KAREL ASSIGNMENT

Map Division Algorithm with Karel

Done by:

Mohammad Alsabi

Supervisors:

Dr. Motasem Aldiab

ABSTRACT

This report introduces and examines a Java-based algorithm designed for the division of maps within the Karel programming environment. The algorithm, embedded in the provided code, systematically divides maps into equal chambers based on their dimensions. The subsequent sections will provide an overview and analysis of the algorithm, focusing on its application and functionality.

ACKNOWLEDGMENT

I would like to express my appreciation and gratitude to everyone that supported me and provided me with the help I needed on my journey to finish this project.

I would also like to give special thanks to my project’s supervisors, **Dr. Motasem Aldiab**, for his continued help and support.

Table of Contents

List of Figures

Table

CHAPTER ONE: Algorithm and Thinking

1.1 Algorithm

- Set the number of beepers in the bag to 1000.
- Get the x and y dimensions of the map.
- Divide the map into equal chambers based on the x and y dimensions.
- If both x and y are greater than or equal to 3, where any map like it will sure be divided into four equal chambers by cross boundary shape:
    - If both x and y are even:
        - Divide the map vertically into two halves.
        - Fill beepers half of the y-axis.
        - Fill beepers half of the x-axis.
        - Move to the lower row.
        - Fill beepers half of the x-axis.
        - Fill beepers half of the y-axis.
        - Move to the left column.
        - Fill beepers half the y-axis.
        - Fill beepers half of the x-axis.
        - Move to the upper row.
        - Fill beepers half of the x-axis.
        - Fill beepers half of the y-axis.
    - If x is even, divide using double lines of beepers:
        - Divide the map vertically into two halves.
        - Fill beepers along the y-axis.
        - Move to the other side.
        - Fill beepers along the y-axis.
        - Return.

- If x is odd:
    - Divide the map vertically into two halves.
    - Fill beepers along the y-axis.
    - If y is even:
    - Divide the map horizontally into two halves.
    - Fill beepers along the x-axis.
    - Move to the other side.
    - Fill beepers along the x-axis.
    - Return.
- If y is odd:
    - Divide the map horizontally into two halves.
    - Fill beepers along the x-axis.

- If either x or y is greater than 6 (special corner case I named it the bar):
    - Flip the map if necessary, so I deal with the bar horizontally.
    - Fill extra beepers to optimize four-part division.
    - Fill beepers by dividing the map by the optimize step.
- If the map is a 2x2 grid, place beepers in the corners, (this step is just to handle a corner case).

    - If one of the dimensions is less than or equal to 2, fill beepers accordingly:
        - Flip the map if necessary, so I deal with the bar horizontally.
        - Fill beepers by 1 in the direction of the larger dimension.
        - If x > 2, place a beeper, (this if statement is just to handle the case 1x2 and 2x1 where these can't be divide).
- Output the total moves made by Karel.

CHAPTER TWO: Optimization

The primary focus of optimization in the provided code revolves around achieving the most efficient division of a map into four equal sections while minimizing the total number of moves made by Karel, which is the highest priority.

Then the priority comes to both the lines of the code and the total placement of beepers.

2.1 Main Goal Optimization

It was done by thinking of two cases of dividing the map, first is the case where the map can be divided with the cross shape starting with the minimum map size 3x3 and for more optimization I give it an easy way to get four equals by splitting every axis in half, if odd then one line to achieve the minimum moves, in the other hand the even length will have more moves by putting two lines in the middle but it will sure to split equally. In addition, when the map axes are even x even then karel will walk complete each half separately, so the moves will be minimized.

