# Karel Project

Abdallah Marashdeh

Atypon training program

**Table of Contents:**

## Introduction:

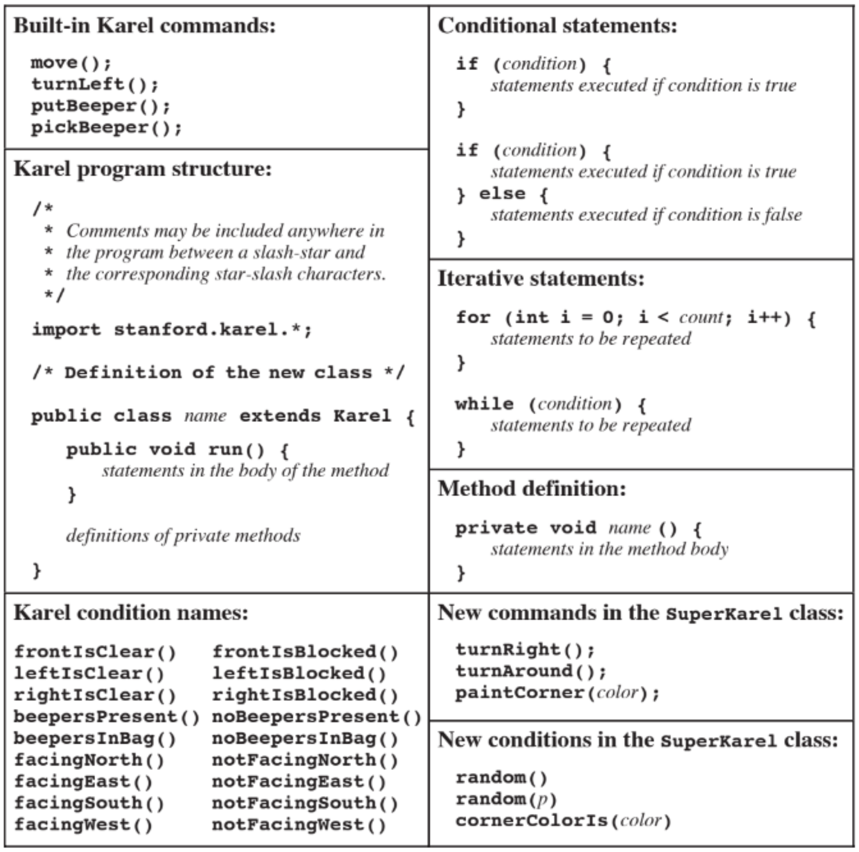
The project was to divide a map with unknown width and height into 4 equal chambers (if possible, if not to divide it to the biggest chambers possible).

This was expected to be done using Karel the robot software made by "Department of Computer Science Stanford University".

We were introduced to Karel by the documents provided by Dr. Motasem,

The documents had the basic functions for completing the project, and we are NOT allowed to work with the classes API to solve the assignment,

The functions that we could work with are only the following:



**Built-in Karel commands:**

```
move();
turnLeft();
putBeeper();
pickBeeper();
```

**Karel program structure:**

```
/*
 * Comments may be included anywhere in
 * the program between a slash-star and
 * the corresponding star-slash characters.
 */

import stanford.karel.*;

/* Definition of the new class */

public class name extends Karel {

   public void run() {
       statements in the body of the method
   }

   definitions of private methods

}
```

**Karel condition names:**

```
frontIsClear()     frontIsBlocked()
leftIsClear()      leftIsBlocked()
rightIsClear()     rightIsBlocked()
beepersPresent()   noBeepersPresent()
beepersInBag()     noBeepersInBag()
facingNorth()      notFacingNorth()
facingEast()       notFacingEast()
facingSouth()      notFacingSouth()
facingWest()       notFacingWest()
```

**Conditional statements:**

```
if (condition) {
    statements executed if condition is true
}

if (condition) {
    statements executed if condition is true
} else {
    statements executed if condition is false
}
```

**Iterative statements:**

```
for (int i = 0; i < count; i++) {
    statements to be repeated
}

while (condition) {
    statements to be repeated
}
```

