ATYPON

Written by: Moayad Al-Saleh

Karel

What is Karel?

Karel is a very simple robot living in a very simple world. By giving Karel a set of commands, you can direct it to perform certain tasks within its world. The process of specifying those commands is called programming. Initially, Karel understands only a very small number of predefined commands, but an important part of the programming process is teaching Karel new commands that extend its capabilities.

What can Karel do?

Move (): Karel moves forward just one step.

Turn Left (): Karel rotates 90 degrees to the left.

Turn Right (): Karel rotates 90 degrees to the right.

Put Beeber (): Karel can put one beeper from its bag.

What is the problem?

It is to make Karel cut the map into equal parts and put points in the center of each part by using beeber to determine the cutting mark and using the movements that Karel can make.

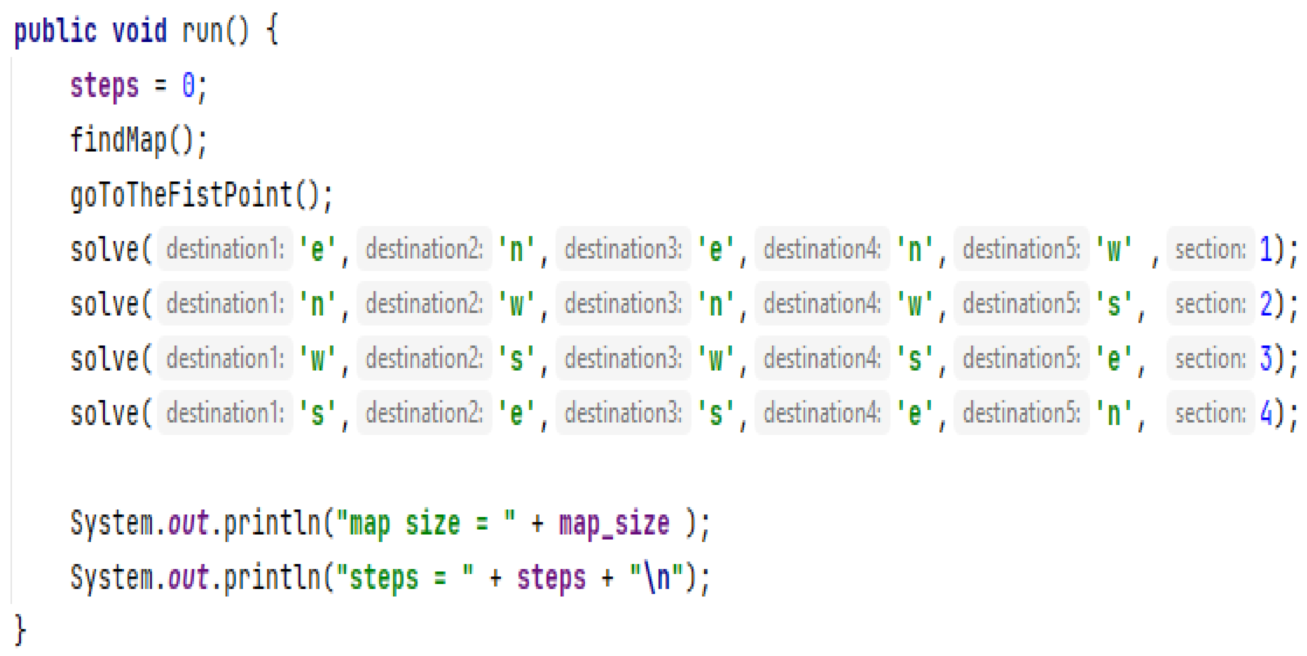
The plan to solve the problem :

I read Karel Assignment very well.

Find the shortest possible way to divide the map.

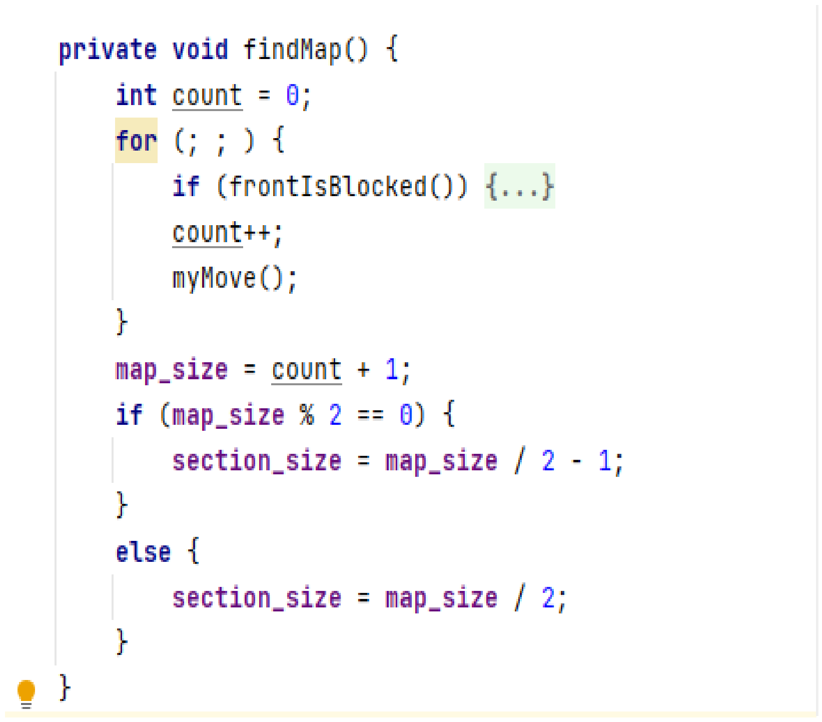
Apply the solution programmatically so that it is practical and covers all cases.

The solution is divided into several stages:

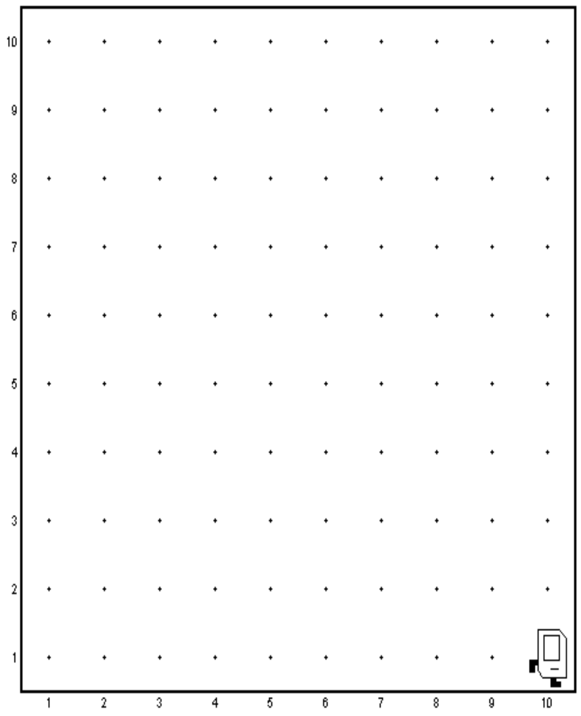


```
public void run() {
    steps = 0;
    findMap();
    goToTheFistPoint();
    solve( destination1: 'e', destination2: 'n', destination3: 'e', destination4: 'n', destination5: 'w' , section: 1);
    solve( destination1: 'n', destination2: 'w', destination3: 'n', destination4: 'w', destination5: 's',  section: 2);
    solve( destination1: 'w', destination2: 's', destination3: 'w', destination4: 's', destination5: 'e',  section: 3);
    solve( destination1: 's', destination2: 'e', destination3: 's', destination4: 'e', destination5: 'n',  section: 4);

    System.out.println("map size = " + map_size );
    System.out.println("steps = " + steps + "\n");
}
```

