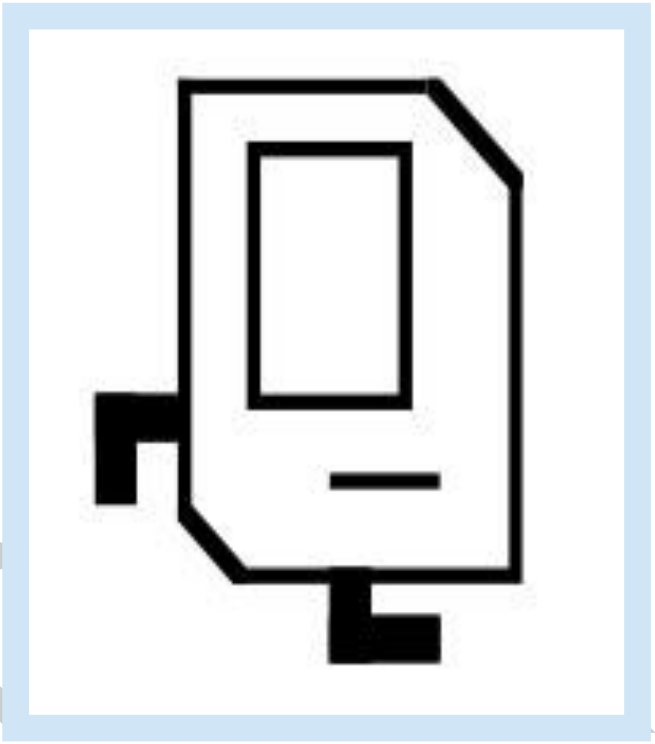


ATYPON

Instructors:
Motasim Aldiab
Fahed Jubair

done by:

Hamza Hassan

KAREL ASSIGNMET

ABSTRACT

The main objective of the report is to find a way to divide the map into 4+4 as show in the diagram below.

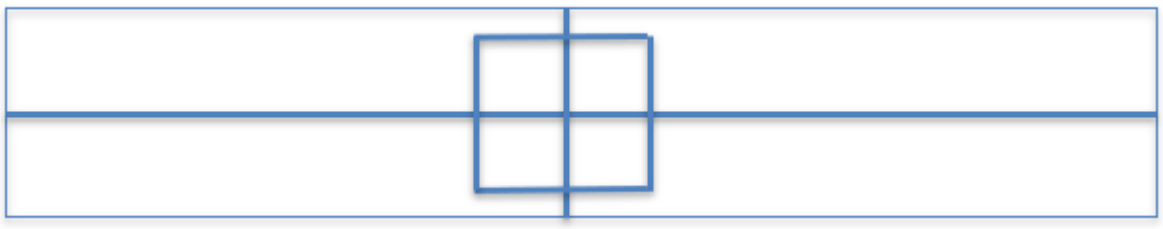


Figure 1

Introduction

This report talk about how can divide any map into required shape and provide the solution for it .This report also aims to highlight the optimizations conducted within the code to reach for the optimal solution.

The problem

The problem addressed by Karel's Assignment is the need to divide a given map of unknown dimensions into 4+4 chambers (if applicable) the outer chamber should equal in size and L-shape, the inner chamber should be the biggest possible equals squares in the lowest number of moves, and the lowest number of lines in code. If dividing the map as required is not applicable, just say maps can't be divided into the required shape because it's too small

1. Finding a way to divide the outer cumbers

Having multiple ways to divide a map into 4 equal chambers, but the question is what is the difference if the all dimension is even or odd or even x odd or odd x even, What is the optimal division that takes the least number of moves? And It represents the general case that applies to all cases to reduce the number of lines in the code Below are a bunch of possible ways to move so we can divide into different dimensions.