**Method definition:**

```
private void name () {
    statements in the method body
}
```

**New commands in the SuperKarel class:**

```
turnRight();
turnAround();
paintCorner(color);
```

**New conditions in the SuperKarel class:**

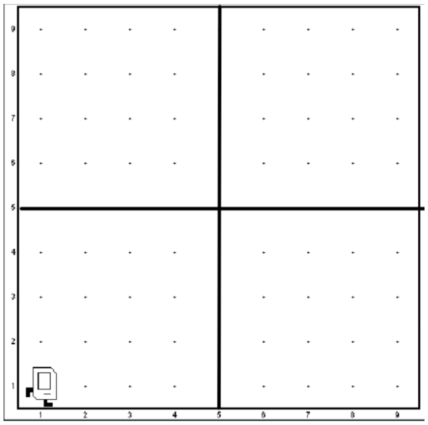
```
random()
random(p)
cornerColorIs(color)
```

## -thought process:

I can describe my thought process as three stages:

-**stage one:**

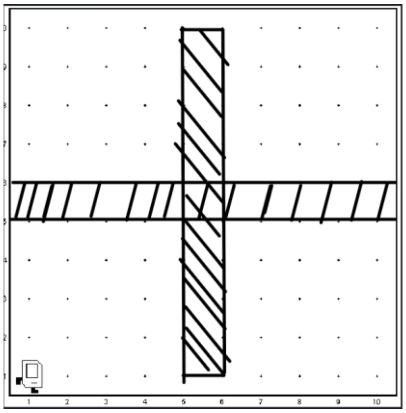
The first thing that came to my mind was to divide it as a cross like this:

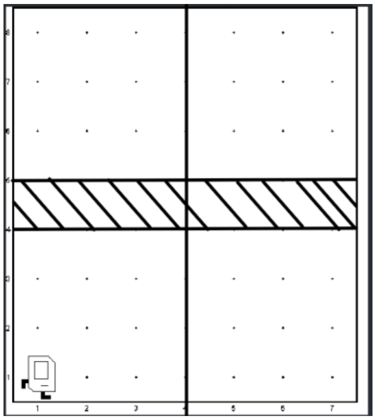


But then I tried in other width and height examples, so I found the following:

- With odd width and height, we only need two lines.
- With odd X even and vice versa we need three lines.
- With even width and height we need four lines.

As you can see below:



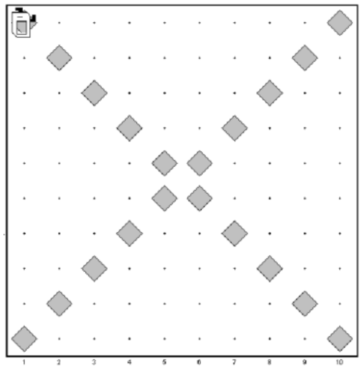


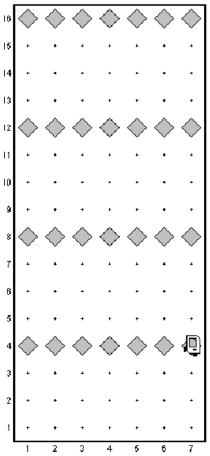
**Stage two:**

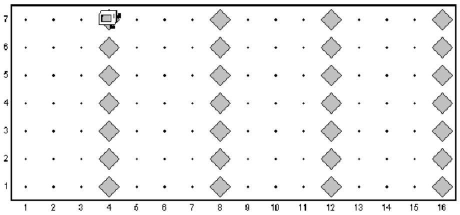
I tried to get better results both in moves and number of beepers.

So, I searched on every way you can divide a rectangle into 4 equal parts and tried to implement them in our cases.