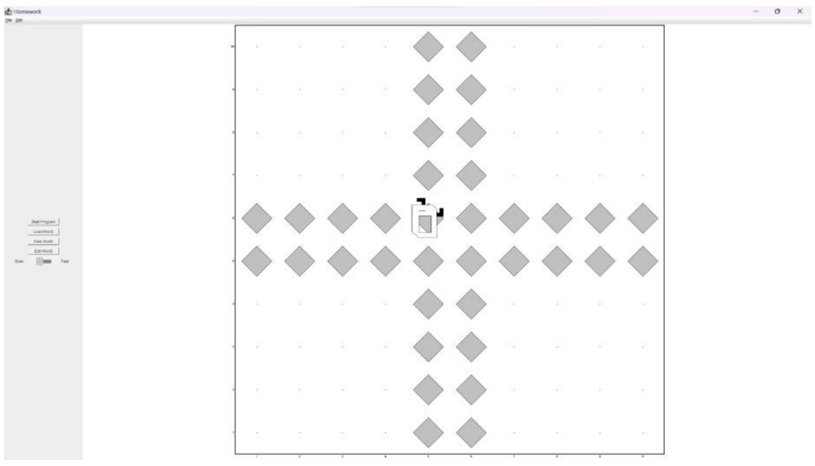


Figure 1: Even x Even

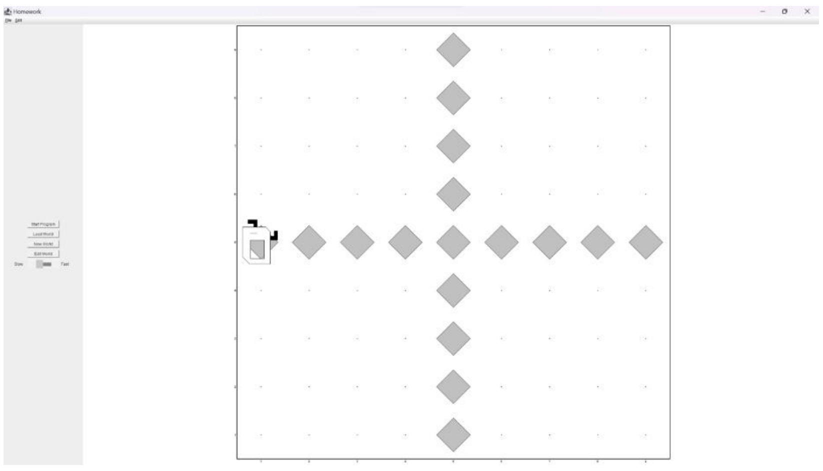


Figure 2: Odd x Odd

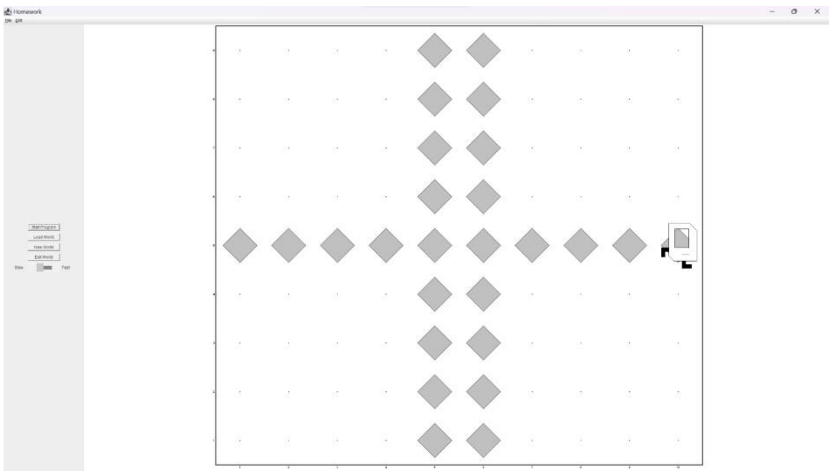


Figure 3: Even x Odd

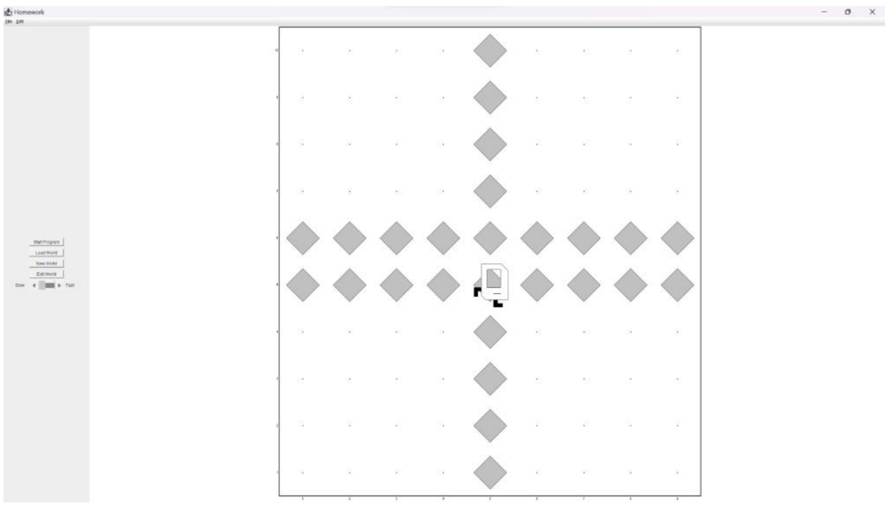


Figure 4: Odd x Even

The second case, which I called the bar case, is handled by placing three walls in a horizontal view with the right step length to split equally, and fill the extra area, that is the (axis-length - 3) %4, with beepers. It can be only done if the horizontal view length is more than six, so that leaves us with dividing the map one after one with beepers where will work fine in splitting the map in this order (3,2,1), but again there is a corner case 2x2 map that can be solve by placing two beepers in any facing corners.