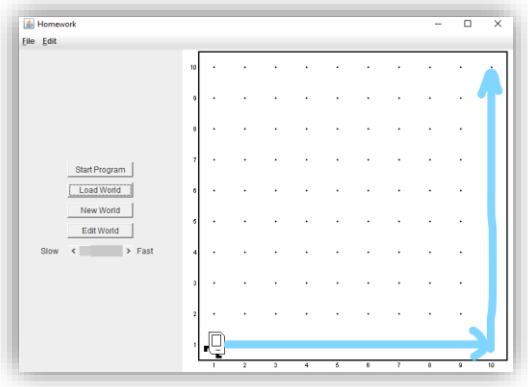
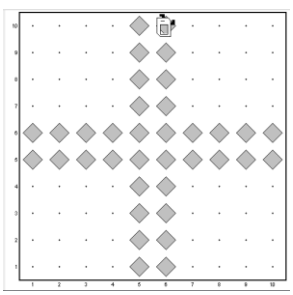


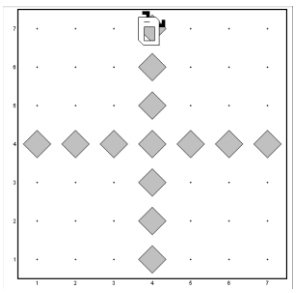
Figure 2

But before any divide, we must know the dimensions by going over all the columns and rows in an L-shape

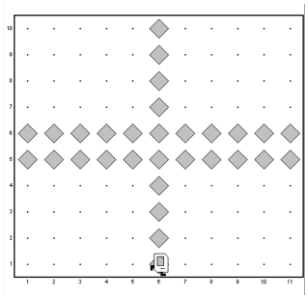
After we know the dimensions (row x column) Now we can see if we can divide the map as required or not by ensuring that the row and column are both <u>greater than 4.</u>
If we can divide, This puts us in front of 4 possible forms of divide



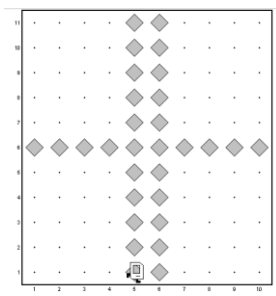
case 1
even x even



case 2
odd x odd



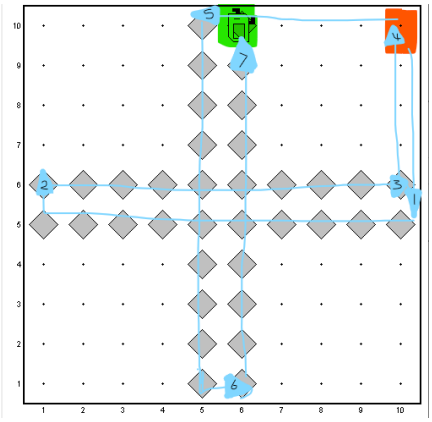
case 3
odd x even



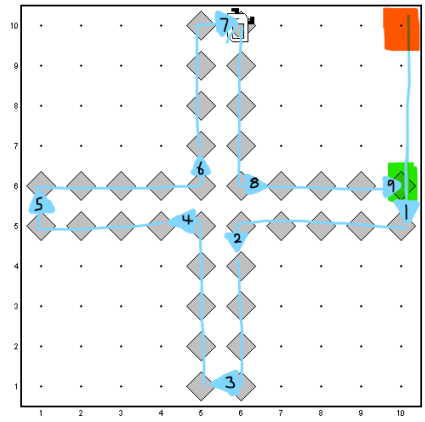
case 4
even x odd

case 1 (even x even) :

there is multiple possible way to draw it, But the least number of steps was done by these two methods.
After get dimension as show in figure 2 go back to mid of the column then start draw



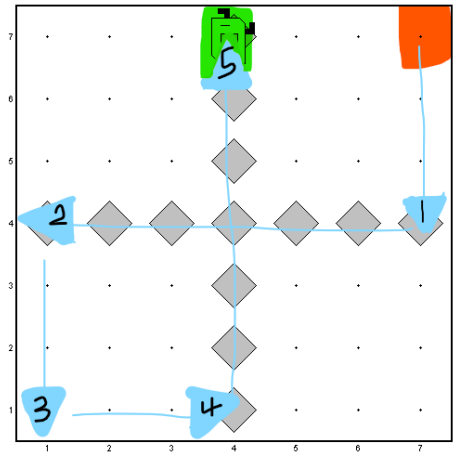
Sol_outer 1
52 moves



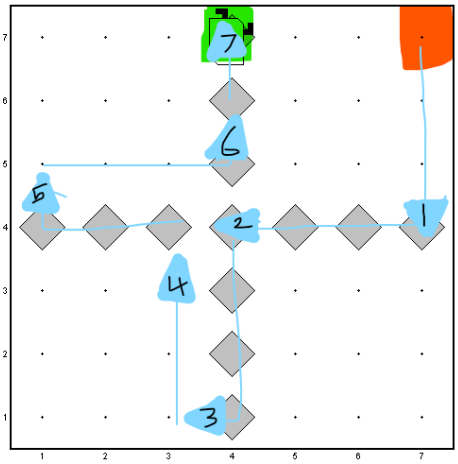
Sol_outer 2
40 moves

As we note, the second method took the least number of steps.