. findMap :

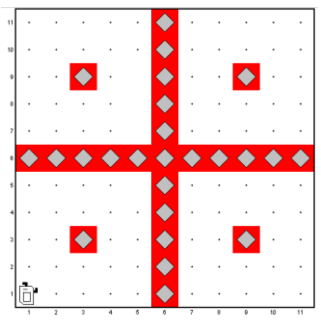


```
private void findMap() {
    int count = 0;
    for (; ; ) {
        if (frontIsBlocked()) {...}
        count++;
        myMove();
    }
    map_size = count + 1;
    if (map_size % 2 == 0) {
        section_size = map_size / 2 - 1;
    }
    else {
        section_size = map_size / 2;
    }
}
```

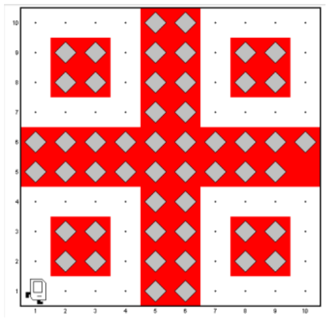


Karel goes to the end of the map and counts how many steps he has walked so that he knows which map he is dealing with.

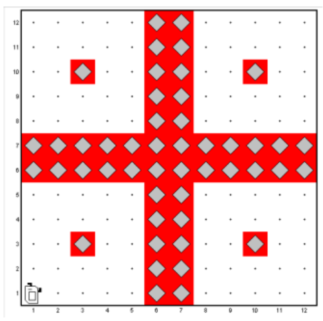
Maps are of four types:



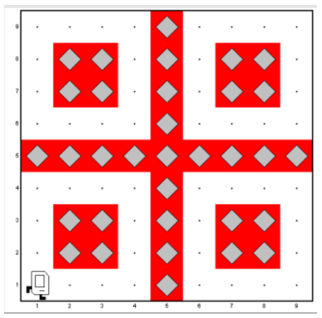
Single line and points



2 lines and squares

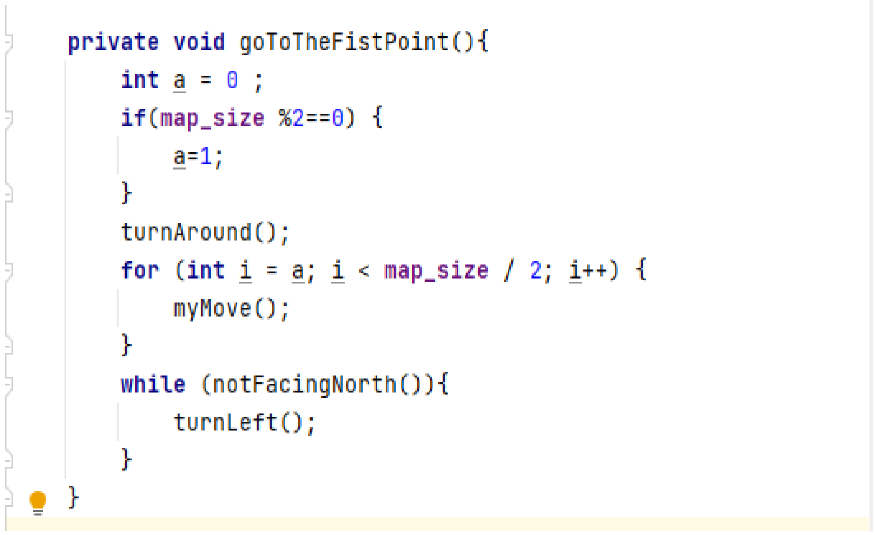


2 lines and points

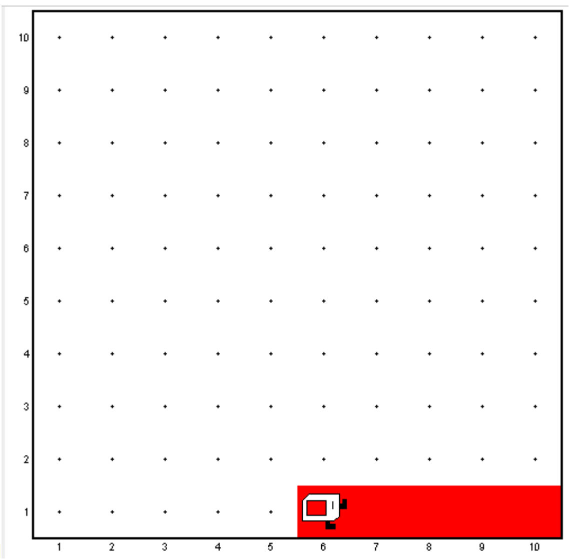


single line and squares

. goToTheFistPoint :



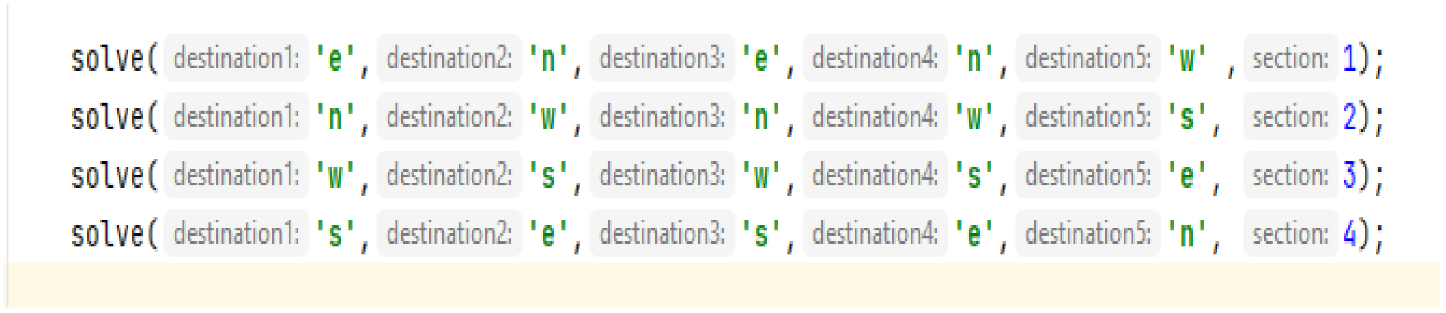
```
private void goToTheFistPoint(){
    int a = 0 ;
    if(map_size %2==0) {
        a=1;
    }
    turnAround();
    for (int i = a; i < map_size / 2; i++) {
        myMove();
    }
    while (notFacingNorth()){
        turnLeft();
    }
}
```



After knowing which map he deals with, Karel goes to the beginning of the main line from which he will start dividing the map, and it will be in the middle and its location will change according to whether the map has double or single line.

