ATYPON

Training Program
Assignment #1
Karel Robot Report

Instructor: Prof. Motasem Al-Diab

Done by: Ahmad Al-Sou'b

➢ Abstraction:

Karel is a programming model designed for beginners to develop algorithms using Java. And in this report, I will explore the report the task of programming Karel the Robot to divide a map into four equal chambers using beepers. If dividing into four isn't possible, the program ensures the map is divided into the largest possible number of equal chambers—three, two, or one. The algorithm also tracks and displays Karel's movements in the terminal, with Karel returning to its starting position at (1,1) after completing the task.

➢ Three levels of optimization were required:

- Minimum Number of moves optimization.
- Minimum Number of Beepers used optimization.
- Minimum Number of code lines optimization.

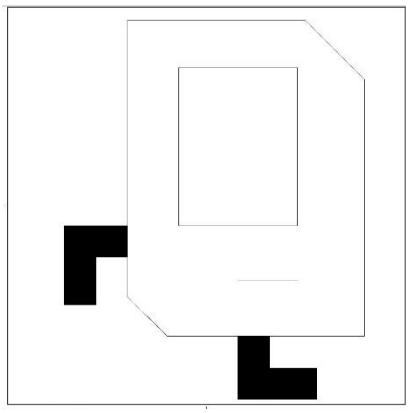
➢ Main Idea:

My approach to solving this problem was based on identifying the general cases and then using a trial-and-error method in the optimization process. In the end, I concluded that there are three main cases.

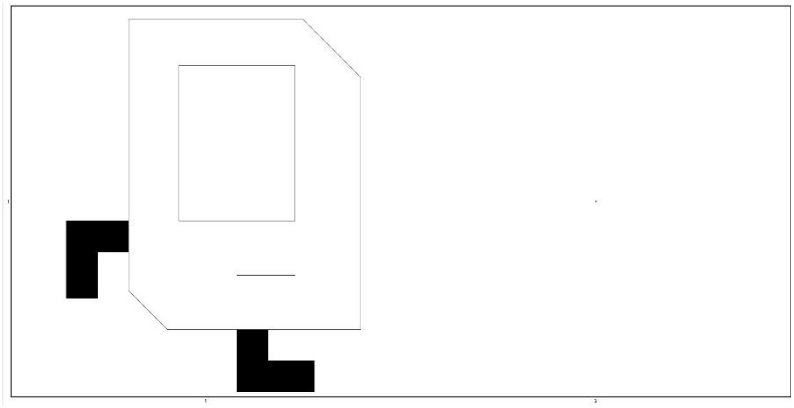
This approach covered all possible cases with the fewest number of steps.

1. **Length and Width more than Two:** The most general case where both the length and the width are greater than 2, regardless of whether they are equal or not.
2. **Length and Width are both Equal Two.**
3. **Length or Width is less than or equal Two:** The case where the length or the width is 2 or less.

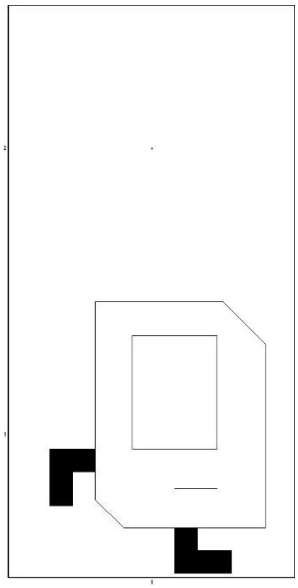
- **Initially:** We have three maps that we cannot divide so we'll keep it as it is.



A. 1x1



B. 2x1



C. 1x2

➢ The functions built inside the SuperKarel class that we will use:

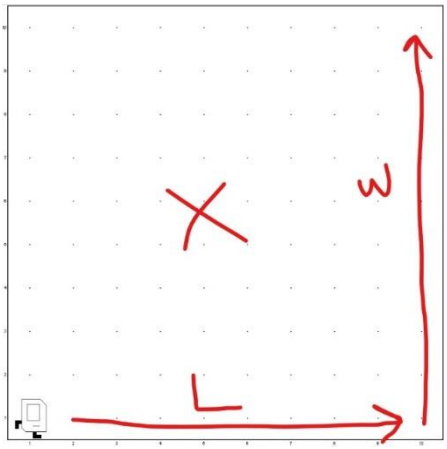
1. **move()** 2. **turnRight()** 3. **turnLeft()** 4. **paintCorner()** 5. **facingEast()**

