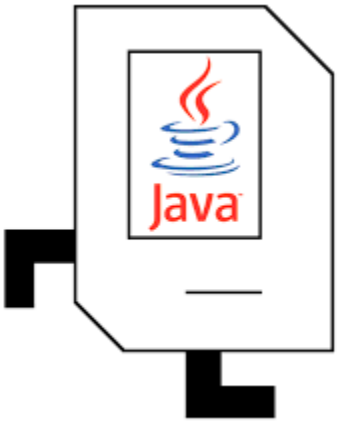


Karel The Robot Solution Report

By

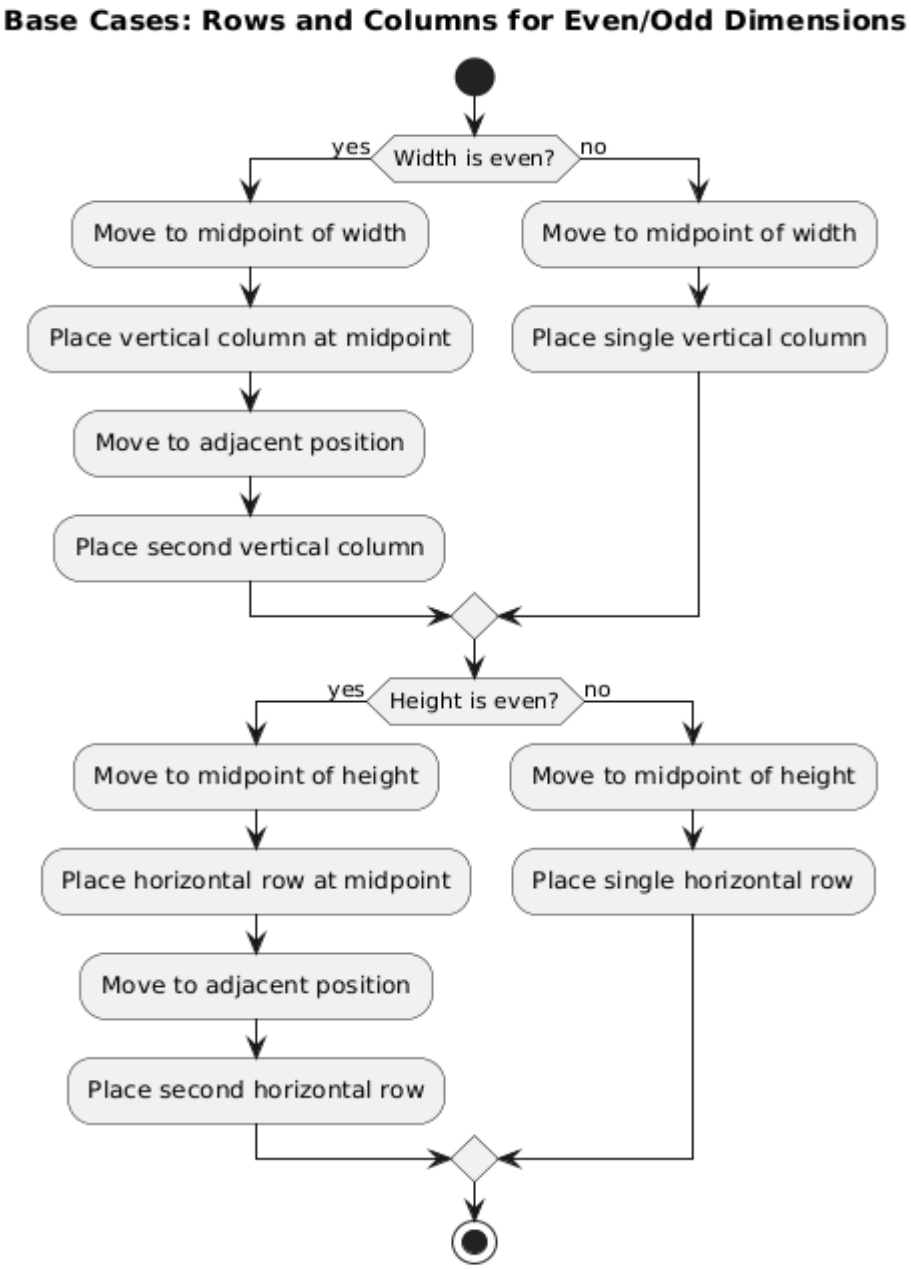
Abdallah Mohammad Abu Hussein

Software Engineering Graduate - 2024 | University Of Petra

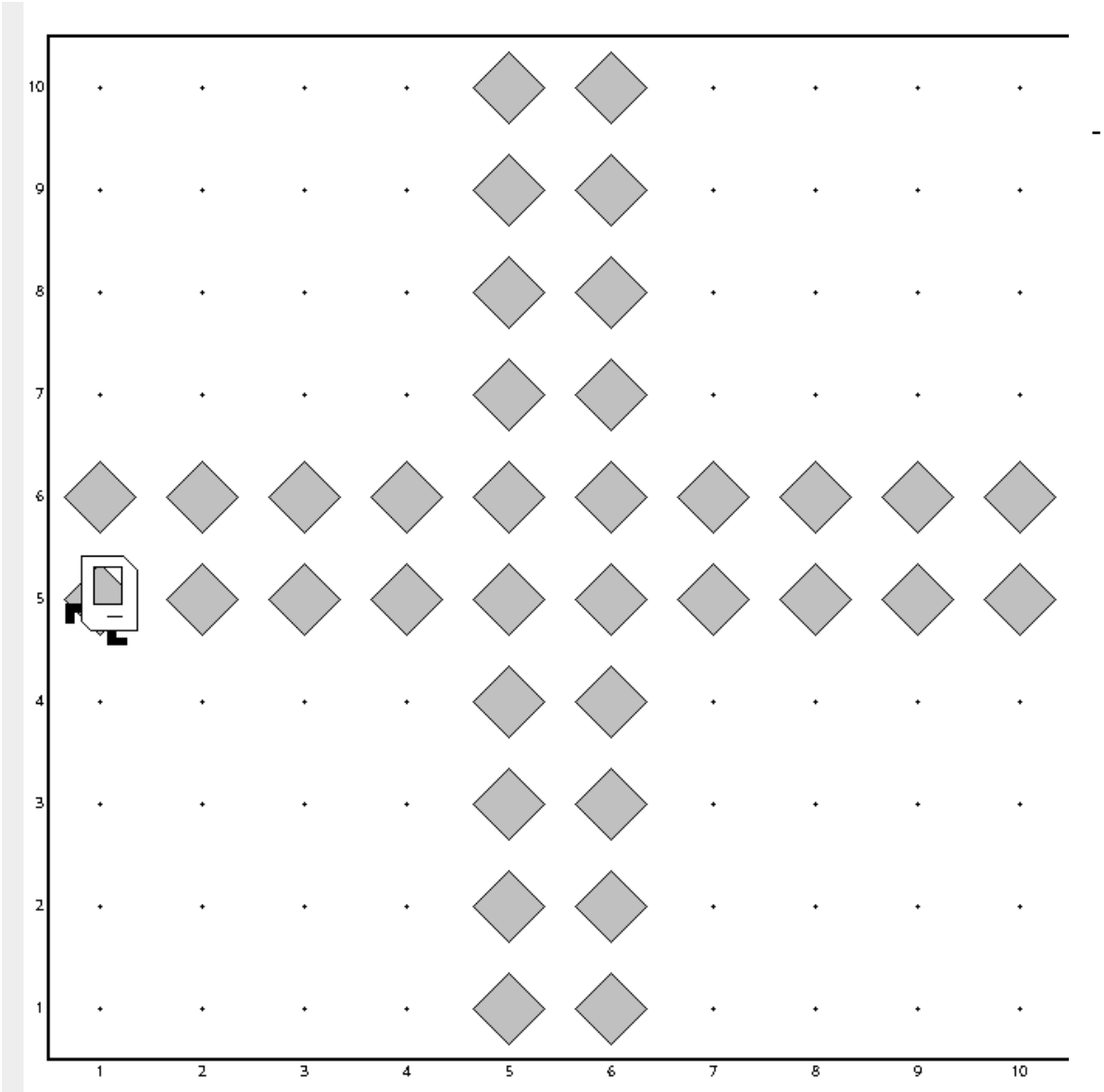
Introduction

This Assignment really fits the description of “challenging ease”, I did not anticipate that it will take this much effort, however it was a good learning experience, my solution handles various cases, including grids with special heights and widths, while optimizing movement and beeper placement.