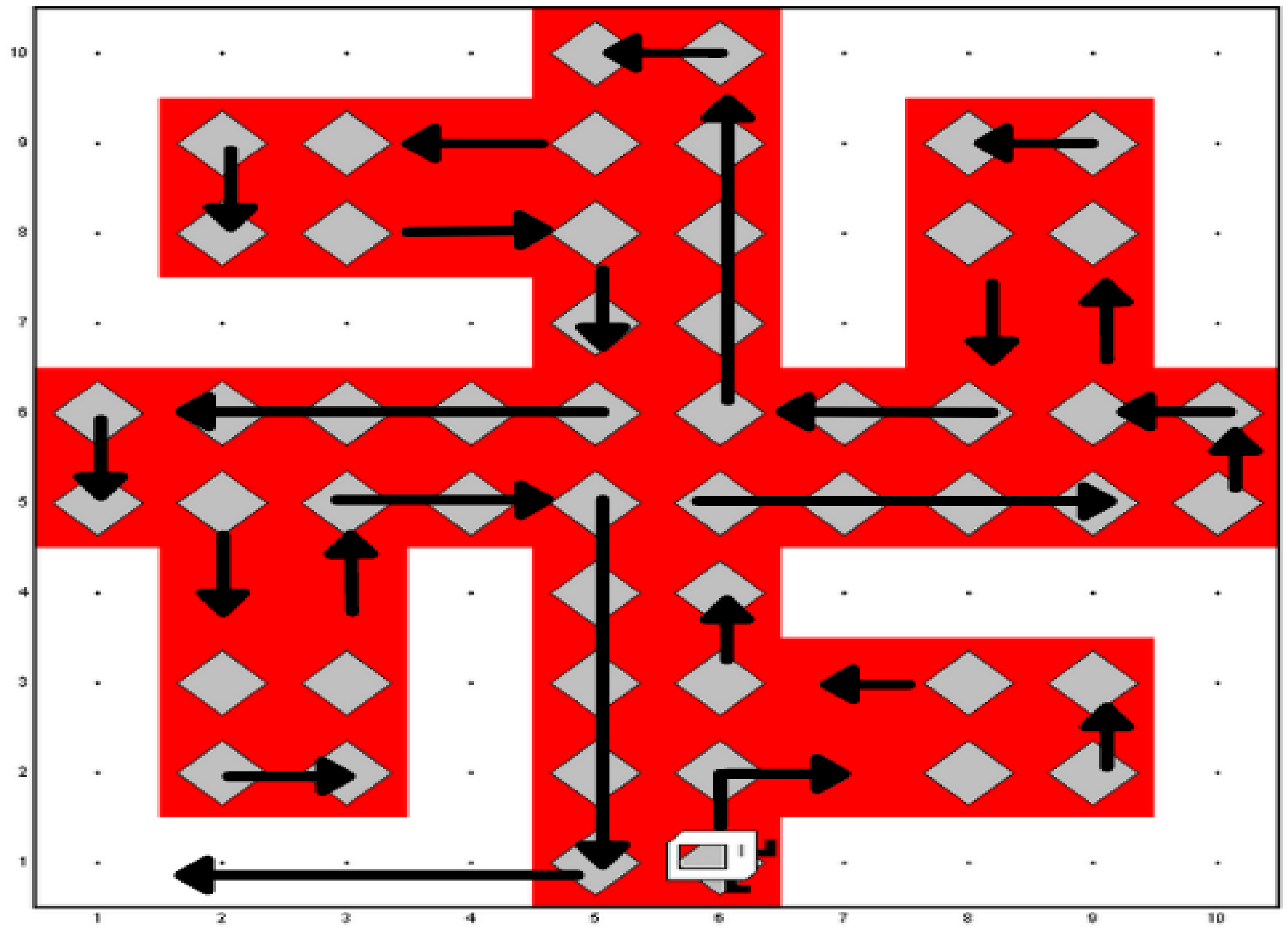
. solve:

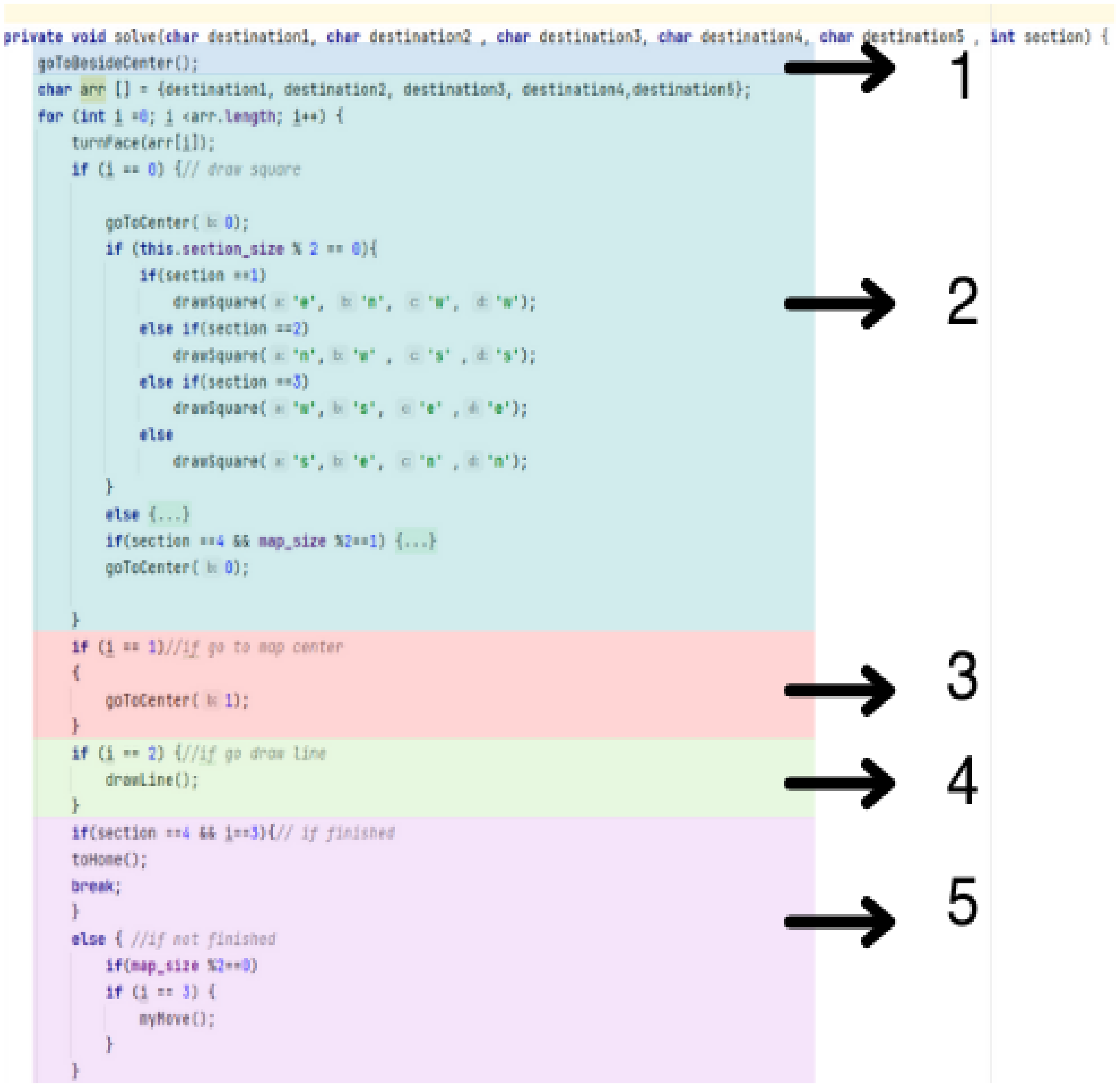
Here we begin to call the main function (solve):



```
solve( destination1: 'e', destination2: 'n', destination3: 'e', destination4: 'n', destination5: 'w' , section: 1);
solve( destination1: 'n', destination2: 'w', destination3: 'n', destination4: 'w', destination5: 's', section: 2);
solve( destination1: 'w', destination2: 's', destination3: 'w', destination4: 's', destination5: 'e', section: 3);
solve( destination1: 's', destination2: 'e', destination3: 's', destination4: 'e', destination5: 'n', section: 4);
```