6. **pickBeeper()** 7. **leftIsClear()** 8. **turnAround()** 9. **putBeeper()**

10. **frontIsClear()** 11. **frontIsBlocked()** 12. **noBeepersPresent()**

13. **setBeepersInBag(1000)**

➢ **Initial Step:** I used the API to set the initial value of the beepers. A general beginning for every cases, this step is important to measure the dimensions using a defined function.

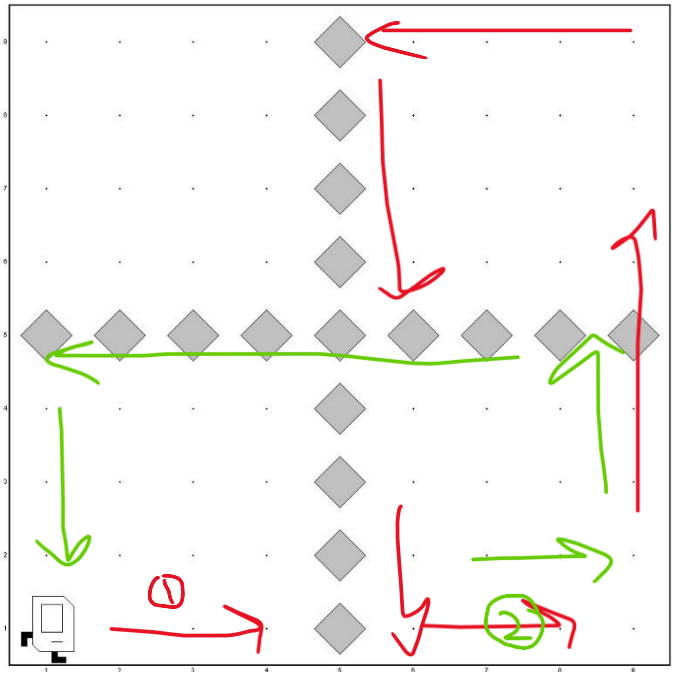


➢ Length and Width more than Two (>= 3):

1. **If both dimensions were 3 or higher:**

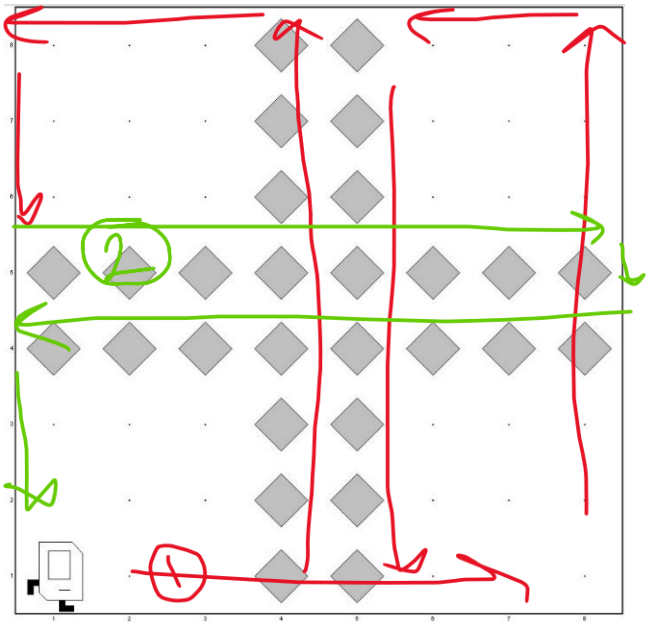
   This solution is used to always create four champers, It means each dimension can be divided into two equal parts and from here we have to approaches:

2. **If the length of the dimension was an odd number -> draw ONE line perpendicular from the center.**

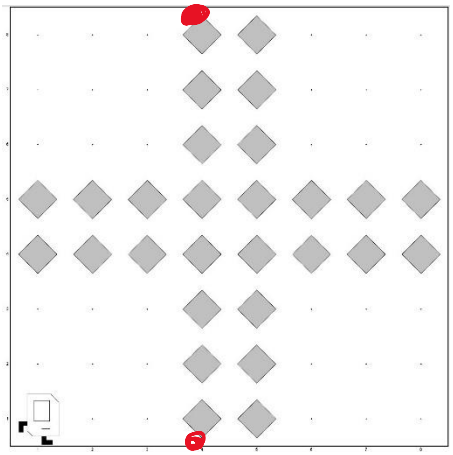


- Here this solution costs us 17 beepers (width+height-1). It's divided the map to the biggest 4 equal chambers with minimum number of beepers.

