



# Karel Assignment Report

## TABLE OF CONTENTS

<b><i>Abstract</i></b>	<b><u><a href="#">3</a></u></b>
<b><i>Section 1: Code Structure</i></b>	<b><u><a href="#">4</a></u></b>
<b><i>Section 2: Level 1 Overview</i></b>	<b><u><a href="#">8</a></u></b>
<b><i>Section 3: Level 2 Overview</i></b>	<b><u><a href="#">9</a></u></b>
<b><i>Section 4: Level 3 Overview</i></b>	<b><u><a href="#">10</a></u></b>
<b><i>Section 5: Limitations &amp; Flaws</i></b>	<b><u><a href="#">11</a></u></b>
<b><i>Section 6: Further Notes</i></b>	<b><u><a href="#">12</a></u></b>

# Abstract

This document will describe and study the requested assignment that revolves around Karel and its basics commands, as well as inspect the three levels.

We will showcase a screenshot of the project itself, as well as some code snapshots. Potential flaws will be discussed, please note that there are some comments in the code which will further aid the understanding of the workflow.

## Section 1: Code Structure

The main class will define global variables in which it will be used for the three levels.

```
//A manually-set flag that determines whether to navigate Karel to the start point in a simple maze or to just return to the south-west corner
static final boolean EXPLORE=false;

//Global variables which is shared by the three functions
boolean returning=false, iterator=true;
int beepersPut=0;

//Variables allocated for the first function
boolean singularHeight=false, stopWidth=false;
int cornersVisited=0;

//Global variables that stores Karel's position as well as the board's proportions
int height=1, width=1, coordinateX=1, coordinateY=1;
```

The run() function is set to call three functions respectively, each function covers one level, and implicitly sets the variable to its default values, as well as moving Karel to the origin point (facing east at the southwest corner) so that the next function will work properly.

However, Karel may not be at the default position/direction, hence, findStartPoint() and returnToStartPoint() will be handy. The first function will navigate to the start point by exploring the world and borders, this is useful in simple mazes. The second function will simply relocate to the start location directly without exploration, so if there was a box within the world in which Karel is inside it, findStartPoint() should be called. You can toggle between exploration and directly traversing to the start point by changing the EXPLORE variable, see [Limitations & Flaws](#) section.

```
public void run(){
    //Some maps may set default beepers to 0, which may cause an error when putting one, hence, it should be set to a high value
    setBeepersInBag(9999);

    if(EXPLORE)
        findStartPoint();
    else
        returnToStartPoint();

    //Executes the requirements accordingly
    exeLv1();
    exeLv2();
    exeLv3();
}
```

The `findStartPoint()` function moves Karel to a border first (by calling `stickToBorder()` function), then, it keeps rotating around the border until it finds an opening right of it, if it exists, it will go straight right until hitting another border. The function will keep doing this until the previous width and height are equal to the current value. Since Karel is at the outer box, it will call `returnToStartPoint()` function.

```
//A function that allows Karel to exit a simple maze, then reaching the south-west corner
//TODO function does not work on some mazes and may not exit it properly
private void findStartPoint() {
    /*
    // width      : the width of the box in which Karel is currently in
    // height     : the height of the box in which Karel is currently in
    // widthPrev  : the previous width of the box in which Karel was previously in
    // heightPrev : the previous height of the box in which Karel was previously in
    //
    // This function will record these variables and breaks whenever the heightPrev/widthPrev is equal to height/width (but not equal to one)
    // and then calls the returnToStartPoint() function, returning karel to the default start point
    */
    int width=1, height=1, widthPrev=1, heightPrev=1;
    //A flag which iterates between whether to start counting the height or the width
    boolean countingTurn=false;
    //Ensures that Karel is by a side
    stickToBorder();

    //A loop that runs indefinitely until the mentioned variables are equal, while so, it will attempt to go right if there is an opening in the box
    while(true) {
        if(frontIsClear()) {
            if(rightIsClear() && !backIsBlocked())
                turnRight();
            if(frontIsClear()) {
                if(countingTurn) width++;
                else height++;
                move();
            }
        } else {
            if(widthPrev==width && heightPrev==height && (width!=1 || height!=1)) {
                returnToStartPoint();
                break;
            }
            if(!countingTurn){
                widthPrev=width;
                width=1;
            }
            else{
                heightPrev=height;
                height=1;
            }
            countingTurn=!countingTurn;
            if(leftIsClear())
                turnLeft();
            else turnRight();
        }
    }
}
```

stickToBorder() function is only invoked in the findStartPoint() function, it simply keeps moving until its blocked, then turns right.

```
//A function that runs forward until it is blocked, then turns right
private void stickToBorder(){
    while(true) {
        if(frontIsClear())
            move();
        else {
            turnRight();
            break;
        }
    }
}
```