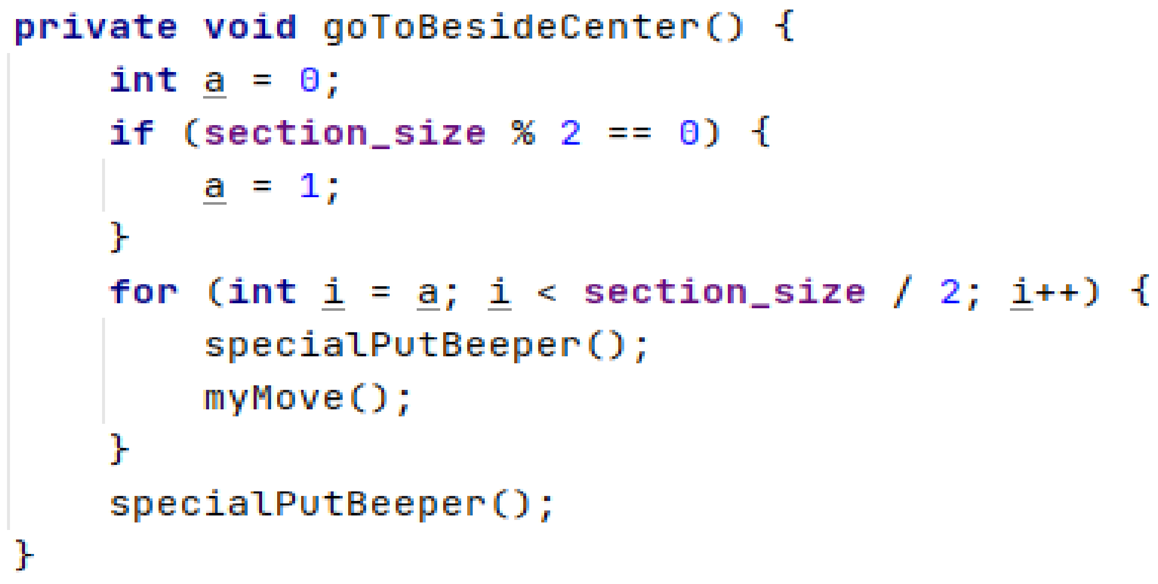
Here are the destinations that Karel will follow , which will be working on all different types of maps dynamically :



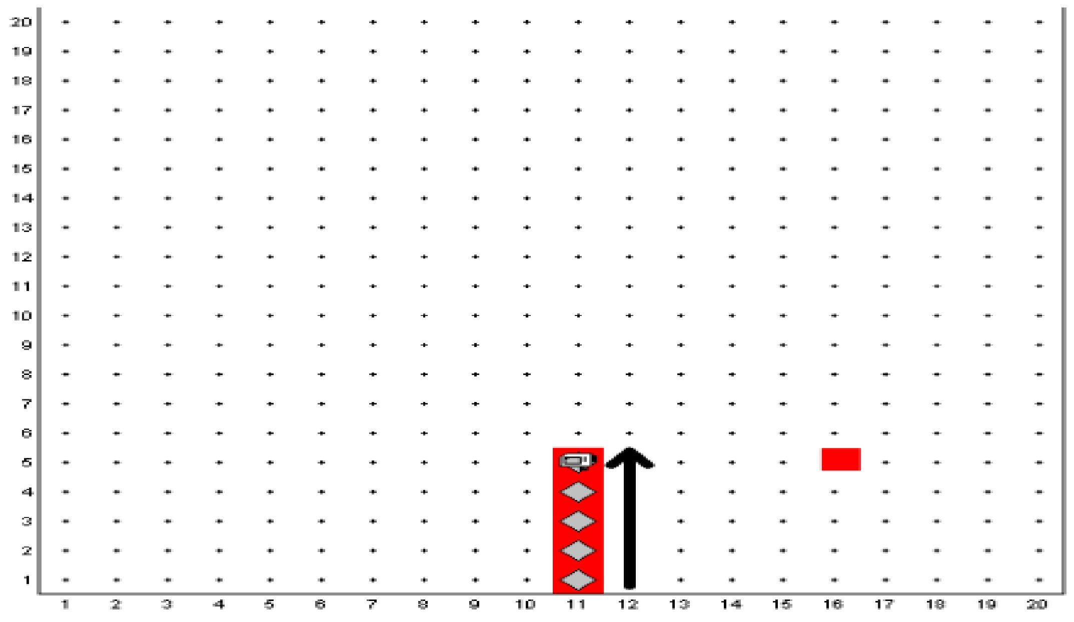


```
private void solve(char destination1, char destination2 , char destination3, char destination4, char destination5 , int section) {
    goToBesideCenter();
    char arr [] = {destination1, destination2, destination3, destination4,destination5};
    for (int i =0; i <arr.length; i++) {
        turnFace(arr[i]);
        if (i == 0) {// draw square

            goToCenter( b: 0);
            if (this.section_size % 2 == 0){
                if(section ==1)
                    drawSquare( a: 'e', b: 'n', c: 'w', d: 'w');
                else if(section ==2)
                    drawSquare( a: 'n', b: 'w' , c: 's' , d: 's');
                else if(section ==3)
                    drawSquare( a: 'w', b: 's', c: 'e' , d: 'e');
                else
                    drawSquare( a: 's', b: 'e', c: 'n' , d: 'n');
            }
            else {...}
            if(section ==4 && map_size %2==1) {...}
            goToCenter( b: 0);

        }
        if (i == 1)//if go to map center
        {
            goToCenter( b: 1);
        }
        if (i == 2) {//if go draw line
            drawLine();
        }
        if(section ==4 && i==3){// if finished
        toHome();
        break;
        }
        else { //if not finished
            if(map_size %2==0)
            if (i == 3) {
                myMove();
            }
        }
```

Solve→ 1 : goToBesideCenter

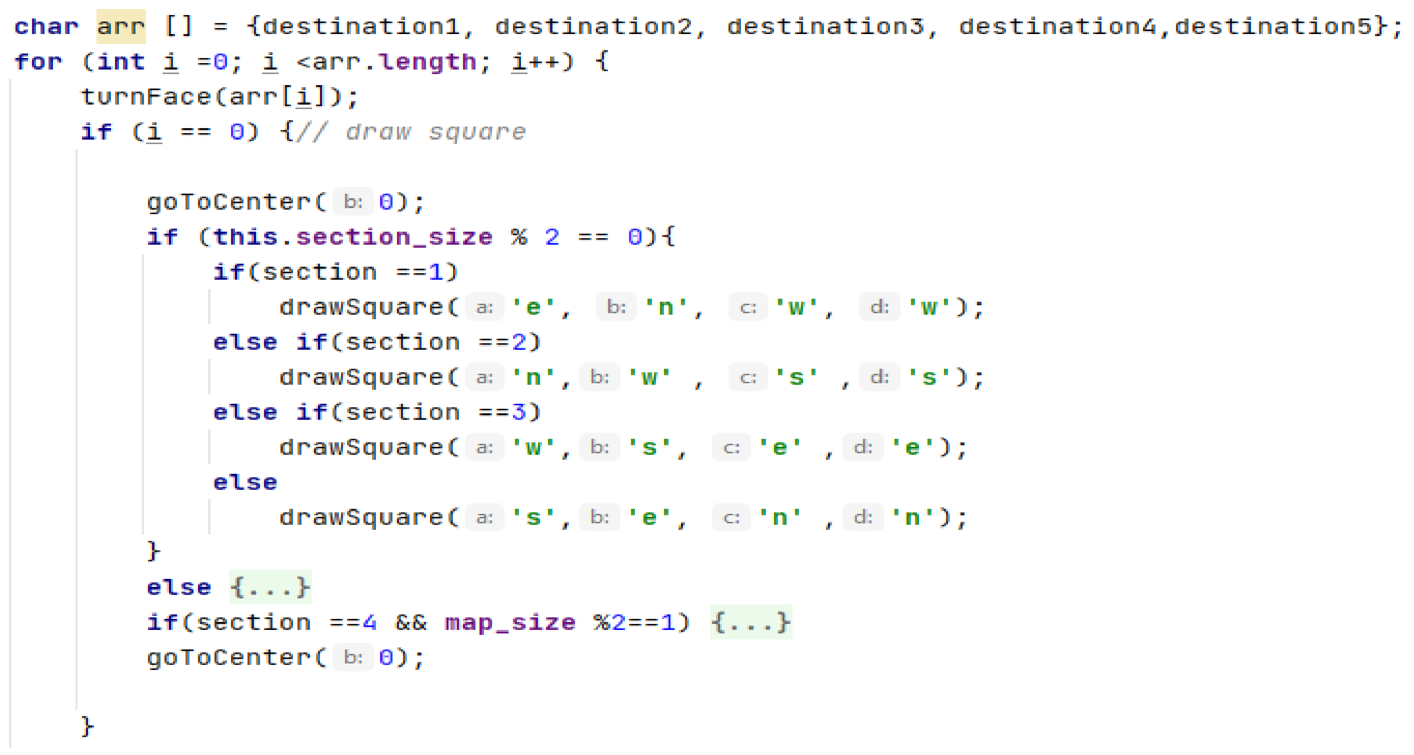


```java
private void goToBesideCenter() {
    int a = 0;
    if (section_size % 2 == 0) {
        a = 1;
    }
    for (int i = a; i < section_size / 2; i++) {
        specialPutBeeper();
        myMove();
    }
    specialPutBeeper();
}
```



