude that you will be cutting the map vertically from the positions (5, 6) **regardless of the height value**. So, instead of consuming (Height) steps in calculating the height, I can cut the map from the middle by a vertical line from the position (5, 6) and calculate the height during cutting it. In this way I have combined two steps into one step, which are:

1. Cut the map vertically from the middle after calculating the width and making sure that this is a regular map (can be divided by a horizontal and vertical lines).
2. Calculate the map’s height while cutting it from the middle.

Regular map case pseudocode:

```
width = calculateMapWidth()
isHorizontal = isHorizontalMap()
if(width > 2 and  !isHorizontal)
    goToPoint(width/2+1, 1)  //goToPoint(x,y)
    putBeepers = true
    height = exploremapHeight(putBeepers)
if(width % 2 == 0)
    putBeepersOnColumn(width/2)

if(height % 2 == 0)
    putBeepersOnRow(height/2+1)
    putBeepersOnRow(height/2)
else
    putBeepersOnRow(height/2+1)
```

2.2 Vertical & Horizontal Maps

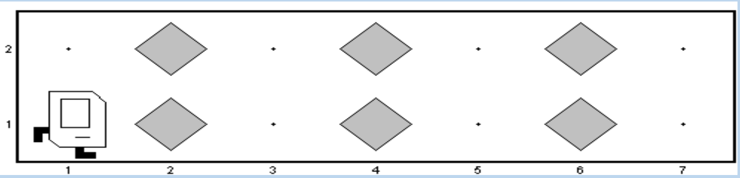
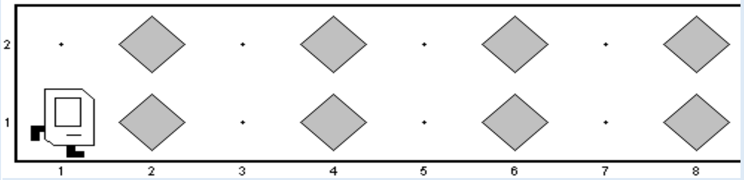
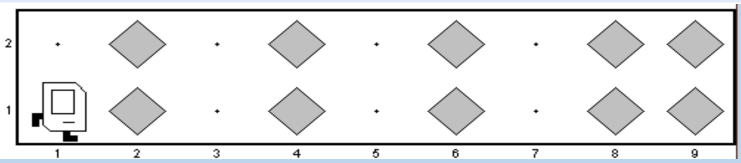
This type of maps must have a width less than 3 **or** height less than 3. This type of maps can be divided into 4 equal chambers, slicing the map into 4 chambers by putting barriers between chambers.

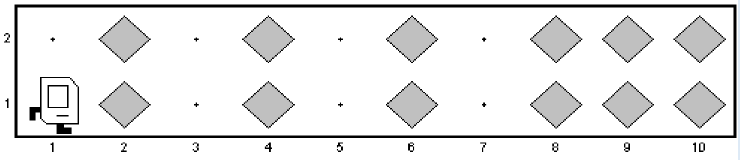
The way I thought about this type of maps is by asking this question:

(If I wanted to divide it into 4 equal chambers, how many barriers do I need to create 4 different chambers?), and the answer was 3 barriers in a vertical or horizontal map will create 4 chambers. So, we can conclude that to divide a horizontal map into 4 equal chambers, the smallest height is 7, because to put 3 barriers of size (one beeper), we will have 4 equal chambers of size (1).

I have categorized the vertical and horizontal maps depending on the result of the following equation : ([Height | Width] – 3) % 4 = result

So, I have 4 different cases that can be handled in simple equation, but it is important to understand the difference between them.

Width	Height	How it will be divided?
Width = 7 (Width – 3) % 4 = 0 4	Height = 2	
Width = 8 (Width – 3) % 4 = 1 5 % 4 = 1	Height = 2	
Width = 9 (Width – 3) % 4 = 2 6 % 4 = 2 4	Height = 2	

Width = 10 (Width – 3) % 4 = 3 10 – 3 – 3 = 4	Height = 2	

Note: This table demonstrated only the case of ((Width > 6) x 2). However, the same pattern applies to any vertical or horizontal map.

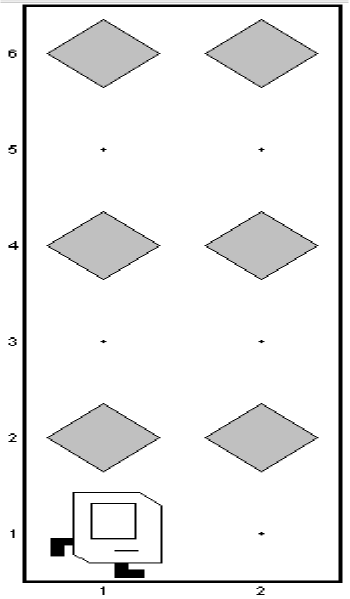
Finally, there is an edge case for the vertical and horizontal maps, which I will explain in the following example. In the beside figure, this is a vertical map with size of (2 x 6)

According to the approach I am using:

Height = 6

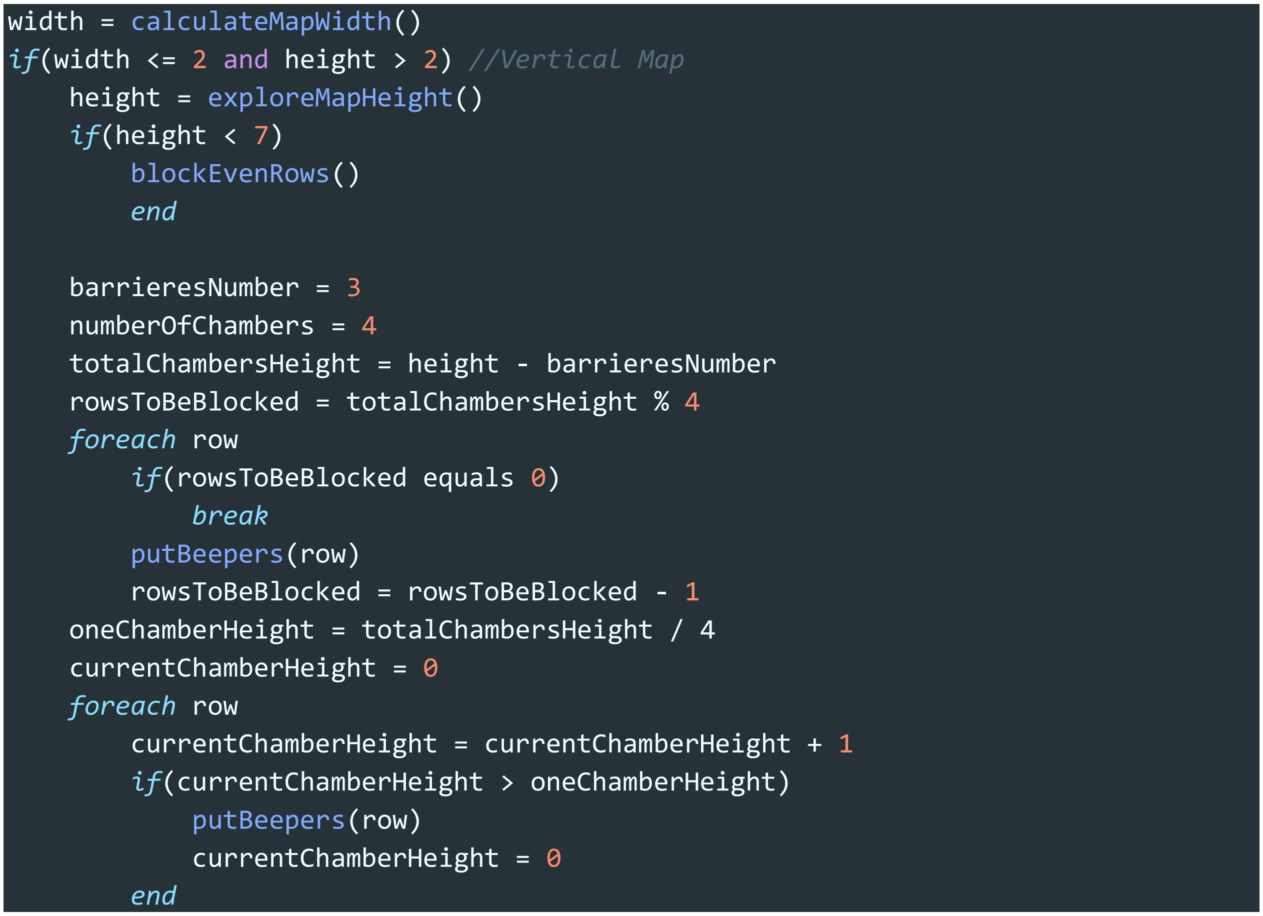
(Height – 3) = 6 – 3 = 3

The total heights of the chambers would be 3, and this cannot be divided into 4 equal chambers. So, to get the maximum number of chambers I block the even rows.



Vertical map pseudocode:

***Note that the pseudocode (approach) applies to the Horizontal maps also.



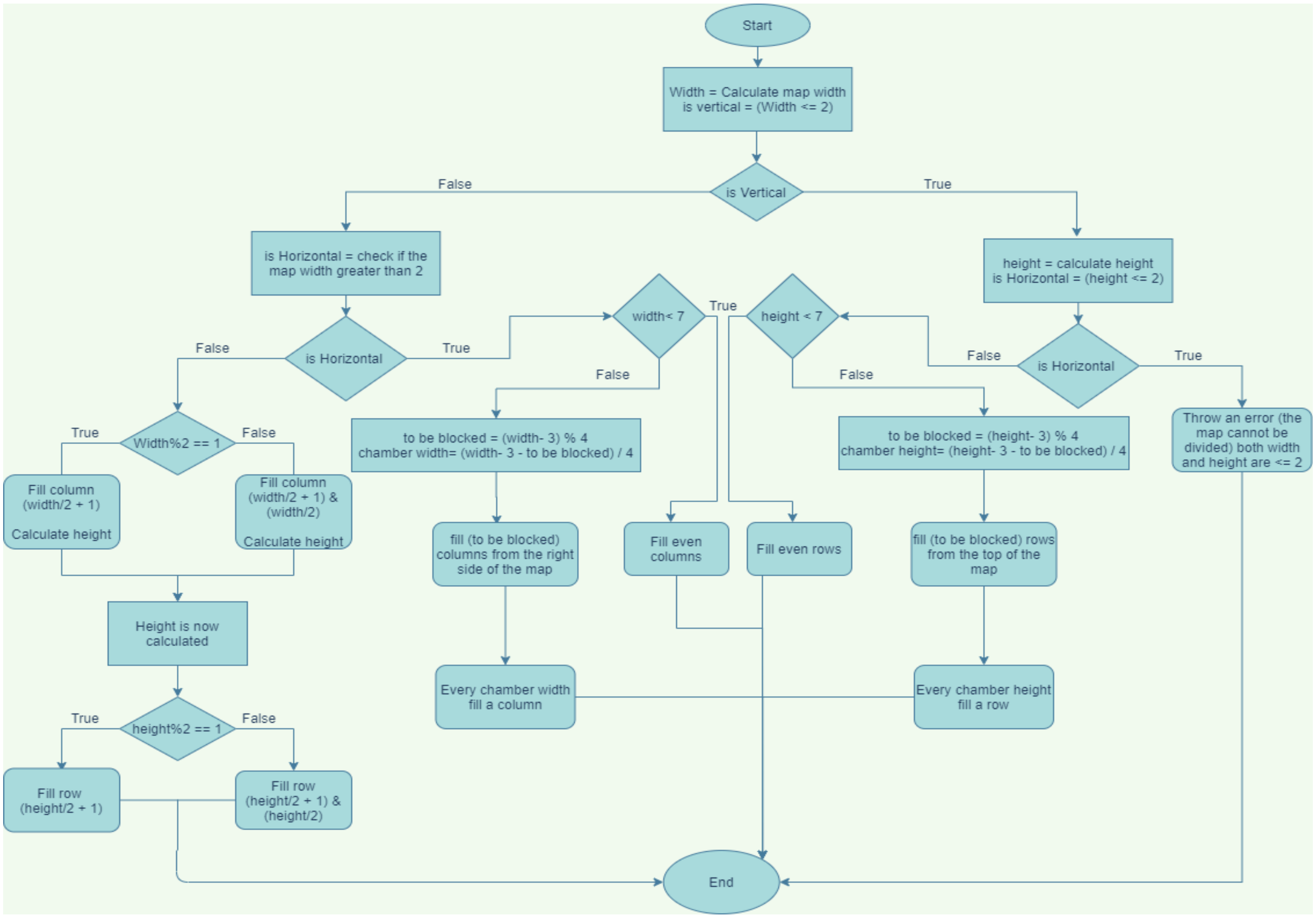
```
width = calculateMapWidth()
if(width <= 2 and height > 2) //Vertical Map
    height = exploreMapHeight()
    if(height < 7)
        blockEvenRows()
        end

    barrieresNumber = 3
    numberOfChambers = 4
    totalChambersHeight = height - barrieresNumber
    rowsToBeBlocked = totalChambersHeight % 4
    foreach row
        if(rowsToBeBlocked equals 0)
            break
        putBeepers(row)
        rowsToBeBlocked = rowsToBeBlocked - 1
    oneChamberHeight = totalChambersHeight / 4
    currentChamberHeight = 0
    foreach row
        currentChamberHeight = currentChamberHeight + 1
        if(currentChamberHeight > oneChamberHeight)
            putBeepers(row)
            currentChamberHeight = 0
        end
```

2.3 Impossible to divide maps

Those are the maps with (height <= 2) and (width <= 2), this type of maps cannot be divided into any number of chambers. So, I throw an exception error for the user.

3. The flowchart of the code


Start
Width = Calculate map width
is vertical = (Width <= 2)
is Vertical
False
True
is Horizontal = check if the map width greater than 2
height = calculate height
is Horizontal = (height <= 2)
is Horizontal
False
True
width< 7
True
height < 7
False
is Horizontal
True
False
False
Width%2 == 1
True
False
to be blocked = (width- 3) % 4
chamber width= (width- 3 - to be blocked) / 4
to be blocked = (height- 3) % 4
chamber height= (height- 3 - to be blocked) / 4
Throw an error (the map cannot be divided) both width and height are <= 2
Fill column
(width/2 + 1)
Calculate height
Fill column
(width/2 + 1) &
(width/2)
Calculate height
fill (to be blocked) columns from the right side of the map
Fill even columns
Fill even rows
fill (to be blocked) rows from the top of the map
Height is now calculated
Every chamber width fill a column
Every chamber height fill a row
True
height%2 == 1
False
Fill row
(height/2 + 1)
Fill row
(height/2 + 1) &
(height/2)
End