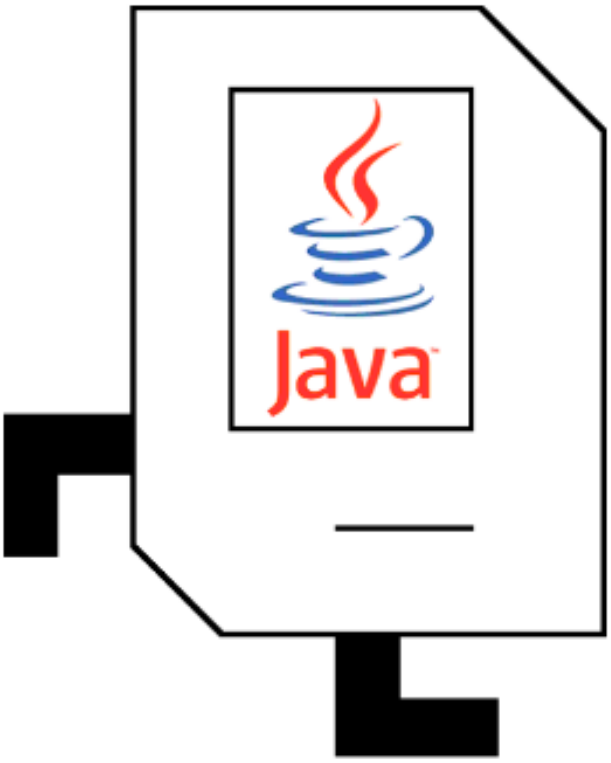


KAREL ASSIGNMENT

Name: Akram Jaghoub

1.0 Introduction

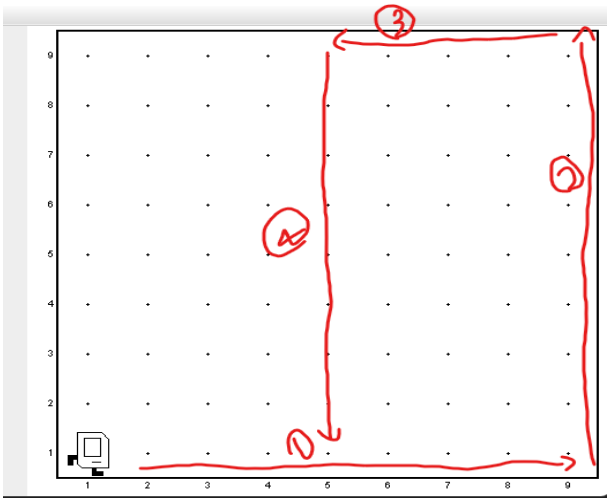
This documentation addresses the assignment of solving Karel the robot problem. The goal is to divide a given map into chambers using the fewest moves and beepers possible. Karel always starts at the bottom left corner of the map and must ensure that the inner chambers do not touch any walls. Double lines of beepers are necessary when the width or height of the map is even.
To approach the problem, I analyzed the requirements and experimented it with different approaches. I went through a range of solutions, prioritizing simplicity, efficiency, and minimizing the number of steps required.
The following sections will explain each method, and the cases which encountered me.

2.0 Decision on map size

Before going into the approaches, it's important to note that I started the count at 0 for both dimensions in my implementation of Karel. Therefore, for a map size of 9x9, the count would range from 0 to 8, not 1 to 9. However, this does not imply that I treated it as an even map. The logic remains consistent with treating it as an odd map; it's simply the counting convention that changed.