returnToStartPoint() function relocates Karel into the southwest facing east.

```
//A function that allows Karel to return to the south-west and facing east
private void returnToStartPoint() {
    if(!isHome()) {
        //Adjust Karel's orientation to face west
        while (!facingWest())
            turnLeft();

        //Moves Karel to the utmost west
        while (true) {
            if (frontIsClear())
                move();
            else {
                turnLeft();
                break;
            }
        }

        //Moves Karel to the utmost south
        while (true) {
            if (frontIsClear())
                move();
            else {
                turnLeft();
                break;
            }
        }
    }
}
```

The isHome() function returns whether if Karel is the southwest facing east or not.

```
//Checks if Karel is at the default positioning
private boolean isHome() {
    return facingEast() && rightIsBlocked() && backIsBlocked();
}
```

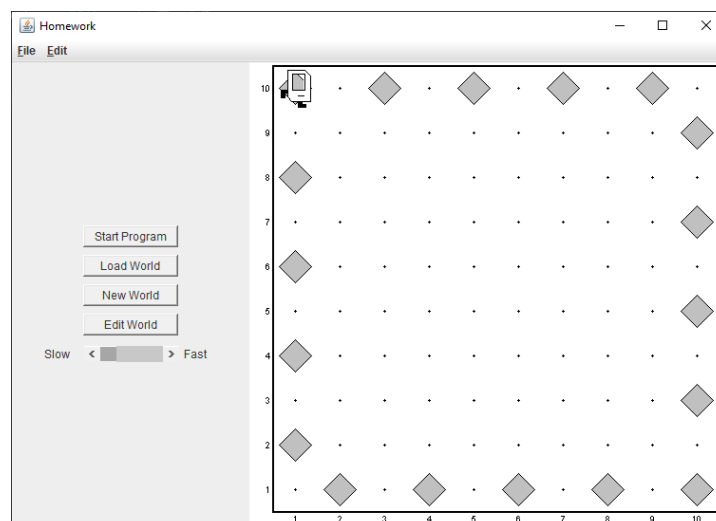
## Section 2: Level 1 Overview

The first level requires that Karel put beepers at the odd coordinates  $(x+y)\%2 \neq 0$ , but only putting the beepers at the border. And then picks them back.

At first, the function picks a beeper at the first corner (if it exists), then it enters a loop in which it will only exit it when Karel has finished the requirements, as well as when Karel is on the correct positioning.

The loop consists of two parts, going and returning.

- **Going:** Karel keeps moving forward while putting the beepers and picking beepers that are misplaced (by using the iterator variable, which iterates to false if the beeper is put, and true if nothing is done), until it hits an obstacle, then it turns either left or right (depending on the direction of Karel), moves up, then turns left or right again. If Karel traversed through all of the corners (indicated if right is blocked and facing west or vice versa "in odd heights"), then the "going" process is finished.
- **Returning:** Karel will not attempt to return the same way as it has gone, instead, it will only go through the borders and picks all the beepers, then turns around to the start point.