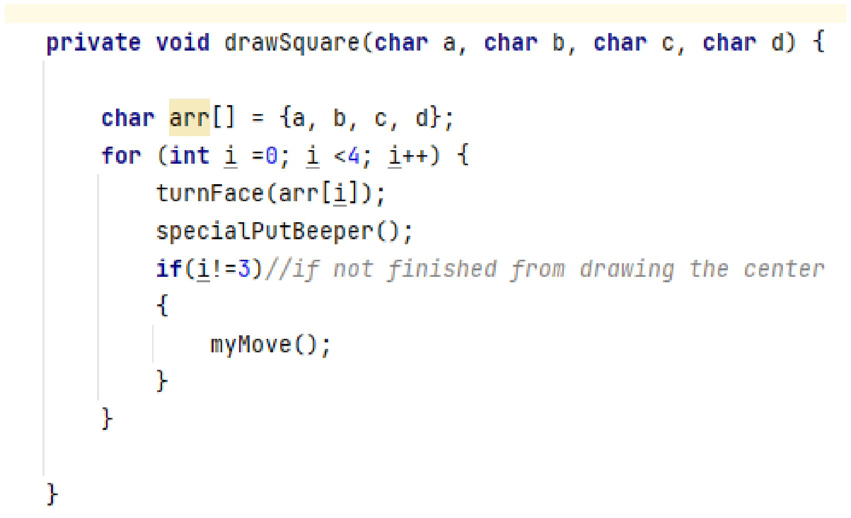
Here Karel begins to move to draw the main line and stops besides the center of the part, whether the center is a square or a point

Solve → 2: drawsquare



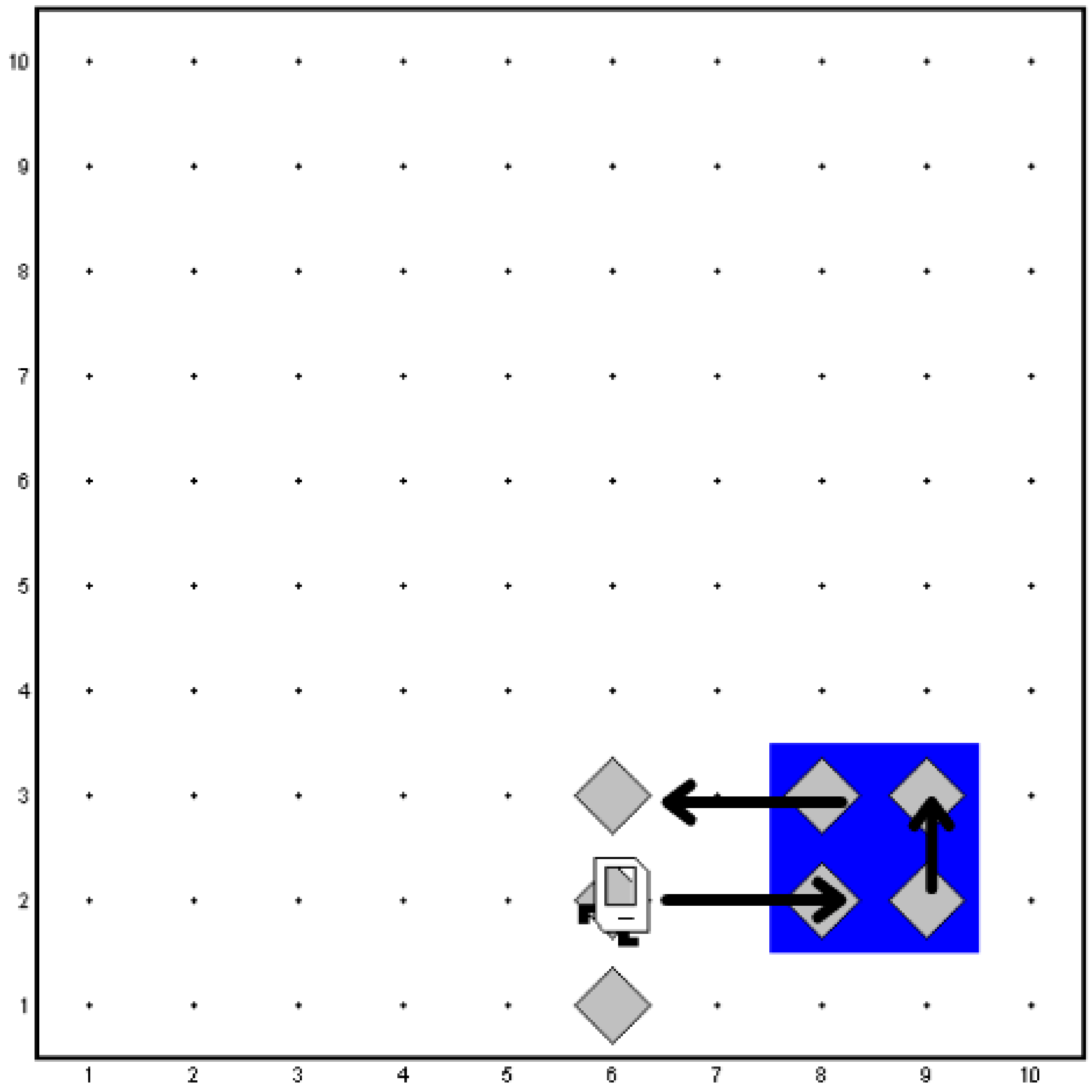
```
char arr [] = {destination1, destination2, destination3, destination4,destination5};
for (int i =0; i <arr.length; i++) {
    turnFace(arr[i]);
    if (i == 0) {// draw square

        goToCenter( b: 0);
        if (this.section_size % 2 == 0){
            if(section ==1)
                drawSquare( a: 'e',  b: 'n',  c: 'w',  d: 'w');
            else if(section ==2)
                drawSquare( a: 'n', b: 'w' ,  c: 's' , d: 's');
            else if(section ==3)
                drawSquare( a: 'w', b: 's',  c: 'e' , d: 'e');
            else
                drawSquare( a: 's', b: 'e',  c: 'n' , d: 'n');
        }
        else {...}
        if(section ==4 && map_size %2==1) {...}
        goToCenter( b: 0);

    }
```

At this stage, go to the center and draw it and return to the main line by entering the destinations of the section .