I was left with good results:

- For even equal width and height I improved the division to use diagonal instead for four lines in cross as follows:
- 

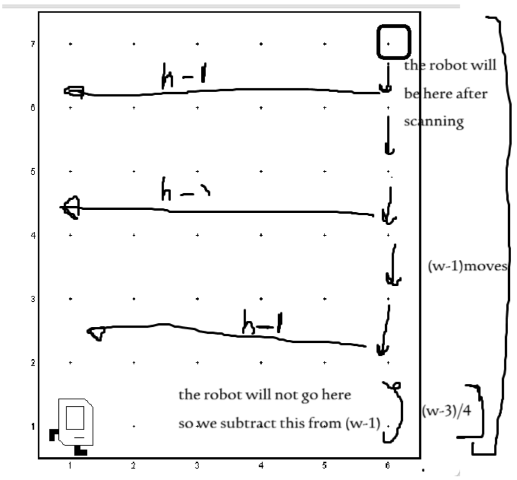
- I found two other ways to cut odd rectangles that in some cases get better results than the cross method, its dividing them to rows or columns like:
- 



I was left with a problem which is how to know if in a certain case the cross is better than the row or the column,

So, I came with the idea of doing a formula for each one and then comparing, the that gives the least number of moves ill choose over the other.

To construct each formula, I used this method:


7
6
5
4
3
2
1
1
2
3
4
5
6
h-1
h-1
h-1
the robot will be here after scanning
(w-1)moves
the robot will not go here so we subtract this from (w-1)
(w-3)/4

if we sum all the moves (without the moves for scanning the map):

Row = (3*(h-1)+((w-1)-((w-3)/4))+((w-3)%4)*(h-1))

And since if we flip it, the rows become columns we just flip the width with the height for columns.

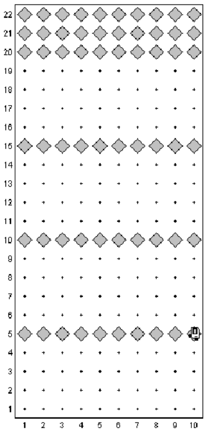
And I found four formulas for cross:

Two lines (both are odd)=2*(width-1)+3*(height-1)/2

Three Lines (width is even )= 2*(width-1)+5*(height-1)/2

Three Lines (height is even )= 3*(width-1)+(height-1)+(height/2)

four Lines (both are even )=3*(width-1)+2*(height-1)+(height)/2 of course one of the restrictions for the col and row method is that it needs to be more than 7 (since at least we need three lines to divide  and one line at each division ) and we need to fill the lines that damage the structure of equality like this example:



We filled the first three lines so the rest of the divisions can be equal.

**-stage three**

In this stage the main point was to know which maps are not divisible to four chambers and which are not divisible at all :

So I found that the following sized can’t be divided at all

1X1,1X2,2X1

Where 2X2 can be divided only diagonally  into two parts.

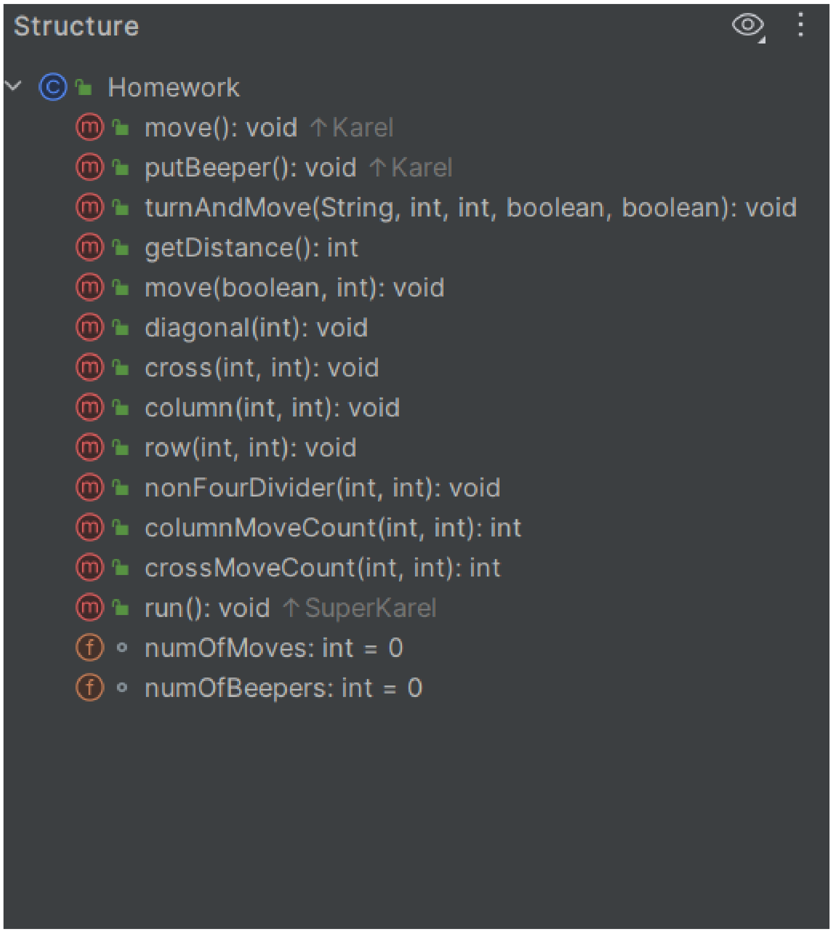
For the case of :