Approach 1:

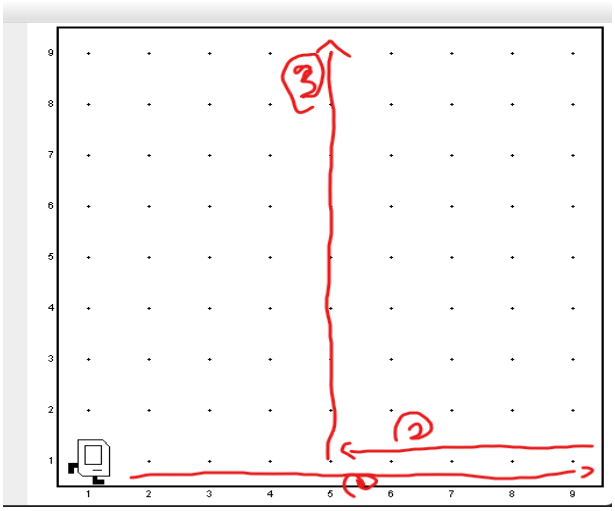
The initial approach I considered is to move Karel horizontally and count the width size until it reaches a wall (used the frontIsClear()). Then, Karel would move vertically and count the height until it encounters another wall. Afterward, Karel would position itself halfway across the width (e.g., 4 units to the left for a 9x9 map) to begin drawing vertically from top to bottom.



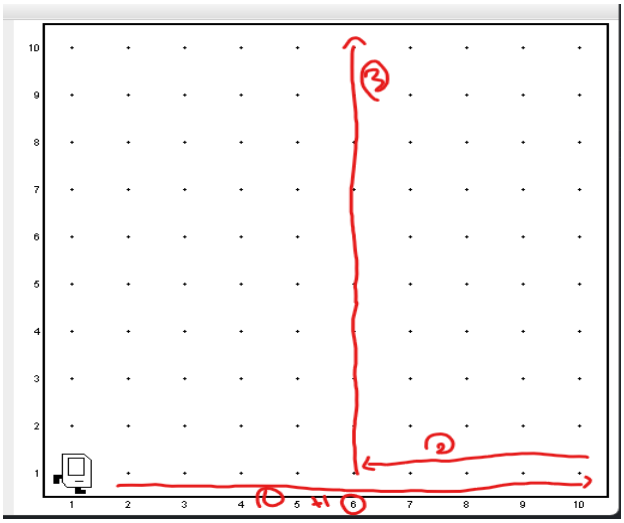
Approach 2:

After further consideration, I realized that the first approach wasn't really this efficient and thus, I discovered an alternative solution that minimizes the number of steps. In this approach, Karel would still move horizontally and count the width until it reaches a wall width (e.g., 4 units to the left for a 9x9 map). Afterwards, Karel turns around and move half the distance in the opposite direction and it takes two decisions:

- If the width is odd, Karel moves back by width / 2 units (e.g., 4 units to the left for a 9x9 map).



- If the width is even, Karel moves back by width / 2 + 1 units.



This change allows Karel to start drawing from the bottom and move vertically, simultaneously counting the height of the map.

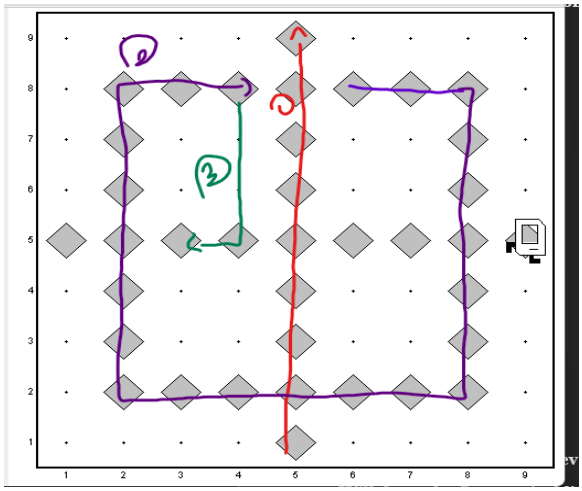
3.0 drawing vertical lines

After deciding where to begin drawing it's time to draw the vertical lines which will take upon some decisions:

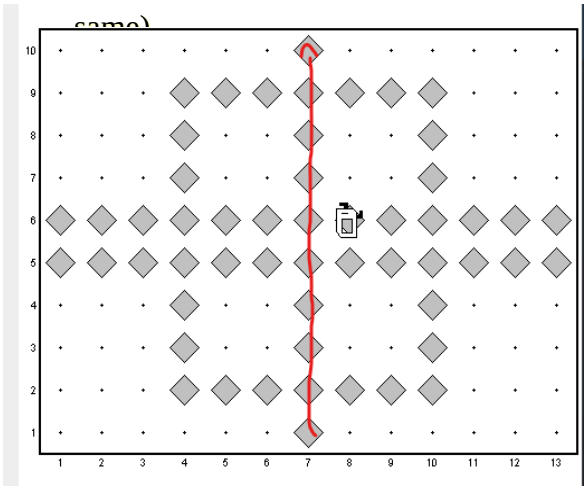
- **width is odd:**

Will draw the only one vertical line of the map and the proceed to draw the inner chambers (explained in 4.0) and finally the horizontal lines (explained in 5.0) will draw one line if height is odd and two if height is even.

Both are odd (drawings are only to show the flow of drawing)

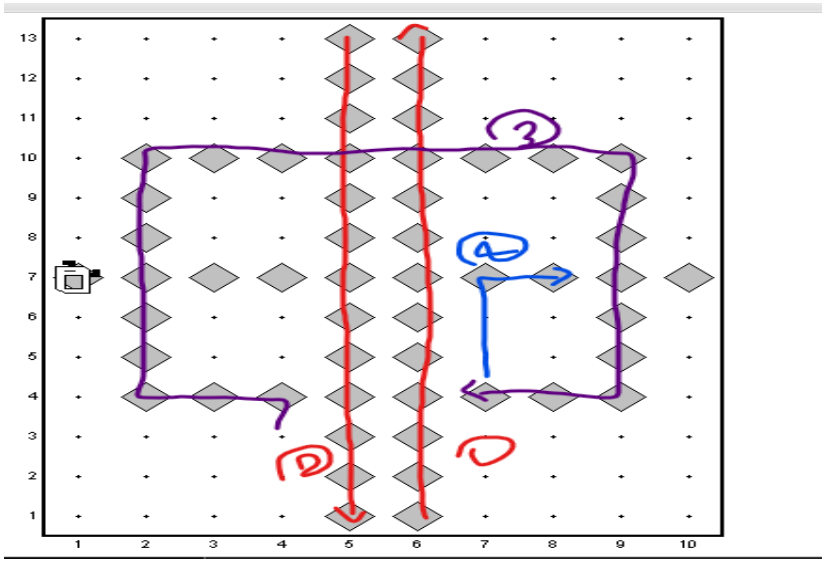


height is even (only difference two horizontal lines rest is the same)



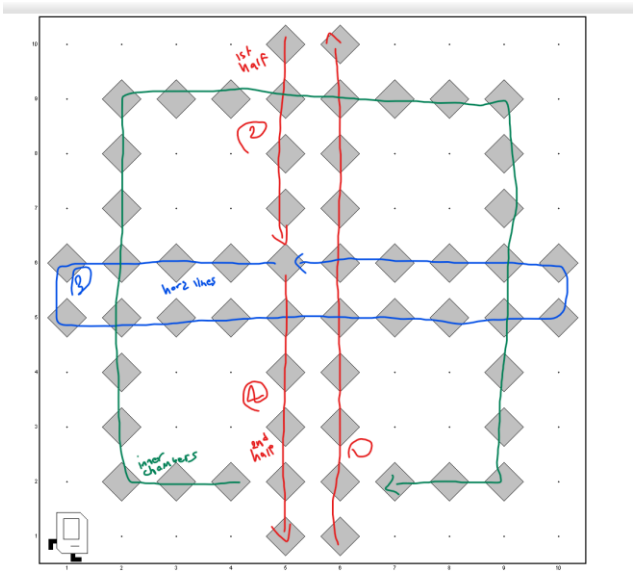
- **Width is even:**

Will draws the first vertical line and in case height is odd it draws the second vertical line (all of it) and then draws the inner chambers and finally the only horizontal line it as similar as it is when width is odd.