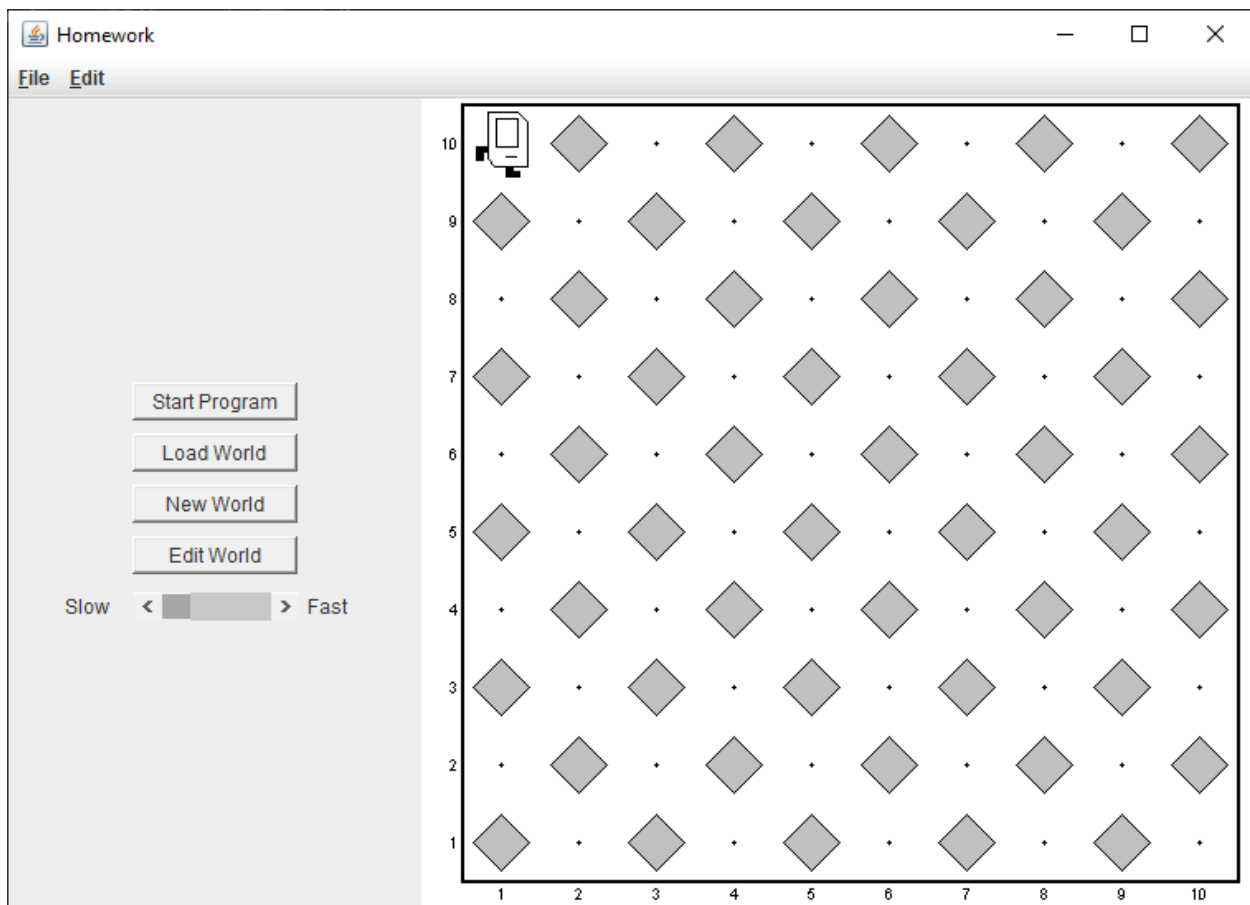


## Section 3: Level 2 Overview

The second level requires that Karel put beepers at the even coordinates " $(x+y)\%2 == 0$ ". And then picks them back.

At first, the function puts a beeper at the first corner (if it does not exist), then it enters a loop in which it will only exit it when Karel has finished the requirements, as well as when Karel is on the correct positioning.

The loop consists of two parts, going and returning, it is very similar to the first level, so it will not be discussed "the only difference is that at return, it will iterate through the whole world since Karel did not only put the beepers at the borders".