case 2 (odd x odd)



Sol_outer 1
21 moves

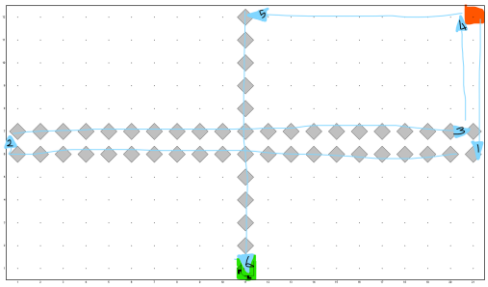


Sol_outer 2
21 moves

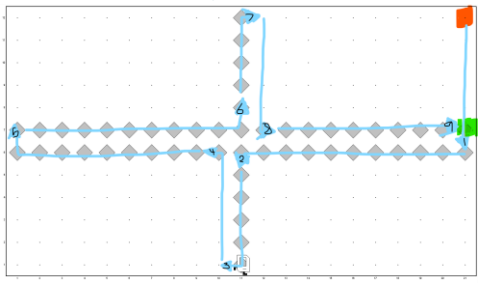
As we note, all method takes the same number of moves.

case 3 (odd x even)

(21x12)

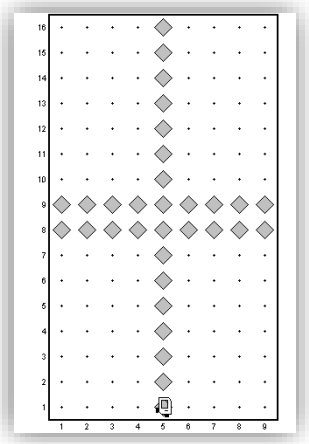


Sol_outer 1
73 moves



Sol_outer 2
67 moves

We still note the second method is better, but what if the column is longer than the row?



Take (9x16) like example:

- In the first method, the number of moves will be 50
- But in the second method it will be 53

This means that the first method will be better and the second method will get worse with increasing column length

Take (7x24) like another example:

- In the first method, the number of moves will be 60
- But in the second method it will be 70

This puts us in front of questioning the ability of Method 2

case 4 (even x odd)

It is exactly the same as case 3 but depends on the length of the row where Sol_outer 2 is still better unless the length of the row is longer than the length of the column

Summary of this problem It seems that we have 6 cases if we take into account the dimension.

The Solution for the first problem:

The best solution I've found after experimenting is to use Sol_outer 1 and ignore Sol_outer 2, for a number of reasons

1. The first reason is that the method contained a bug that would cost a lot of moves as we noticed in case 3 and 4
2. And because it needs more lines and it's hard to write code, because we'll have to switch between and remove Piper mode a lot,
3. In addition to the existence of many special cases and change of direction that must be paid attention to while writing the code

2. Finding a way to create inner cumbers

After completing the problem of divide the outer part, we must move on to divide the inner part, which should be the biggest possible equals squares
This problem contains two subproblems:

- The first:
  We need to figure out where to start division and move Karel to that point
- The second problem :
  it is how do we know what is the size of the biggest square/rectangle that can be drawn to represent with the outer L-shape 4 squares of equal size.

All these problems are interrelated, so let's start by solving these problems.

Solution the second problem:

to solve the first problem and knowing where the point to start division
It depends on where we left off after drawing the outer chamber.
And since I chose Sol_outer 1 to draw the outer chamber, this means that we will always stop at the end of the column, either at the bottom or at the top.
And to solve the second problem and find the size of biggest square that can be drawn
Logically, the smallest dimension in the map should represent one of the dimensions of the square, but we must subtract two from it to maintain a L-shape for outer chamber.