- **Both are even:**

Here the implementation differs from the rest to get the most minimal steps possible as we draw the first vertical line and then draw half of the second vertical line afterwards, draw both of the horizontal lines and then draw the other half of the second line , finally draw the inner chambers.



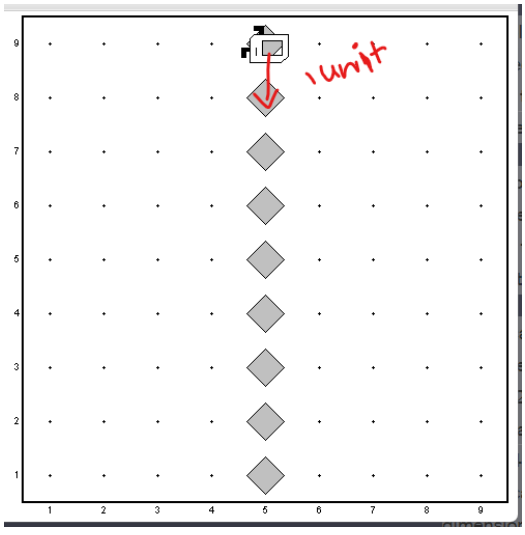
4.0 drawing the Inner chambers

After implementing Approach 2, Karel begins by drawing the four inner chambers, ensuring they are equal squares. This step is taken prior to drawing the horizontal lines in order to minimize the number of steps required. The positioning of these squares depends on the size of the map.

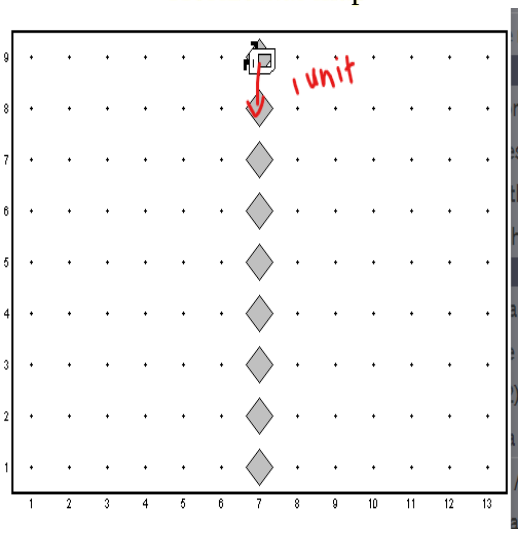
4.1 Calculating the starting point

- For horizontal and square maps (where width == height or width > height), the algorithm determines that Karel should start drawing the chambers either 1 unit down or 1 unit up, based on the size of the map. Specifically, Karel moves 1 unit down if the width is odd and 1 unit up if the width is even.

Square map

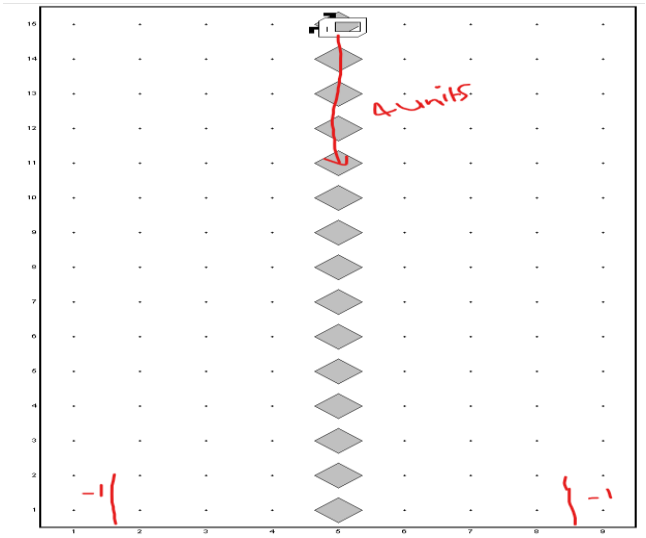


Horizontal map



- For vertical maps where (width < height), I used the following formula:

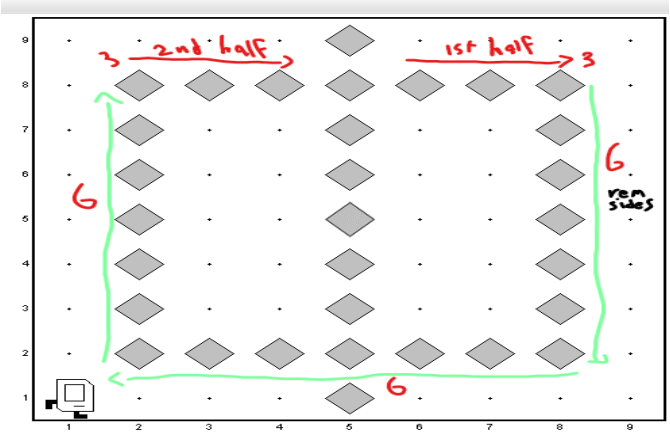
(height / 2) - calculateSquareLength(minimum) / 2. Let me give an example, consider a map size of 15 x 9. As the width is smaller than the height, we take half of the height (14 / 2 = 7, note that it is 14 as explained earlier). Then, we subtract the square length, which is calculated as the minimum dimension minus 2 through calculateSquareLength(minimum). In this case, the minimum dimension is the width (8 - 2) because the square must be at least 1 dimension less from the walls. Considering the presence of two vertical walls, we subtract 2. Finally, we take half of the minimum value. The resulting value for the example I have provided would be 4, so Karel would move 4 units down.



4.2 drawing the square

After determining the starting point for the square, the drawing process begins. I achieved this by using the calculateSquareLength(minimum) method mentioned in the previous section. Next, I used the drawHalfSize(length) method, which involves a loop that iterates half the length of the square. For instance, in a 9x9 map, the length would be 6, so we would draw half of it, which is 3. Thus, the length of the square "horizontally" would be 3 on both the right and left sides of the middle (total of 6).
Following this, I created the drawRemainingSides(length, start) method, which involves making decisions based on the following scenarios:

- if both width and height are even or odd draw the remaining sides by using the length and multiply it by 3 through a loop because the remaining sides are actually after drawing the first half let's say for a 9x9 map the length would be 6 so (6 * 3 = 18 ➔ remaining sides are 6 + 6 + 6 "3 left sides")



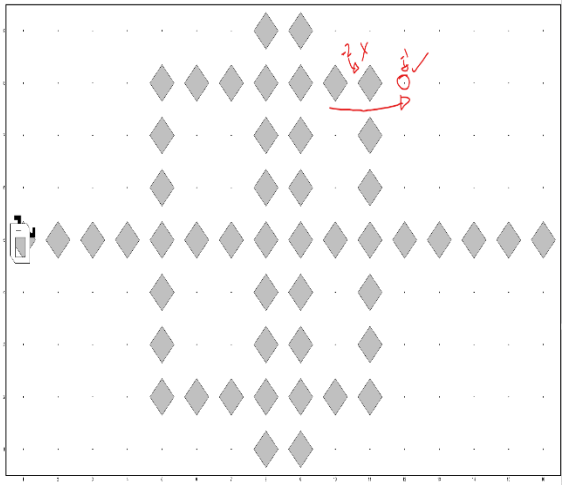
- else in case one of the dimensions is odd and the other is even, we take the start point as an argument. First, the drawVerticalEdge(start) method is called to draw the right or left side of the square. Then, the drawHorizontalEdge() method is called to draw the bottom or top side of the square. Finally, the drawVerticalEdge(start) Is called again to draw the other remaining half side of the square let's go deep into these methods:

  drawVerticalEdge(start): we used a counter that subtracts the height by the starting point of the square to calculate where to stop the counter and then loop through this to draw the left or right edge of the square.