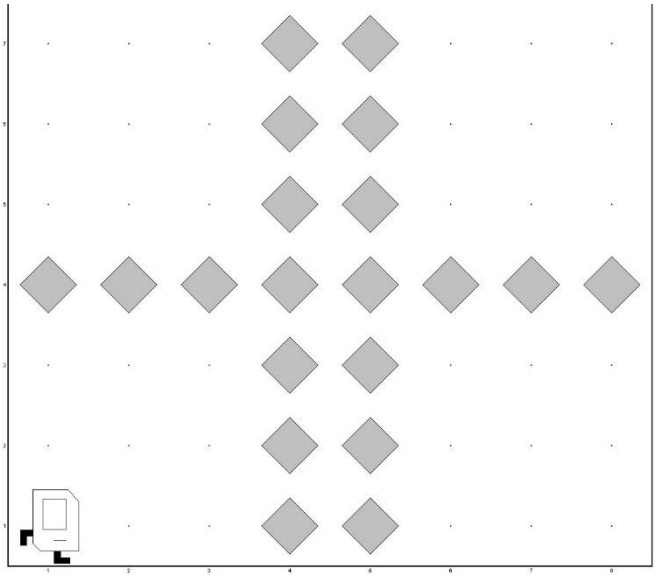
3. **If the length of the dimension was an even number -> draw TWO lines perpendicular from the center.**



- The equation below represents this point location. point = length + 1 /2.



4. **If one of the lengths of the dimension is even and the other is odd -> draw two lines at the even dimension and one line at the odd dimension.**

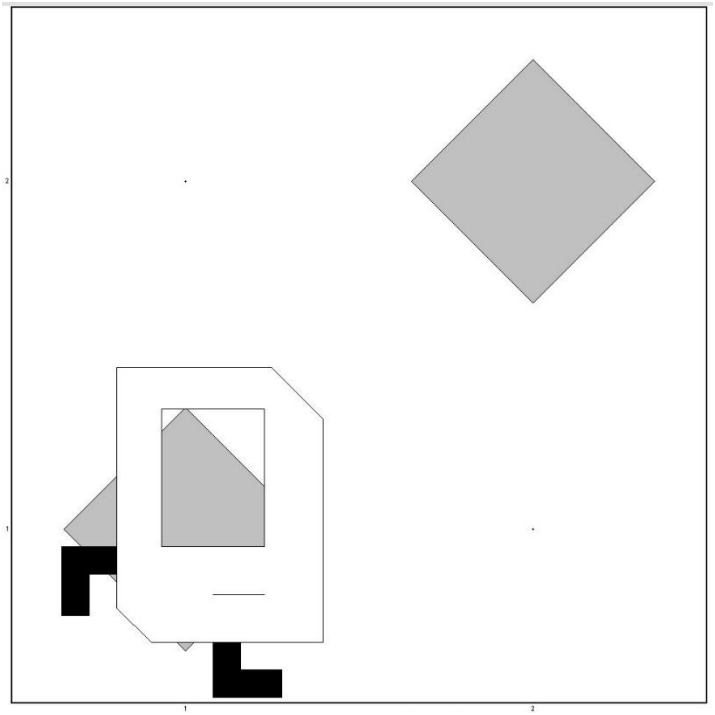


➢ **The methods used in these cases are:**

- **Private void divide2D(int x):** this method is called when both dimensions are 3 or higher so it divide the map as mentioned before.
- **Private void moveLineAndPutBeeper():** this method does the same as the previous one but it also put a beeper if there is no one while moving.