(number less than 7)X( 1 or 2 ) or ( 1 or 2) X (number less than 7)

Like the 3X2 or 5X2 : These can divided wither by 2 or 3 partitions

## functions used:

I used several functions I’ll list them below.



Structure

Homework
- move(): void ↑Karel
- putBeeper(): void ↑Karel
- turnAndMove(String, int, int, boolean, boolean): void
- getDistance(): int
- move(boolean, int): void
- diagonal(int): void
- cross(int, int): void
- column(int, int): void
- row(int, int): void
- nonFourDivider(int, int): void
- columnMoveCount(int, int): int
- crossMoveCount(int, int): int
- run(): void ↑SuperKarel
- numOfMoves: int = 0
- numOfBeepers: int = 0

-move (): overrides the move function just to count the moves at every move

-putBeeper (): overrides the putBeeper function just to count the beepers

-turnAndMove (): this function is done to reduce redundant code, I investigated the code and saw that a multiple lines were used a lot in really small differences.

-get Distance (): is used primarily for scanning the map and returning move count - (width or height )

-move(Boolean, int): is and overloading function of the move() function, where it moves for x number of times if flag is true, it will put beepers with every move

-diagonal(int): this function divides the map diagonally

-cross (width, height): this function divides the map in the cross shape (single or double lined)

-Column (width, height): this function divides the map into columns

-Row (width, height): this function divides the map into rows

-NonFourDivider (width, height): this function divides the map into the maximum number of chambers.

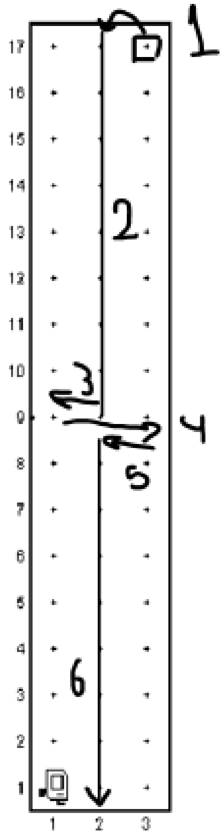
-crossMoveCount (): this function returns the number of moves needed to divide the map in cross shape

-columnMoveCount (): this function returns the number of moves needed to divide the map into columns or rows (because they are the same if width is swapped with height)

I’ve tried to make the code as short and simple as possible, it was about 320 lines and I managed to shorten it to about 190 lines of code!