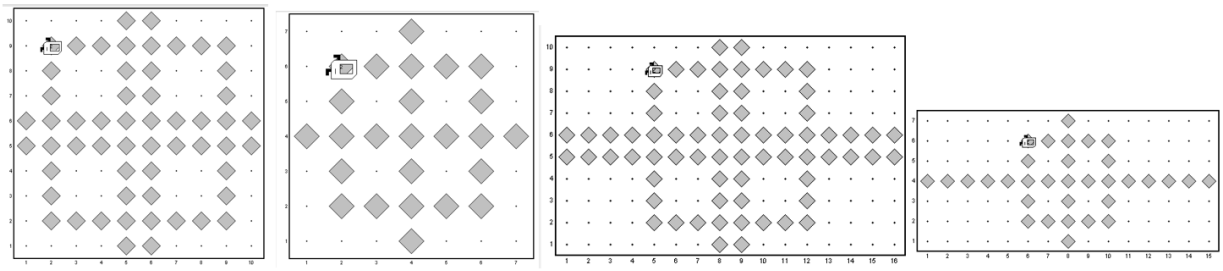
Therefore, I created a method to determine the starting place of the drawing,
called `goToInner(int midOfSmaller, int secondPosition)`
The way it works is to take the middle of the smallest dimension in the map, subtract one from it, and move by that amount
In order to obtain the second dimension, this has cases that we must study

Let's study our possible cases
As it happened earlier, there is a problem with the columns being different from the rows. So let's put all the possible cases and divide them in a smarter way.

Case 1: When the column is equal or smaller than row.

It has 6 forms that can be limited to three sub-cases.

Case 1.1

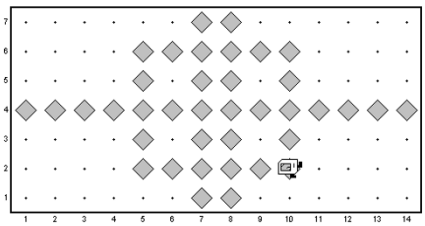


Equal in size diff size

(Even x even) or (odd x odd)

In this case we have the case when all dimension is equal
and when the dimension is (even x even) or (odd x odd)
we notice that it is easy to draw this square since all its sides are equal and
it is equal to the length of the column minus two(column – 2)

Case 1.2

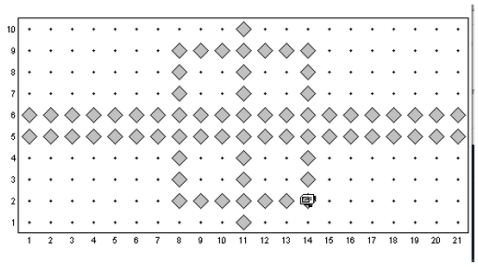


/

(Even x odd)

In this case, we must draw a rectangle whose height is equal to the length of the column minus two(column – 2), and the same for the width, but we must add one for the width to compensate for the decrease that will occur because the column is even and will take two places from the width()

Case 1.3



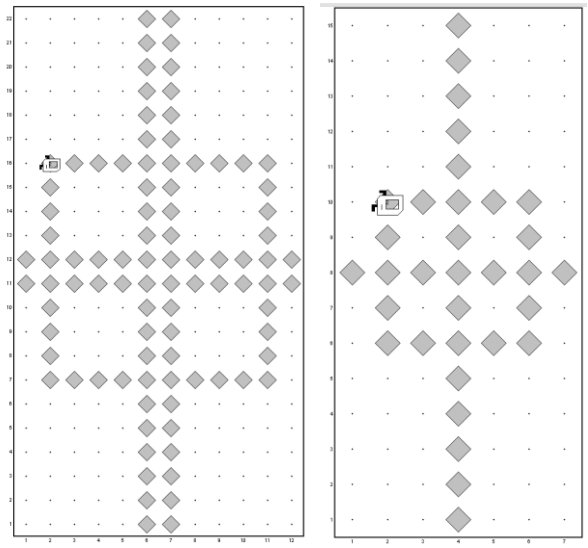
(Odd x even)

In this case, we must draw a rectangle whose height is equal to the length of the column minus two(column – 2), and the same for the width, but we must add one for the height to compensate for the decrease that will occur because the row is even and will take two places from the height(column).

Case 2: When the row is smaller than column.

It has 3 sub-cases.