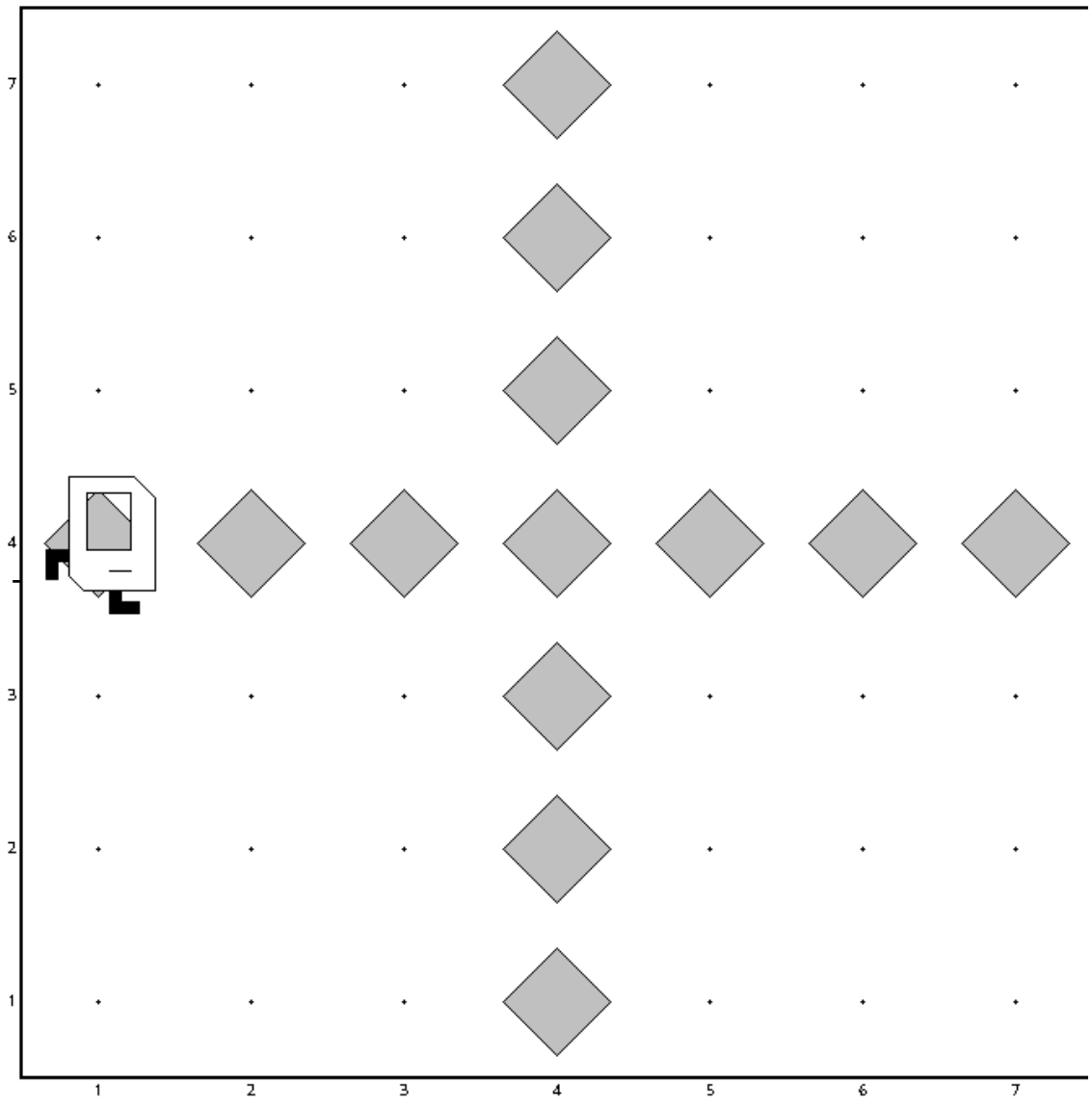
Base Cases



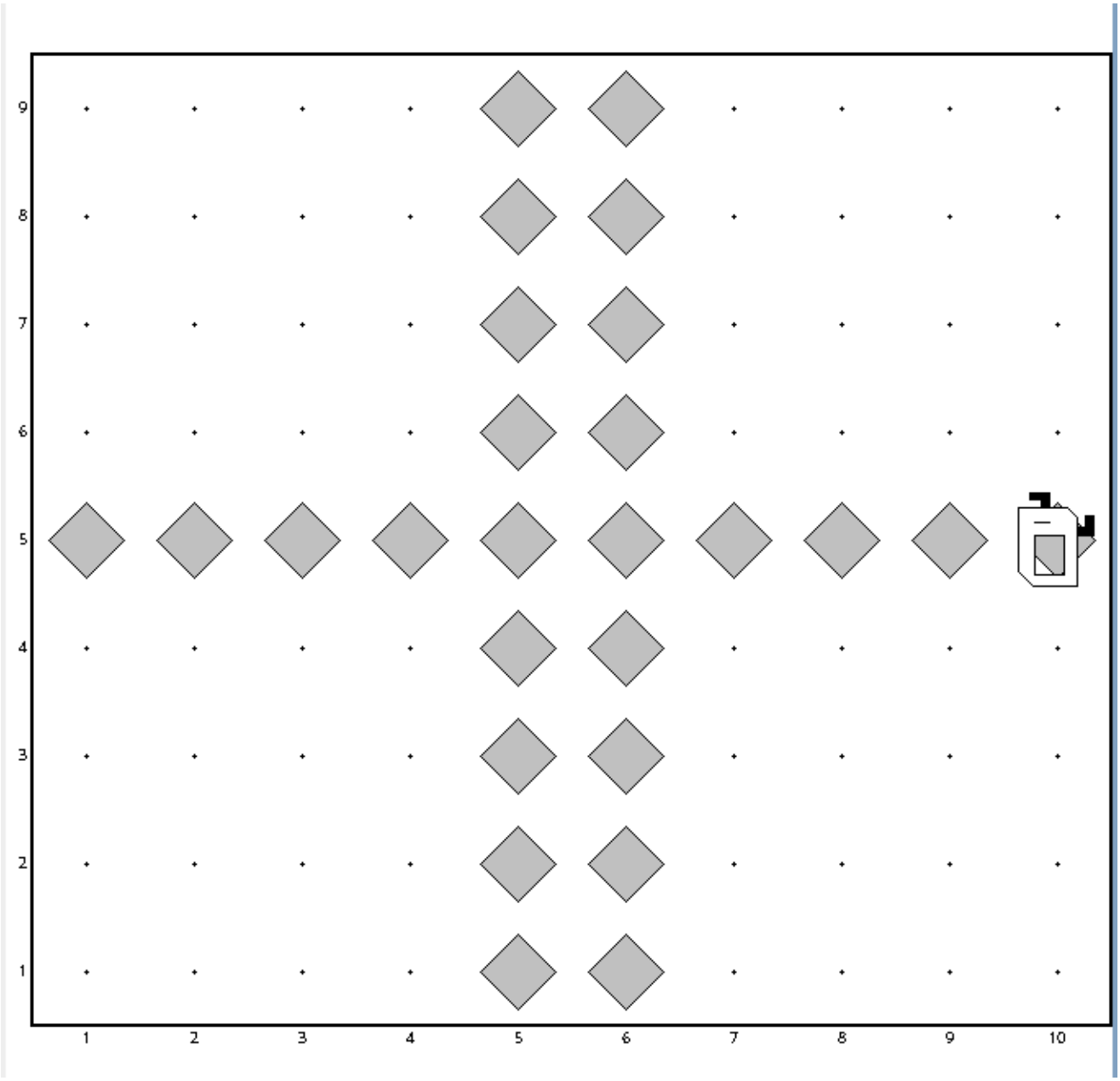
- **Even Width and Even Height**



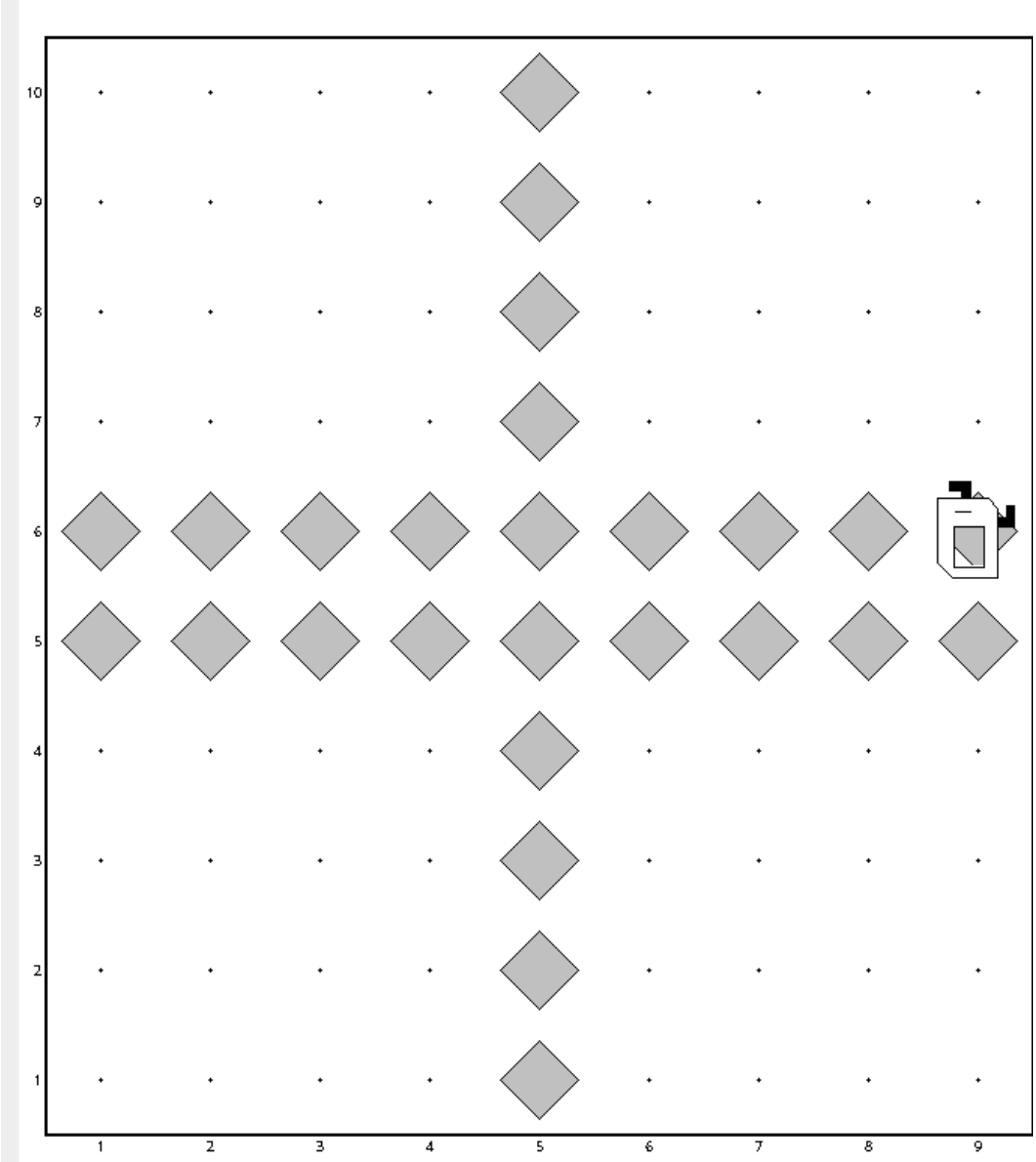
- **Odd Width and Odd Height**