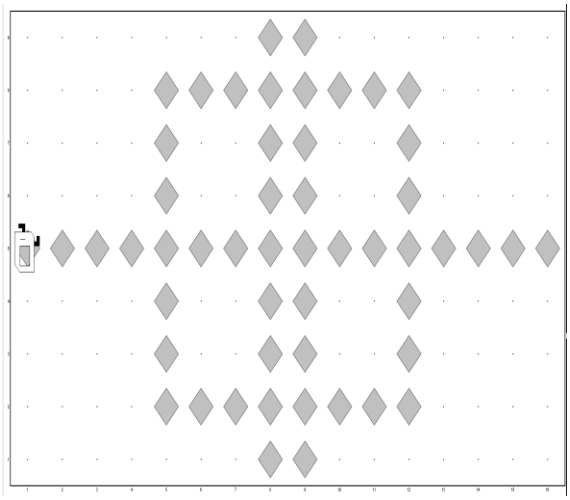
  drawHorizontalEdge(): it draws the bottom or top edge of the square it depends on the size of the map during the implementation of this method I encountered two problems and they're:

1. if the height is odd and the height is less than the width, I had to subtract the minimum dimension by 1 not 2 (e.g., 9x16 map size the minimum dimension would be 9 so it's 8 "again as I mentioned in section 2.0" and subtracting it by 2 would become 6 that's why it was drawn 1 step earlier so we only subtract it by 1 which will become 7 and draw till the circled point in the picture below)

Subtracting by 2 (wrong)

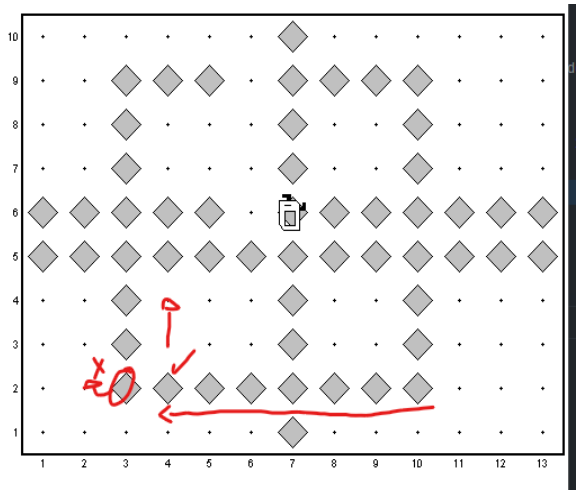


Subtracting by 1 (correct)

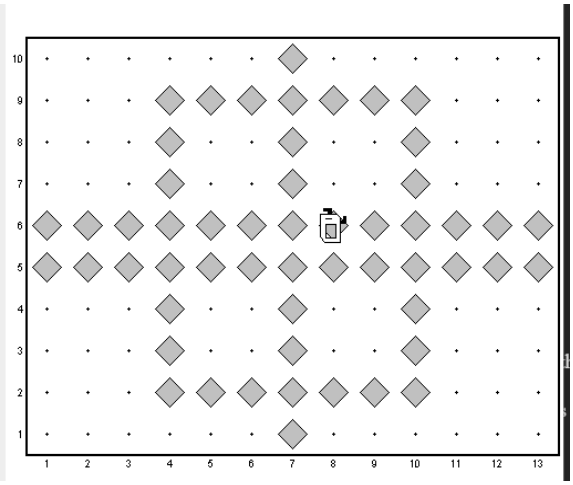


2. if the width of the map is odd and the width is larger than the height the minimum would initially be subtracted by 2 which will make it move one step further and draw the square incorrectly, so I had to subtract it by one more "3" (e.g., 10x13 map size the minimum dimension would be 10 so it's 9 "again as I mentioned in section 2.0" and subtracting it by 2 would become 7 that's why it moves one step further so we subtract it by 1 more which makes it 6 as shown in the picture below.