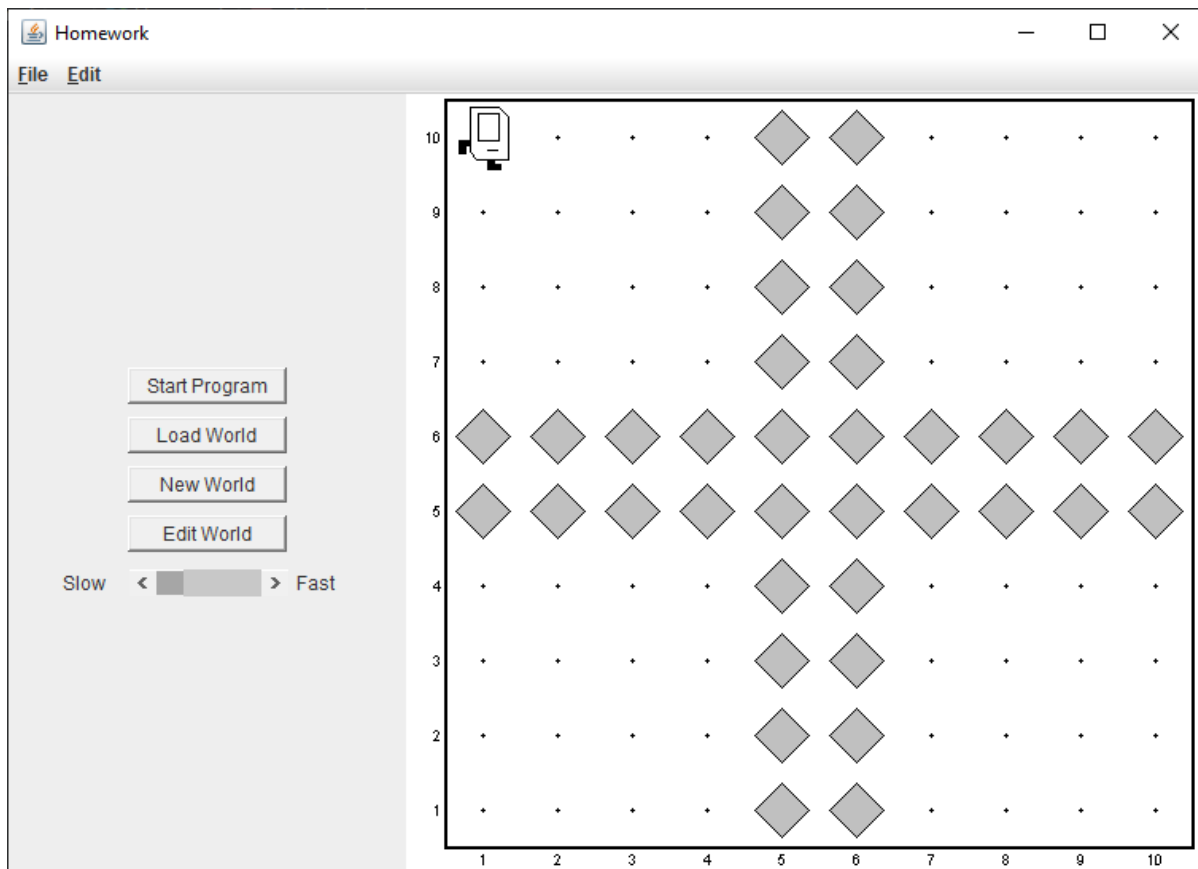


## Section 4: Level 3 Overview

The third level requires that Karel put beepers at the center coordinates “ $x/2 == x_{Karel} \parallel y/2 == y_{Karel}$ ;  $\rightarrow$  if the rows/columns are odd”. And then picks them back.

The function enters a loop in which it will only exit it when Karel has finished the requirements, as well as when Karel is on the correct positioning, Karel will put beepers at positions which are either centered horizontally or vertically. This is done by using `isCentered()` function, which uses Karel’s coordinates as well as the world’s dimensions. The world’s dimensions (width, height) are already known since it was calculated at the first requirement.

The loop consists of two parts, going and returning, it is very similar to the first level, so it will not be discussed “the only difference is that at return, it will try to iterate through only the coordinates where the beepers where put”.



## Section 5: Limitations & Flaws






Although the program runs as required, there are certain situations where Karel may react undesirably.

- 1. backIsBlocked() Function:** this function is not provided by the superclass, hence, the only way to check if back is blocked or clear is by rotating either left or right, then check if left/ right is clear. While this approach works as intended, it will display an annoying visual of Karel rotating in, then rotating back. The reason this function is needed is to handle the corner cases of either singular width or height.
- 2. First Level Return Movement:** Even though the first return procedure is optimized to iterate through only the borders, at odd columns, Karel will return to the origin point, and instead of stopping, it will go forward the whole way, then returns and breaks.
- 3. Third Level Return Movement:** As the first level, the third level return procedure is also optimized. However, if there are two centered rows/ columns. Karel will loop through it twice instead of once.
- 4. Mazes in The World:** Although the code will be able to exit simple mazes successfully when EXPLORE variable is set to true, the three functions are not made to handle drawing in such worlds; expect inconsistent results.
- 5. Code Complexity:** The code may be deducted to a few dozens of codes; however, it requires a certain level of problem-solving skills that is not present currently.

## Section 6: Further Notes

There are a few notes that should be covered before diving into the code.

- Although the description specified that Karel is defaulted at the southwest facing east, the code have the capability to handle other positions and exit simple mazes (such as “CollectNewspaperKarel” world).
- The code can run any function individually. **However**, you must provide the width and height of the map in the fields respectively, otherwise, the third level will not function properly.
- Since the levels are set to run individually, each function is responsible to check all the corners and pick the beeper that is in the wrong position while coming; expect longer times collecting the beepers for the “coming” process due to this functionality.
- The code is set to play an audio upon an event, the audio files are already sent in addition to the code. They must be at the same directory “src/” (please refer to the following image). The code can still function even without them. The audio is set to run upon these events:
  1. Clicking the “Start Program” button
  2. Picking up a beeper
  3. Putting a beeper

 BlankKarel.java	08-Sep-20 10:40 AM	IntelliJ IDEA Com...	1 KB
 Homework.java	22-Oct-20 4:34 PM	IntelliJ IDEA Com...	21 KB
 Pick.wav	22-Oct-20 3:08 PM	Wave Sound	40 KB
 Put.wav	22-Oct-20 3:08 PM	Wave Sound	40 KB
 Start.wav	22-Oct-20 2:44 PM	Wave Sound	256 KB