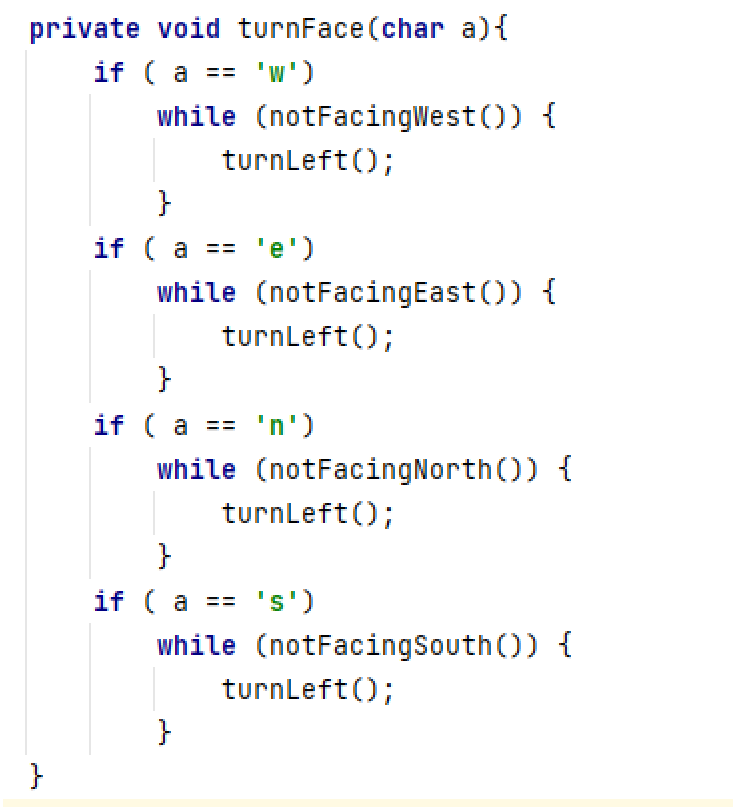


```
private void drawSquare(char a, char b, char c, char d) {

    char arr[] = {a, b, c, d};
    for (int i =0; i <4; i++) {
        turnFace(arr[i]);
        specialPutBeeper();
        if(i!=3)//if not finished from drawing the center
        {
            myMove();
        }
    }

}
```

Solve → 2: drawsquare



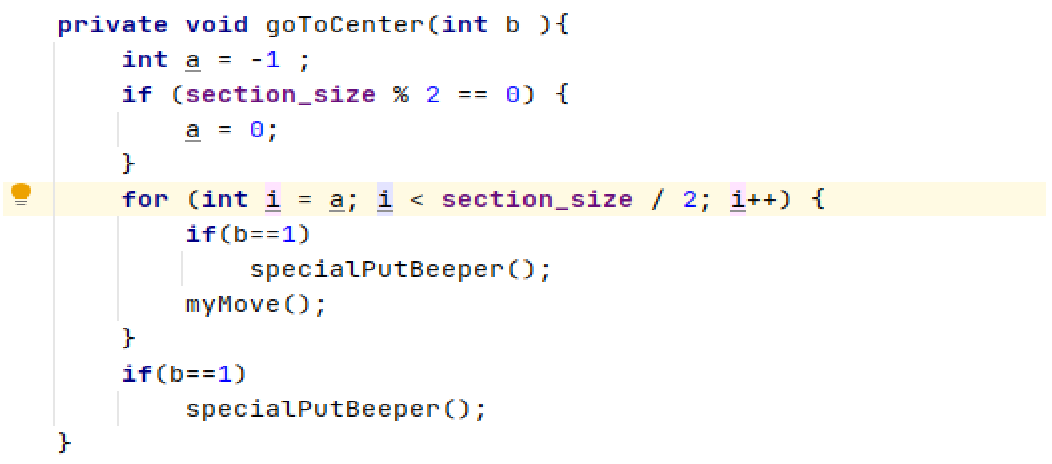
drawsquare → turnFace



```
private void turnFace(char a){
    if ( a == 'w')
        while (notFacingWest()) {
            turnLeft();
        }
    if ( a == 'e')
        while (notFacingEast()) {
            turnLeft();
        }
    if ( a == 'n')
        while (notFacingNorth()) {
            turnLeft();
        }
    if ( a == 's')
        while (notFacingSouth()) {
            turnLeft();
        }
}
```

This is one of the most important functions where you receive directions from other functions and actually turn Karel face in the right direction.

Solve → 3: goToCenter



```java
private void goToCenter(int b ){
    int a = -1 ;
    if (section_size % 2 == 0) {
        a = 0;
    }
    for (int i = a; i < section_size / 2; i++) {
        if(b==1)
            specialPutBeeper();
        myMove();
    }
    if(b==1)
        specialPutBeeper();
}
```

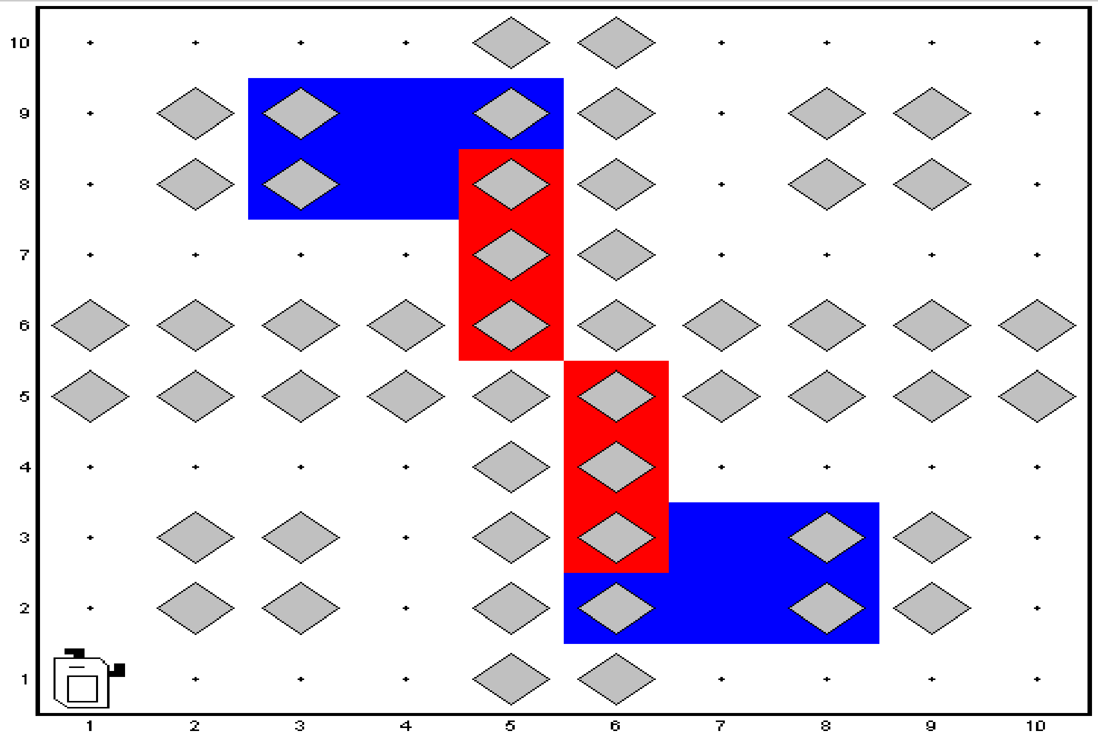
This function does two things:

1. *.Karel moves to the center of the section and also returning from it .*

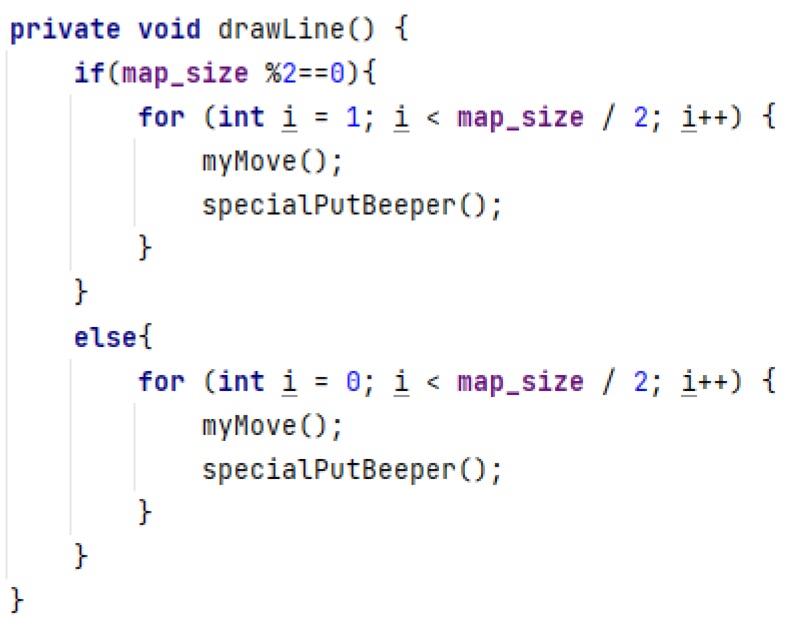
   This is demonstrated in blue in the figure below.

2. *Karel moves to the center of the map.*

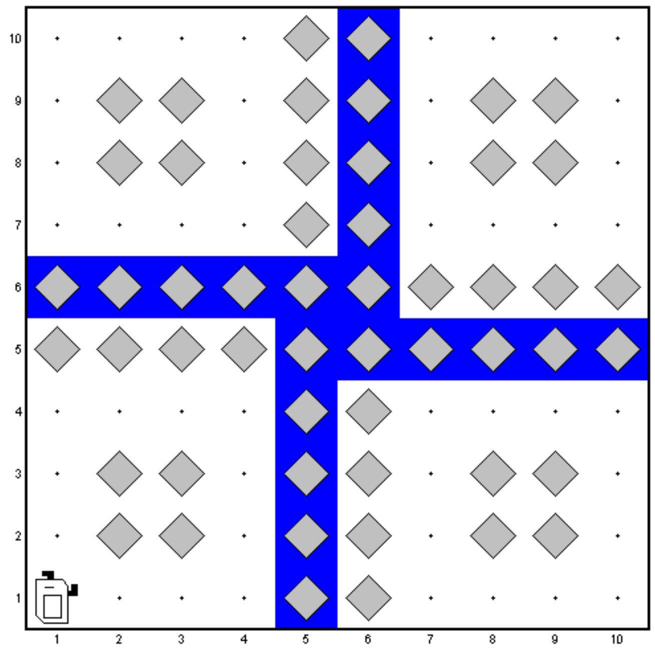
   This is demonstrated in red in the figure below.



Solve → 4 : drawLine



```
private void drawLine() {
    if(map_size %2==0){
        for (int i = 1; i < map_size / 2; i++) {
            myMove();
            specialPutBeeper();
        }
    }
    else{
        for (int i = 0; i < map_size / 2; i++) {
            myMove();
            specialPutBeeper();
        }
    }
}
```

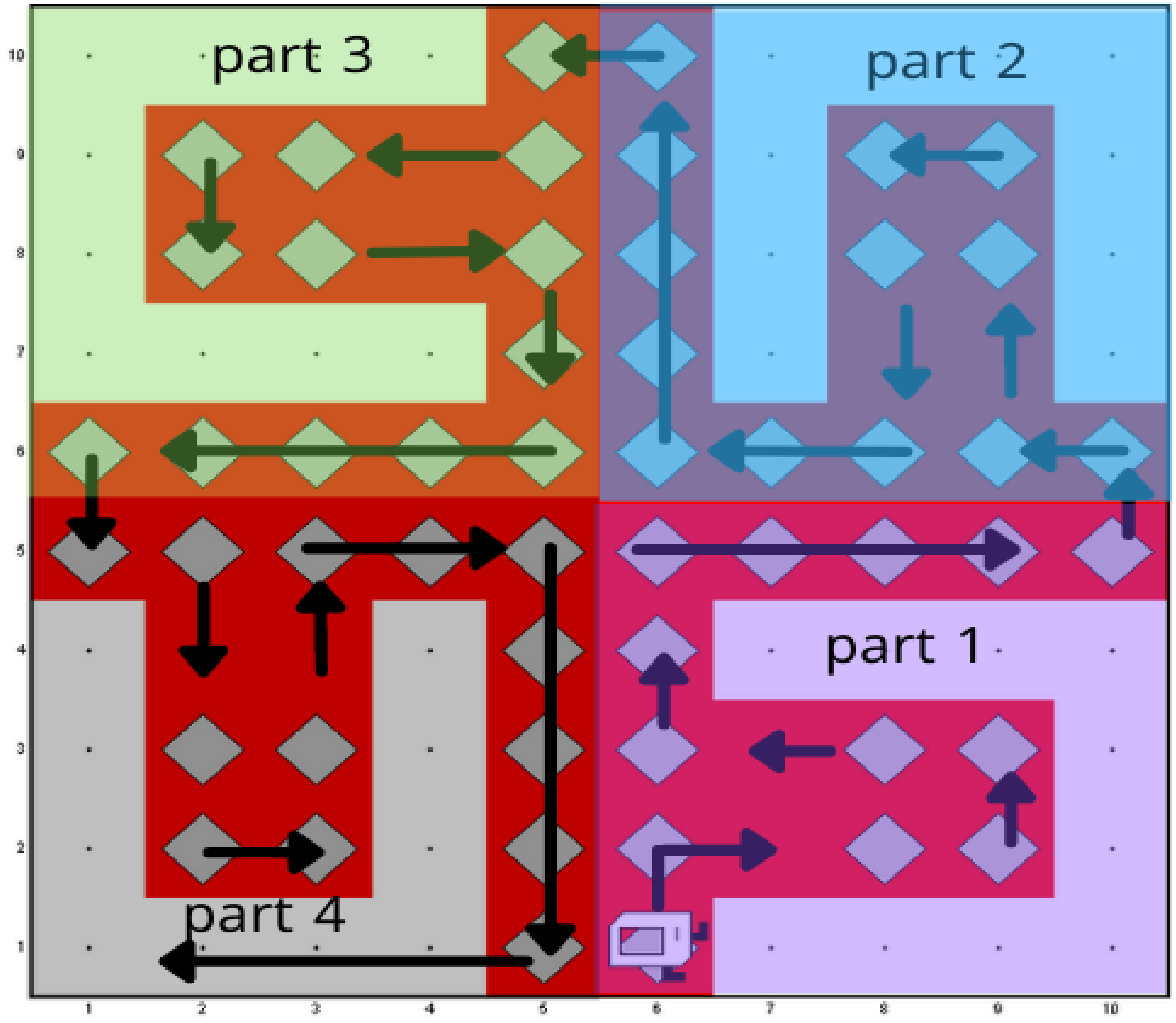


Here Karel is sent to draw the line extending from the main center to the border of the map.