Subtracting by 2 (wrong)



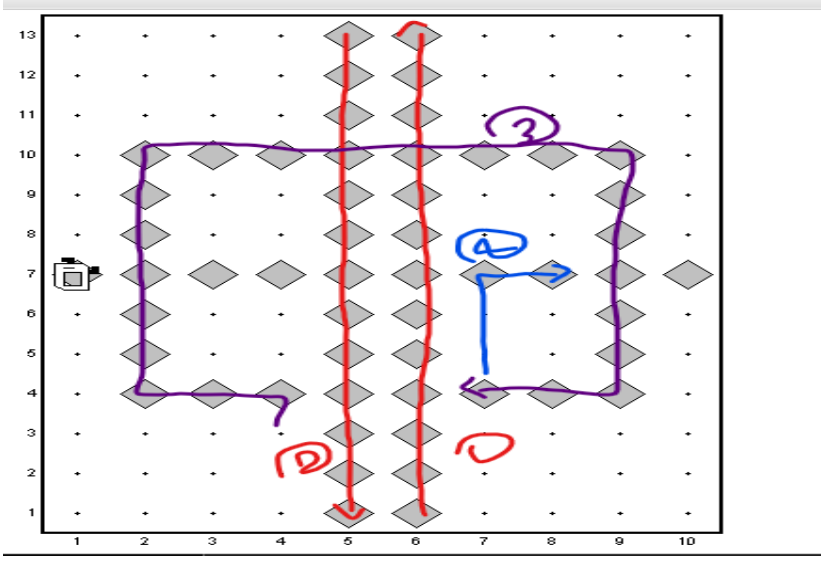
Subtracting by 3 (correct)



5.0 drawing horizontal lines

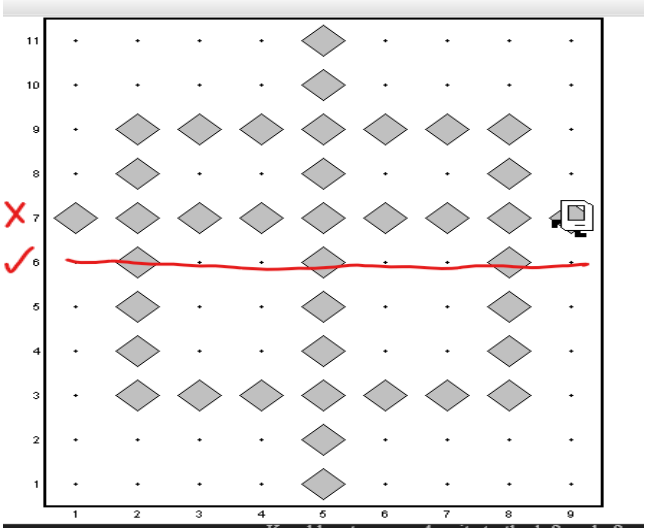
During this stage, we call the drawWidth() method, which makes two decisions:

1. Call the method drawOddWidth() if height is odd the method consists of this formula: height - calculateStart(findMinimumDimension() - 2), which will take the height of the map and subtract it by the square starting point, obviously as Karel will present there after drawing the square and then we loop through the result of this by calling the method skipPoints(counter) and skip points for minimizing the steps as show in step 4 in the picture below.