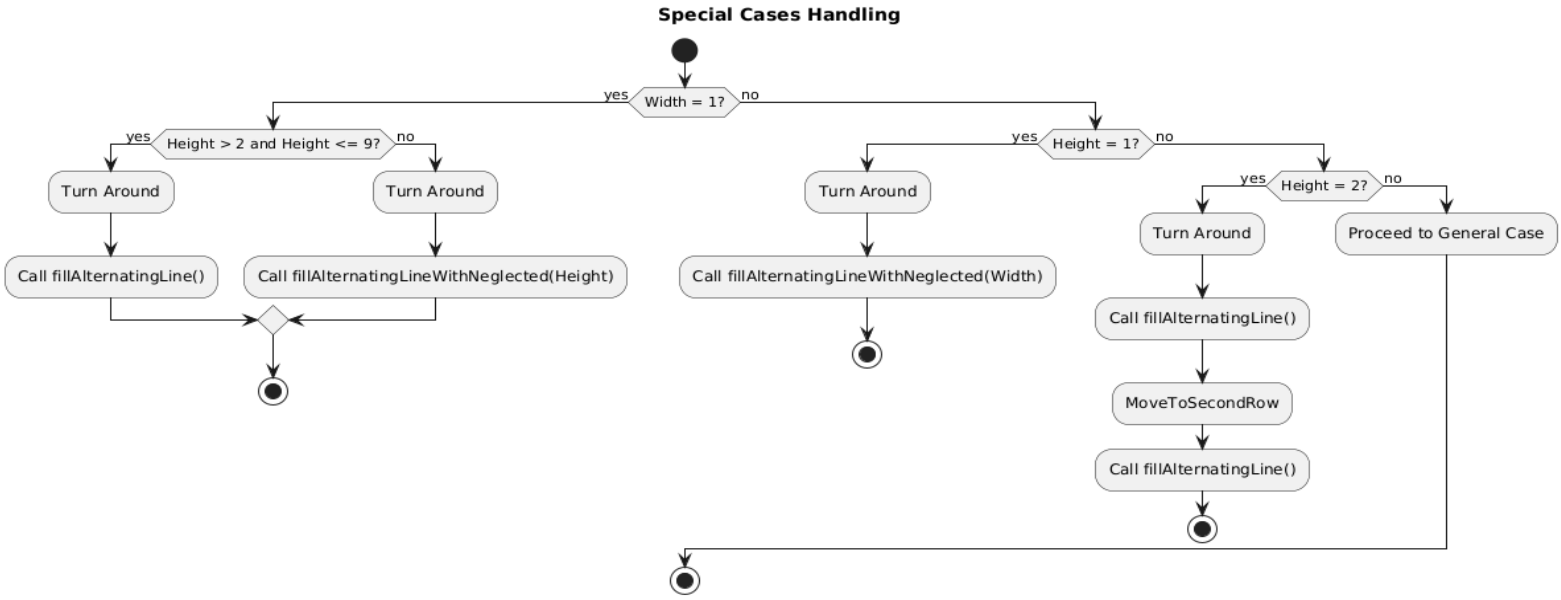


- Even Width and Odd Height



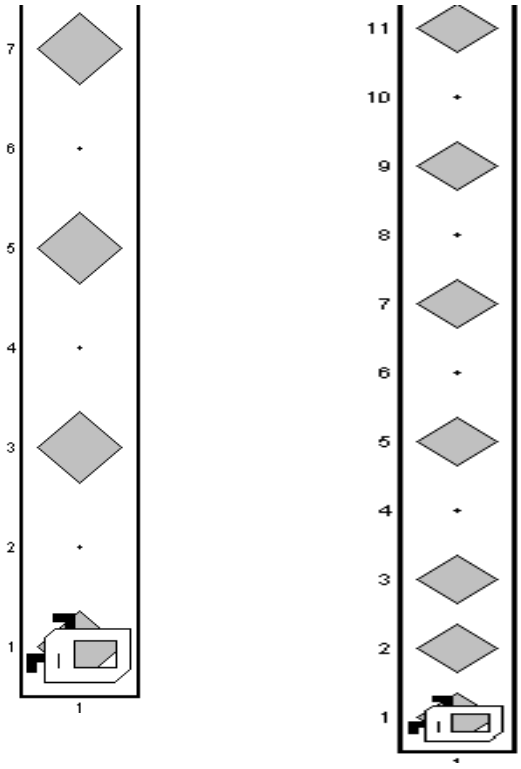
- **Odd Width and Even Height**



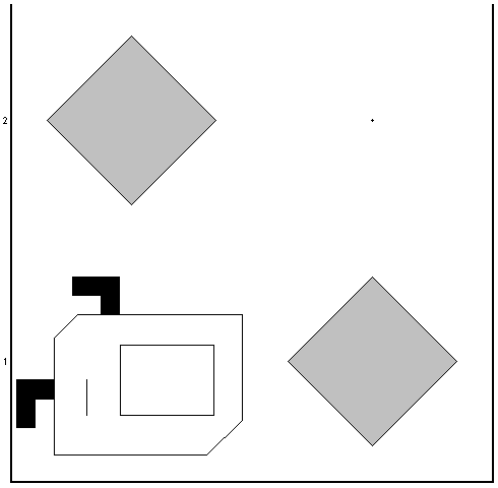


Special Cases and Their Effects

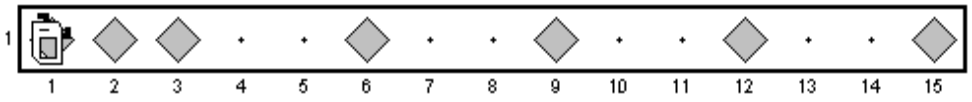
1. **Width = 1**:
    - **Height between 3 and 9**: Place alternating beepers along the height.
    - **Height > 9**: Divide the height into four chambers and handle neglected squares.



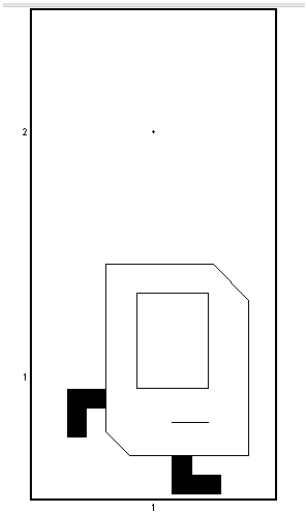
2. **Width = 2**:
    - **Height > 2**: Place two vertical columns with alternating beepers.