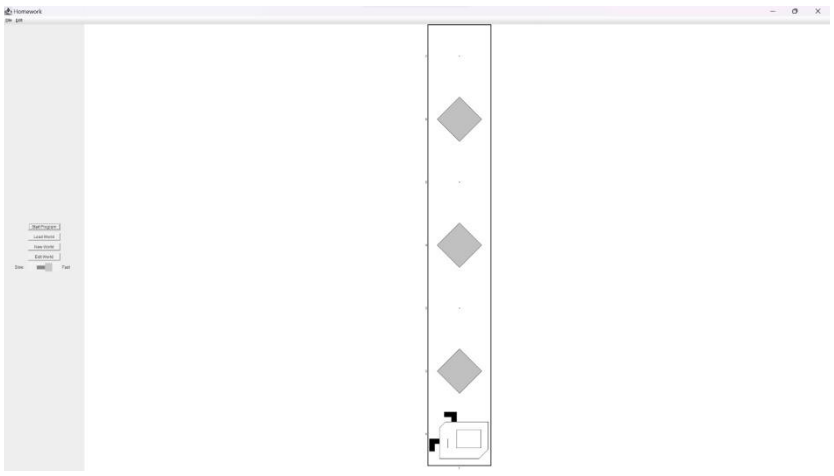


Figure 5: Smallest Bar Map of Four Chambers

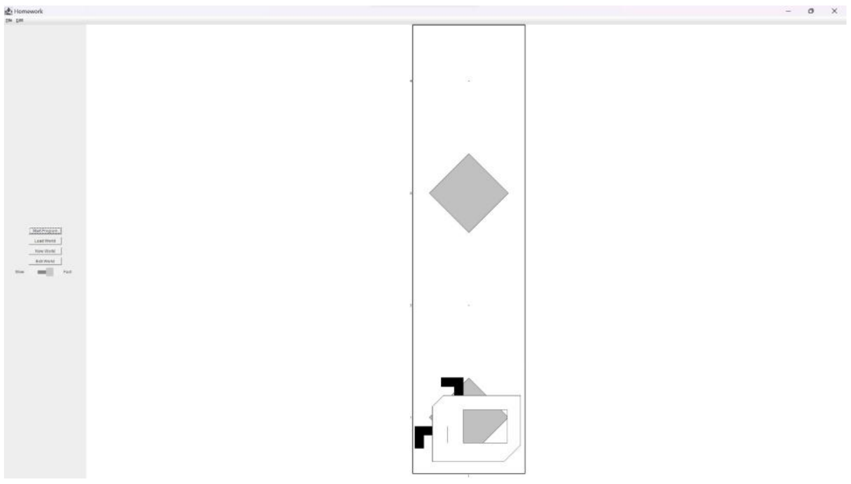


Figure 6: One Case less than Four Chambers

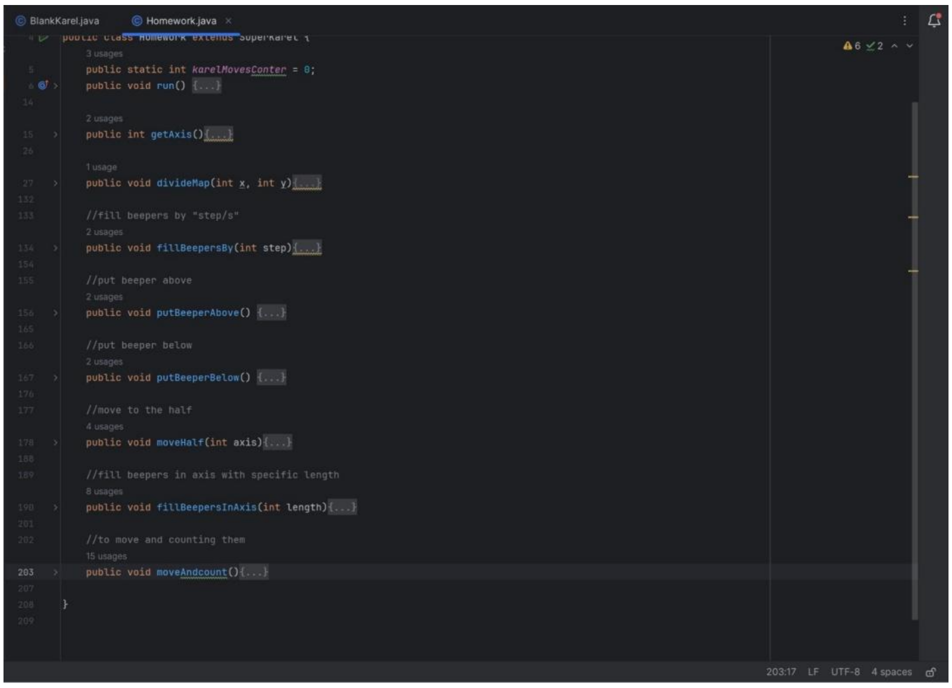
2.2 Code Lines Optimization

The code achieves optimization through the implementation of reusable methods:

Table 1: Reusable Methods

Method name	Number of Calls
fillBeepersBy	2
putBeeperAbove	2
putBeeperBelow	2
moveHalf	5
fillBeepersInAxis	16
moveAndcount	18

Significantly contribute to reducing redundancy in the code. The algorithm strategically uses these methods to perform specific tasks, enhancing readability and minimizing the overall number of code lines.



```java
public class Homework extends SuperKarel {
    3 usages
    public static int karelMovesConter = 0;
    public void run() {...}

    2 usages
    public int getAxis(){...}

    1 usage
    public void divideMap(int x, int y){...}

    //fill beepers by "step/s"
    2 usages