➢ Length and Width are both Equal Two:

This is an exceptional case, first because it is impossible to divide the map into four equal squares, and second because it requires us to choose only one shape to divide the map into the largest possible number of squares, which is the diagonal shape.



➢ The methods used in these cases are:

putBeeper();

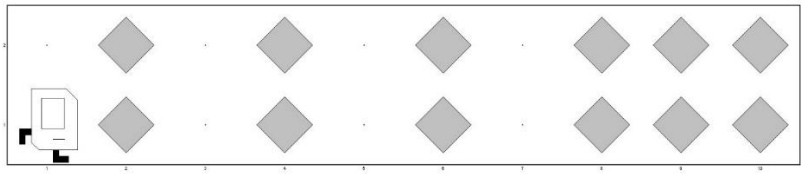
Private void moveToOrigin(): returns Karel to the origin point.

➢ Length or Width is less than or equal Two:

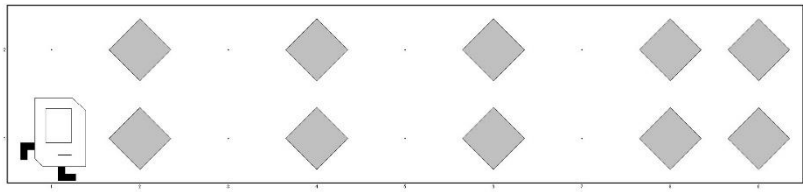
- It means that the other dimension is either 1 or 2 and can't be divided, so in this case we have to check if the other dimension can be divided into the maximum number of chambers starting from 4.
- I have reached these conclusions:
  - In many maps, Karel starts plotting from some displacement (not measured yet) of its initial location.
  - Whenever the biggest dimension is in the set {10, 14, 18...} the map will contain consecutive two beepers.
  - If the biggest dimension is in the set {9, 13, 17…} Karel will start from where it's placed in from the initial step (measuring dimensions step).
- This approach is achieved mainly by getDivideInfo (int Dimension , int numOfPieces) which takes two integers as mentioned and returns an array that contains useful info about how the map should be divided.

➢ Some of the cases:

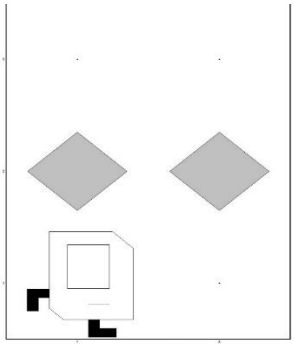
1. **10x2**



2. **9x2**



3. **2x3**



The methods used in these cases are:

- **Private int [ ] getDivideInfo(int dimension , int numOfPieces):** this method take these arguments and return an array that contains the number of units of each area, the number of extra beeper (the extra beeper that the robot will put them as a walls to provide the maximum number of champers) and the maximum number of champers that can be divided into.
- **Private void divideALine(int numOfUnitsInEachArea , int numOfExtraBeeper , int numOfPieces , int length):** this method divide a line by using the information in its parameters.
- **Private void divide1D(int x):** this method is called if only one of the dimensions the 3 or higher, It calls the two methods above to achieve the required.
- **Private void putAnotherBeeper():** this method is used to put another beeper behind the other one (this method is only called when one of the dimensions is 2).

Helper Methods:

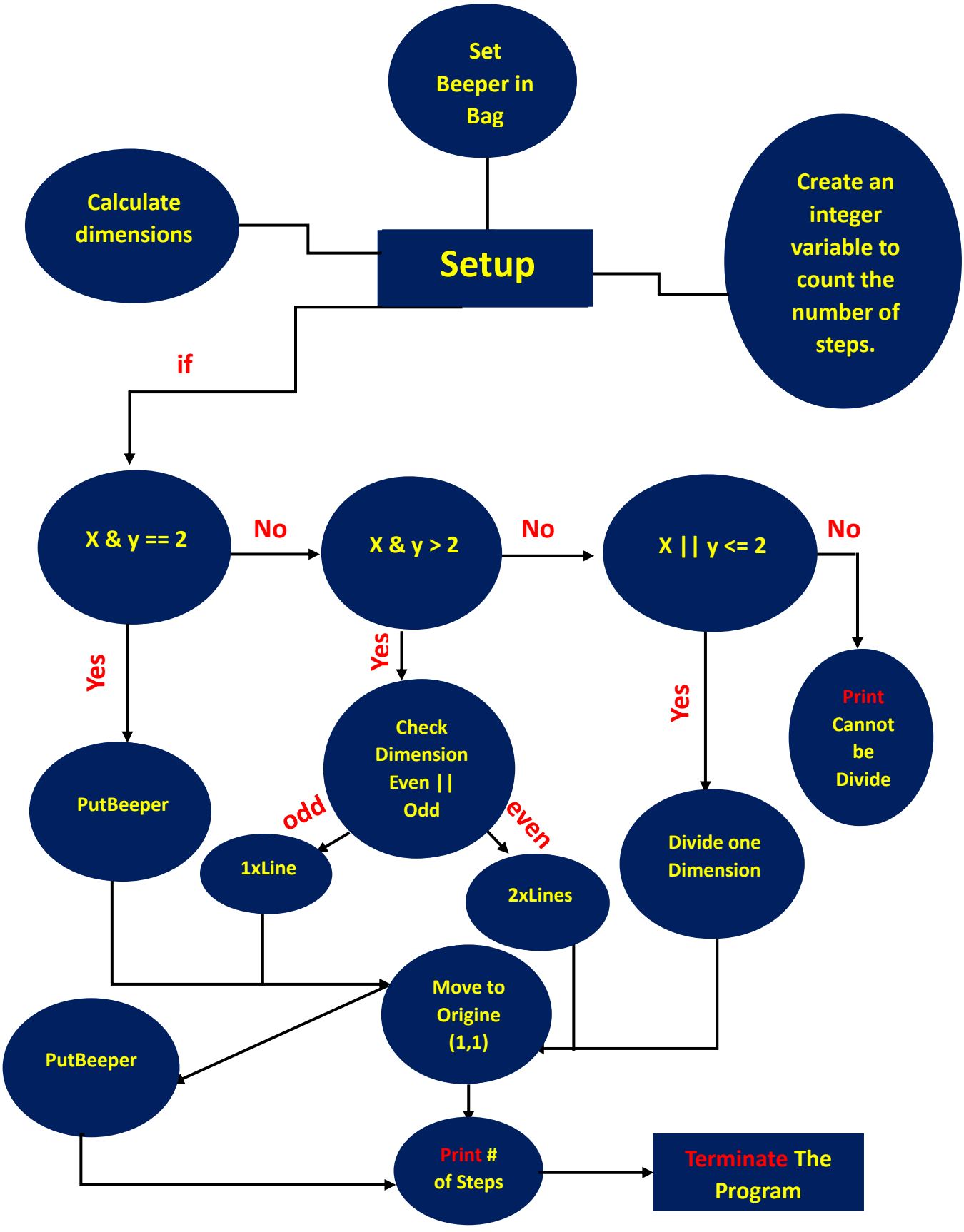
I have defined some methods to modulate my code:

- **public int[] calculateXAndY():** I used this method to calculate dimensions of any map and return them in an array.
- **Override move parent method:** This method calls the original transfer method to update the transfer counter.

```
@Override
public void move(){
   super.move();
   movesNumber++;
}
```

- **Private int moveLine():** this method moves until Karel faces a wall and return the number of steps (it is used to calculate the dimensions).
- **Private void facingRight()**
- **Private void moveIfClear()**

➢ **Flowchart:**


Set Beeper in Bag
Calculate dimensions
Setup
Create an integer variable to count the number of steps.
if
X & y == 2
No
X & y > 2
No
X || y <= 2
No
Yes
Yes
Yes
Print Cannot be Divide
PutBeeper
Check Dimension Even || Odd
odd
even
1xLine
2xLines
Divide one Dimension
Move to Origine (1,1)
PutBeeper
Print # of Steps
Terminate The Program

➢ Conclusion

In conclusion, this problem involves handling different cases based on the size of the map. Depending on the map's dimensions, we choose the most suitable method. For a 2 × 2 even square map, we draw a diagonal. For maps with equal dimensions, we draw a (+), and if one of the dimensions is even, we draw a (+) with double lines. Lastly, if either dimension is 1 or 2, we use a different method for division.

The End…