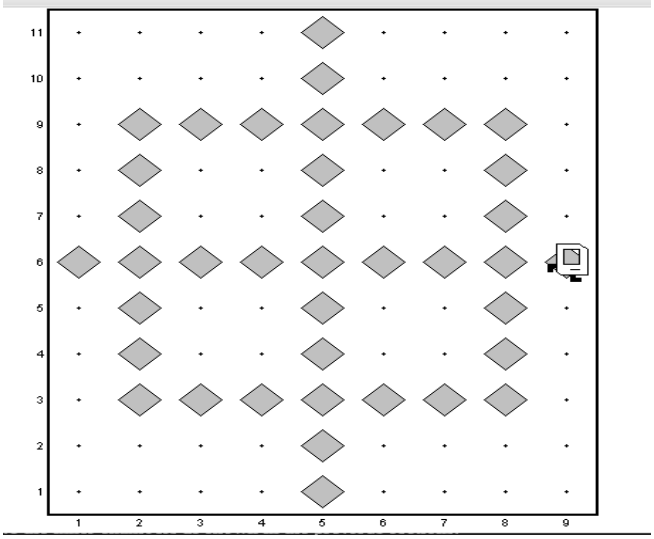


During the drawOddWidth() method, I encountered an error where the width is smaller than the height, resulting in incorrect horizontal line placement (1 unit earlier). To fix this, I increased the counter by 1 to draw the line one unit further.

Before adding 1 (wrong)

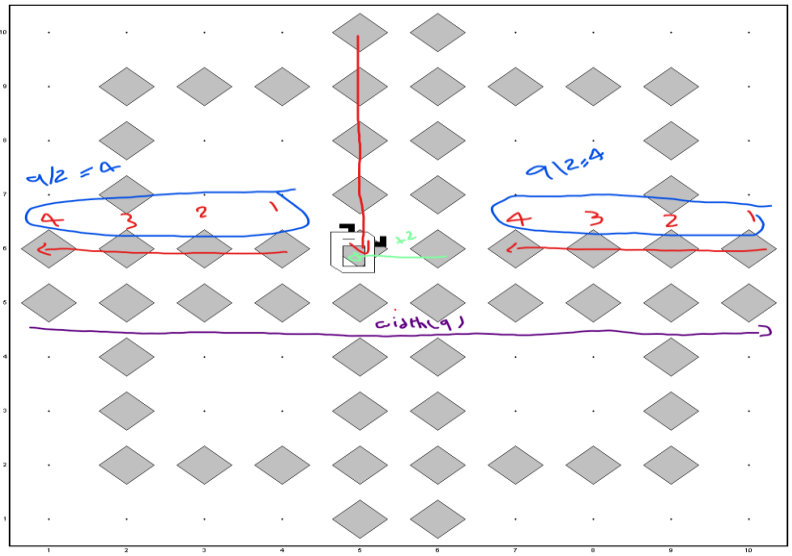


After adding 1 (correct)



2. in case height is not odd, call the drawEvenWidth() method which take upon two decisions:

If both the width and height are even, it follows a different implementation. After drawing the first half of the second vertical line, we invoke the drawWidth() method, which then calls the drawEvenWidth() method. In this case, the formula used is "((width / 2) * 2) + (width + 2)". Let me clarify this using the picture below. After drawing the first half, Karel stops at the current position shown in the picture. As you can see, Karel needs to move 4 units to the left and, after making a full turn, another 4 units to the left (indicated by the red lines). The ((width / 2) * 2) portion of the formula represents this movement. The width, circled in blue, is accounted for by the second half of the formula. The +2, represented by the green line, ensures that Karel returns to the second vertical line to draw its other half. After this, the method proceeds to draw the inner chambers, as shown in the previous sections.



- If the width is odd, we encounter a problem mentioned in section 3.2. When the width is both odd and larger than the height, I had to decrease the counter by 1. We used the skipPoints(counter) method mentioned in the drawOddWidth() method. After drawing the inner chambers, we use the formula "2 * width - 1" to draw the horizontal lines. Let me clarify this with the picture below. As you can see, after drawing the inner chambers, we draw the vertical lines with numbers 1, 2, and 23 on the other side. This is calculated using the formula "(2 * 12 - 1 = 23)". Since the loop condition of the drawLines() method is "greater than or equal to 0", it moves one step further and reaches 24 (one step after 23) before stopping.