    public void fillBeepersBy(int step){...}

    //put beeper above
    2 usages
    public void putBeeperAbove() {...}

    //put beeper below
    2 usages
    public void putBeeperBelow() {...}

    //move to the half
    4 usages
    public void moveHalf(int axis){...}

    //fill beepers in axis with specific length
    8 usages
    public void fillBeepersInAxis(int length){...}

    //to move and counting them
    15 usages
    public void moveAndcount(){...}

}
```

Figure 7: Reusable Methods

2.3 Beepers' Number Optimization

The usage of beepers was done by checking if the spot is empty to place one else we don't, and if the map is a bar-case then place a vertical line of beepers that can't exceed two, finally using the cross shape rather than walls, where walls will always take 4 * x, or y axis-length, but the cross will be (x-axis + y-axis) x 2 in worst case which is better.

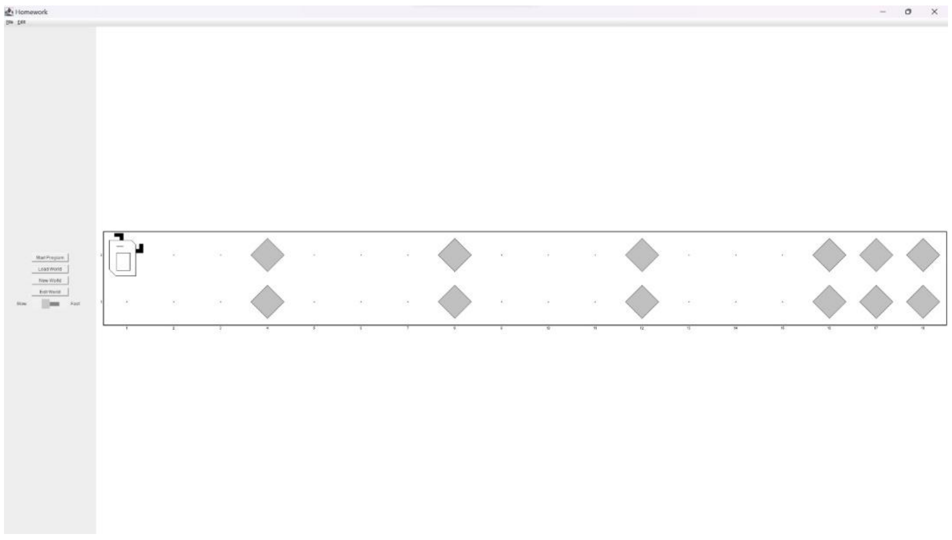


Figure 8: Compromising (Four Chambers over Number of Beepers)