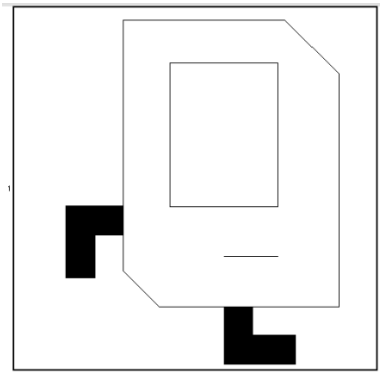


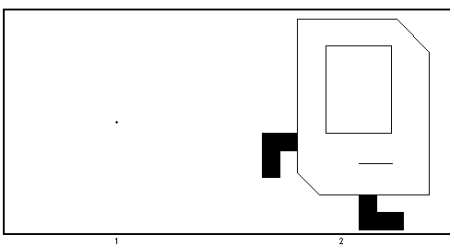
3. **Height = 1**:
    - **Width > 2**: Place alternating beepers along the width.



4. **One-Chamber Case**:
    - Grids like 1x1, 1x2, or 2x1: No action as the grid is already a single chamber







Code Break Down

1. run Method

```
public void run() {   Abdallah Abu Hussein
    analyzeAndBuild();
}
```

Explanation:
The main entry point of the program. It invokes the analyzeAndBuild method to analyze the grid dimensions and handle special and general cases.

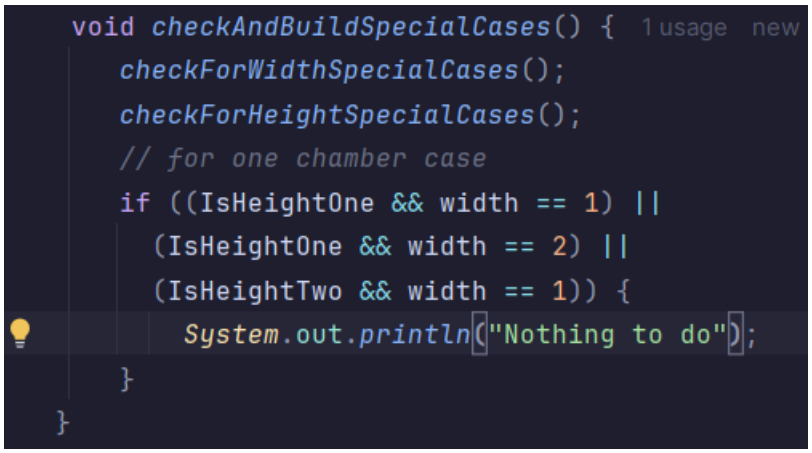
2.**analyzeAndBuild Method**

```
private void analyzeAndBuild() {  1 usage  new *
    checkAndBuildSpecialCases();
    // I check for only the height special cases since I do
    // to waste moves getting the full height
    if (width > 2 && !IsHeightTwo && !IsHeightOne) {
        buildFourChambers();
    }
    System.out.println("Height : " + height);
    System.out.println("Total Moves : " + movesCounter);
}
```

Explanation:

1. Handles special cases (width = 1, height = 1 or 2) via checkAndBuildSpecialCases.
2. For larger grids, invokes buildFourChambers to divide the grid into four chambers.
3. Logs the height and total moves for debugging.

3. checkAndBuildSpecialCases Method



Explanation:

This method acts as a controller for handling all special cases for the grid's width and height dimensions. It delegates the logic to specific methods and ensures edge cases are properly addressed.

Key Steps:

1. Check Width Special Cases:
    - Calls checkForWidthSpecialCases() to handle scenarios where:
        - Width = 1: Handles narrow grids with alternating or neglected beepers.
        - Width = 2: Places two vertical columns of beepers.
2. Check Height Special Cases:
    - Calls checkForHeightSpecialCases() to handle scenarios where:
        - Height = 1: Places alternating beepers along the width.
        - Height = 2: Places two horizontal rows of beepers.
3. Handle One-Chamber Cases:
    - Checks if the grid is effectively one chamber:
        - Examples: 1x1, 1x2, or 2x1 grids.
    - Prints "Nothing to do" and skips further operations for such grids.

4. **checkForWidthSpecialCases Method**

```
void checkForWidthSpecialCases() {  1 usage   Abdallah
    // check for width = 1 special case
    if (frontIsBlocked()) {
        set_height();
        if (height > 2 && height <= 9) {
            turnAround();
            fillAlternatingLine();
        }
        //width is one and height is n where n > 2
        else if (height > 2) {
            turnAround();
            fillAlternatingLineWithNeglected(height
        }
    } else {
        //the width is not one special case it
        //checks if height is a special case the
        //then set width
        checkIfHeightIsNotOneOrTwo();
        set_width();
    }

    // when width = 2 special case
    if (width == 2 && !IsHeightOne) {
        turnLeft();
        fillAlternatingLine();
        turnLeft();
        moveAndCount();
        turnLeft();
        fillAlternatingLine();
    }
}
```

Explanation:

Handles special cases for the width of the grid:

1. **Width = 1:**
    - Calls set_height() to determine height.
    - For heights between 3 and 9: Alternating beepers are placed along the height.
    - For heights > 9: Divides height into four chambers with neglected squares.