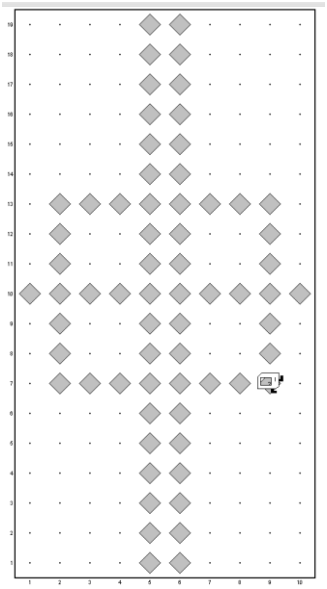
Case 2.1



(even x even) or (odd x odd)

we notice that it is easy to draw this square since all its sides are equal and it is equal to the length of the row minus two(row – 2)

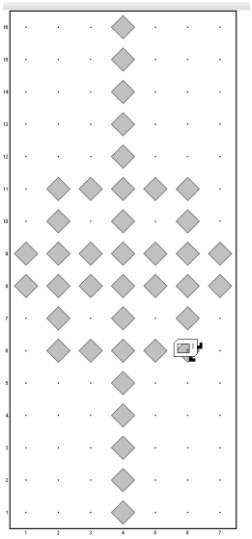
Case 2.2



(even x odd)

In this case, we must draw a rectangle whose height is equal to the length of the row minus two(row – 2), and the same for the width, but we must add one for the width to compensate for the decrease that will occur because the column is even and will take two places from the width(row).

Case 2.3



(odd x even)

In this case, we must draw a rectangle whose height is equal to the length of the row minus two(row – 2), and the same for the width, but we must add one for the height to compensate for the decrease that will occur because the row is even and will take two places from the column.

5 Optimizations

Well, we've come up with good solutions, but how can we optimize?

1. Optimization 1:

After choosing Sol_outer 1 to divide the outer chamber, and we noticed that we have 6 possible cases of division, it was necessary to find a way to write the code to avoid repetition and reduce the number of lines, so I made method to do the division of the outer chamber, and they were as follows.

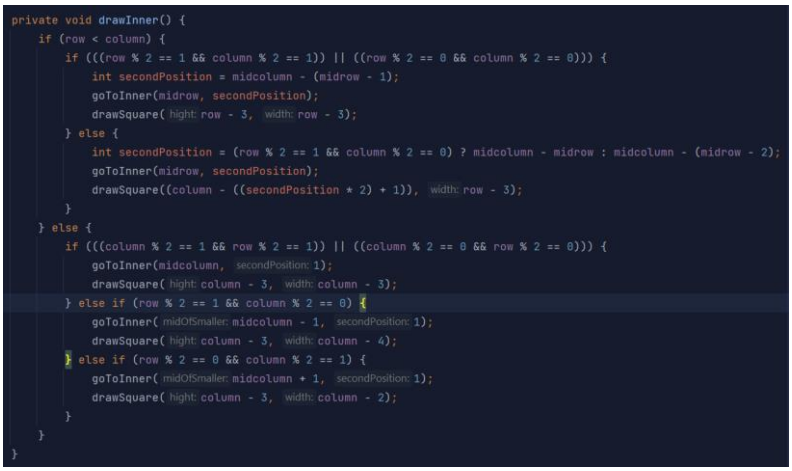
```
private void drawOuter() {
    turnAround();
    move(midcolumn);
    turnRight();
    Outerdivid(column);
    moveToWall();
    turnLeft();
    move(midrow);
    turnLeft();
    Outerdivid(row);
}
```

- divideOuter(): use this method to put a line of Beeper if the input to it is an odd number, and put double lines of Beeper if the input to it is even.
- drawOuter(): Then use this method to control Karel's direction and draw all the outer chamber

and in this way I cover all the cases with minimal number of code and step

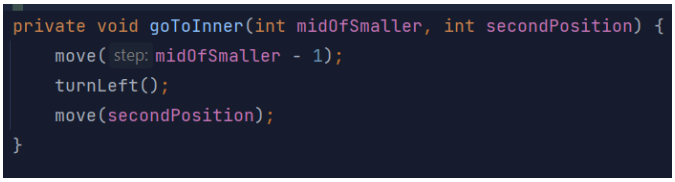
2. Optimization 2:

After we studied all the possible options for divide the inner chamber, it was necessary to find a way to write the code with the least possible number of lines, so I created 2 methods that provided more lines of code

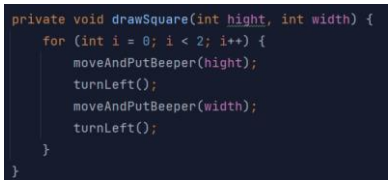


```
private void drawInner() {
    if (row < column) {
        if (((row % 2 == 1 && column % 2 == 1)) || ((row % 2 == 0 && column % 2 == 0))) {
            int secondPosition = midcolumn - (midrow - 1);
            goToInner(midrow, secondPosition);
            drawSquare( hight: row - 3,  width: row - 3);
        } else {
            int secondPosition = (row % 2 == 1 && column % 2 == 0) ? midcolumn - midrow : midcolumn - (midrow - 2);
            goToInner(midrow, secondPosition);
            drawSquare((column - ((secondPosition * 2) + 1)),  width: row - 3);
        }
    } else {
        if (((column % 2 == 1 && row % 2 == 1)) || ((column % 2 == 0 && row % 2 == 0))) {
            goToInner(midcolumn,  secondPosition: 1);
            drawSquare( hight: column - 3,  width: column - 3);
        } else if (row % 2 == 1 && column % 2 == 0) {
            goToInner( midOfSmaller: midcolumn - 1,  secondPosition: 1);
            drawSquare( hight: column - 3,  width: column - 4);
        } else if (row % 2 == 0 && column % 2 == 1) {
            goToInner( midOfSmaller: midcolumn + 1,  secondPosition: 1);
            drawSquare( hight: column - 3,  width: column - 2);
        }
    }
}
```

- goToInner(): This method I talked about earlier is responsible for making Karel go to the place from which the division will begin
- drawSquare(): It is responsible for drawing the four dimensions

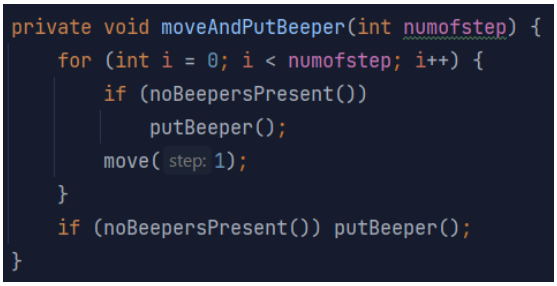


```
private void goToInner(int midOfSmaller, int secondPosition) {
    move( step: midOfSmaller - 1);
    turnLeft();
    move(secondPosition);
}
```



```
private void drawSquare(int hight, int width) {
    for (int i = 0; i < 2; i++) {
        moveAndPutBeeper(hight);
        turnLeft();
        moveAndPutBeeper(width);
        turnLeft();
    }
}
```

3. Optimization 3:
Use the method moveAndputBeeper() to move and put beepers
In a specified number of steps



```
private void moveAndPutBeeper(int numofstep) {
    for (int i = 0; i < numofstep; i++) {
        if (noBeepersPresent())
            putBeeper();
        move( step: 1);
    }
    if (noBeepersPresent()) putBeeper();
}
```

Conclusion:

In this report, we have addressed the problem of dividing a given map into specific shapes, specifically into 4+4 rooms. The goal was to find optimal solutions while reducing the number of moves and lines of code.

We explored different scenarios based on the dimensions of the map and identified different cases, including even x even, odd x odd, odd x even, and even x odd. For each case, we analyzed possible ways to segment the map and determined the optimal approach.

After doing experiments and improvements, we found that using "Sol_outer 1" to divide the outer rooms achieved the best results, then we start to study the inner room and take all cases into account, and we also developed methods to determine the starting point for drawing the inner rooms and to calculate the size of the largest square / rectangle that can be Fits with the L shape.

To improve the code, we have implemented several techniques. This included creating ways to divide the outer rooms and draw the inner rooms, and we created important methods that reduced duplication and reduced the number of lines of code.

In short, this report provides a comprehensive solution to split the map into the required shapes, taking into account different scenarios and optimizing the code for efficiency. By following the strategies shown and using the techniques provided, one can effectively divide the maps into 4+4 rooms as required with minimal effort and code complexity.

The end