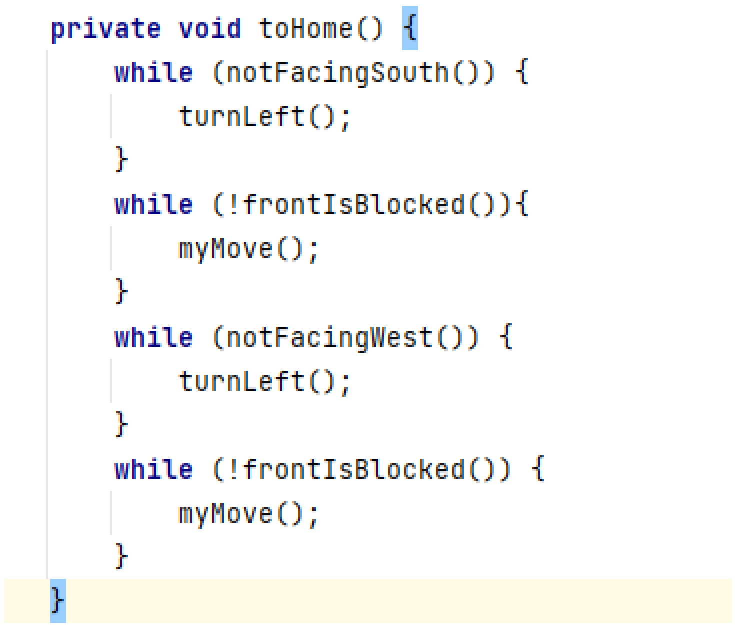
Solve

Function solve is called four times to complete the map

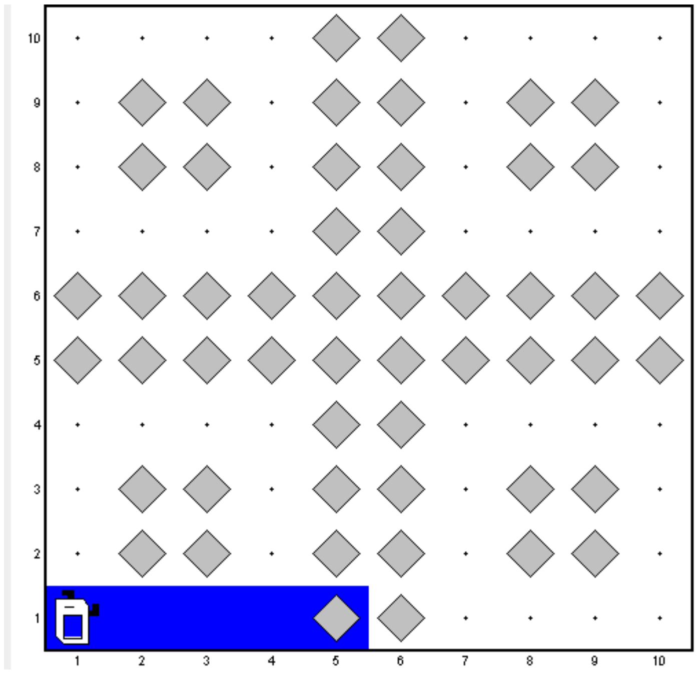
```
solve( destination1: 'e', destination2: 'n', destination3: 'e', destination4: 'n', destination5: 'w' , section: 1);
solve( destination1: 'n', destination2: 'w', destination3: 'n', destination4: 'w', destination5: 's', section: 2);
solve( destination1: 'w', destination2: 's', destination3: 'w', destination4: 's', destination5: 'e', section: 3);
solve( destination1: 's', destination2: 'e', destination3: 's', destination4: 'e', destination5: 'n', section: 4);
```



Solve → 5 : toHome



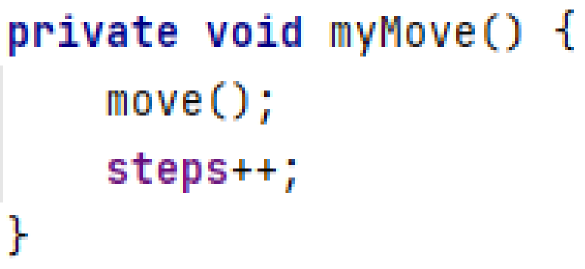
```
private void toHome() {
    while (notFacingSouth()) {
        turnLeft();
    }
    while (!frontIsBlocked()){
        myMove();
    }
    while (notFacingWest()) {
        turnLeft();
    }
    while (!frontIsBlocked()) {
        myMove();
    }
}
```



It directs karel to go home to the point (1,1).

Some other functions:

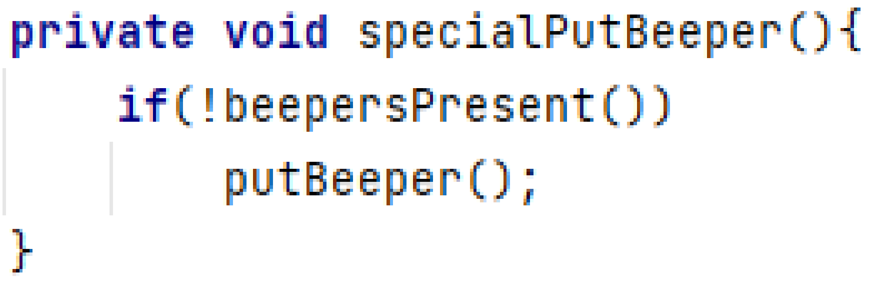
myMove



```
private void myMove() {
    move();
    steps++;
}
```

counting the steps of Karel.

specialPutBeeper

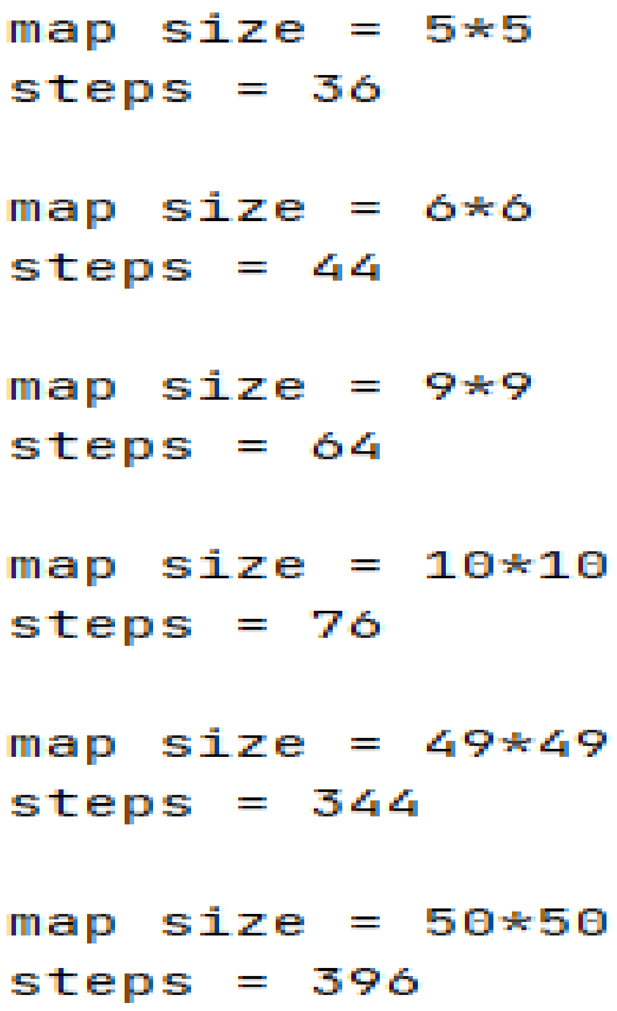


```
private void specialPutBeeper(){
    if(!beepersPresent())
        putBeeper();
}
```

Putting beepers in the designated place

In the end:

I thought of this solution to make Karel divide the map into equal parts and with the least possible code that I could come up with covering all types of maps and these are some of the results my solution reached and I hope you like it.



```
map size = 5*5
steps = 36

map size = 6*6
steps = 44

map size = 9*9
steps = 64

map size = 10*10
steps = 76

map size = 49*49
steps = 344

map size = 50*50
steps = 396
```