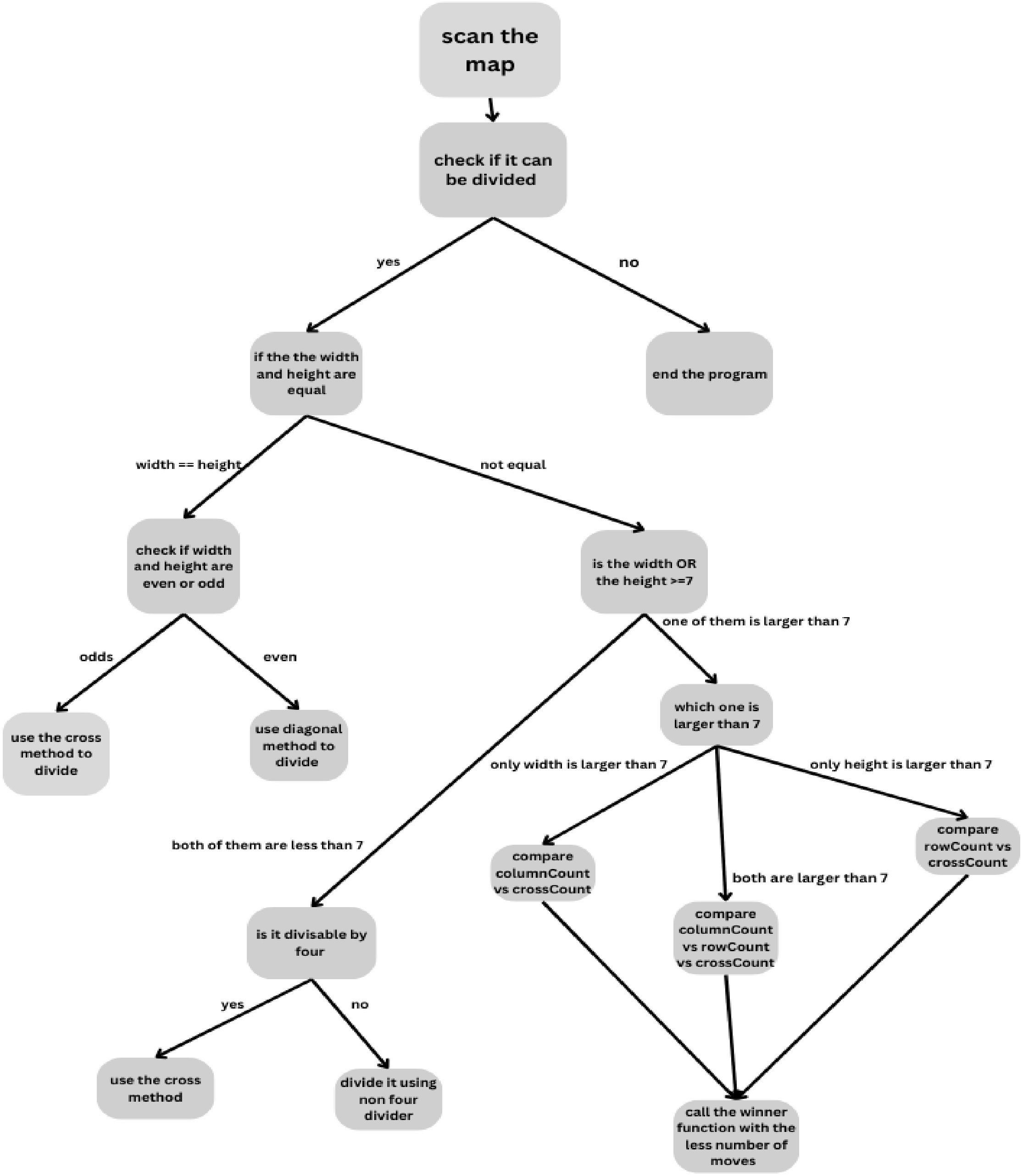
## Worthey mentions:

I also found a way to make the robot move less in some extreme cases like below:



In this manner it will walk 21 moves and put 19 beepers and the row method will move 23 moves with only 15 beepers (in larger scales the difference in beepers number would be significantly larger but the move count is still a small difference ) , I thought this wouldn’t be considered as optimization since we care about both moves and beepers .

## diagram explaining the process:


scan the map
check if it can be divided
yes
no
if the the width and height are equal
end the program
width == height
not equal
check if width and height are even or odd
is the width OR the height >=7
one of them is larger than 7
odds
even
use the cross method to divide
use diagonal method to divide
which one is larger than 7
only width is larger than 7
only height is larger than 7
both of them are less than 7
compare columnCount vs crossCount
compare rowCount vs crossCount
both are larger than 7
compare columnCount vs rowCount vs crossCount
is it divisable by four
yes
no
use the cross method
divide it using non four divider
call the winner function with the less number of moves

**special thanks to Dr.Motasem,**