2. **Width = 2:**
    - Creates two vertical columns with alternating beepers.

3. **For other widths:**
    - Calls checkIfHeightIsNotOneOrTwo() to identify special cases in height.
    - Sets the width for general case

5. **checkForHeightSpecialCases Method**

```
void checkForHeightSpecialCases() {  1 usage  new *
    // if height is one special case
    if (IsHeightOne && width > 2) {
        turnAround();
        fillAlternatingLineWithNeglected(width);
    }
    //height 2 special case
    if (IsHeightTwo && width > 2) {
        turnAround();
        fillAlternatingLine();
        turnRight();
        moveAndCount();
        turnRight();
        fillAlternatingLine();
    }
}
```

Explanation:
Handles special cases for the **height** of the grid:

1. **Height = 1**:
    - For widths > 2, alternating beepers are placed along the width.
2. **Height = 2**:
    - For widths > 2, two horizontal rows are created with alternating beepers.

6. checkIfHeightIsNotOneOrTwo Method

```
void checkIfHeightIsNotOneOrTwo() {  2 usages  Abdallah Abu Hussein *
    turnLeft();
    if (frontIsBlocked()) {
        System.out.println("Height is one ");
        turnRight();
        IsHeightOne = true;
        height = 1;
    } else if (frontIsClear()) {
        // move to next squar
        moveAndCount();
        if (frontIsBlocked()) {
            // check blocked after moving one square, height is 2
            IsHeightTwo = true;
            height = 2;
        }
        // Turn back to return to the starting position
        turnAround();
        moveAndCount();
        turnLeft();
    }
}
```

Explanation:

Determines if the height is a special case:

1. **Height = 1**:
    - If Karel cannot move north from the starting position, sets IsHeightOne = true.
2. **Height = 2**:
    - If Karel can move one step north but encounters a wall after that, sets IsHeightTwo = true.
3. Returns control to the starting position after checking.

7. fillAlternatingLine Method

```
void fillAlternatingLine() {  5 usages  Abdallah Abu Hussein *
    boolean placeBeeper = true; // Start with placing a beeper
    while (frontIsClear()) {
        if (placeBeeper) {
            putBeeperSafely();
        }
        moveAndCount();
        // Change between placing and not placing a beeper
        placeBeeper = !placeBeeper;
    }
    // Place a beeper on the last position if needed
    if (placeBeeper && !beepersPresent()) {
        putBeeperSafely();
    }
}
```

Explanation:

The fillAlternatingLine method is designed to place beepers in an alternating -Like Chess Board -pattern along a single line in Karel's grid. It ensures that every other square contains a beeper, creating a visually distinct and efficient layout. This method can be used to handle both horizontal and vertical lines.

- **Initialization:**
  - A boolean placeBeeper determines whether a beeper is placed on the current square. It starts as true**.**
- **While Loop:**
  - The method iterates over the line until Karel encounters a wall (frontIsClear() is false).
  - **Action in Loop:**
    - If placeBeeper is true, a beeper is placed on the current square using putBeeperSafely().
    - Karel moves forward using moveAndCount(), which also tracks total moves.
    - The placeBeeper variable toggles to alternate placement.
- **Final Placement:**
    - After exiting the loop, if placeBeeper is still true, a beeper is placed on the last square to maintain the pattern.

8. Fill_Alternating_Line_With_Neglected Method

```
void fillAlternatingLineWithNeglected(int length) {  2 usages   Abdallah Abu Hussein *
    int chambers = 4; // used to divide the line into 4 chambers
    int chamberSize = length / chambers; // Size of each chamber

    int counter = 0; // Tracks position

    while (counter < length) {
        // Place beepers in chamber boundaries or handle extra squares
        if (counter % chamberSize == 0 || counter >= (chambers * chamberSize)) {
            putBeeperSafely(); // Place a beeper at chamber boundaries or extra squares
        }
        // Move to the next position
        counter++;
        if (frontIsClear()) {
            moveAndCount();
        }
    }
}
```

NOTE: This is Very Important Method for Special cases Its explanation will be more detailed

This method divides a line into 4 equal chambers and places beepers at their boundaries. It also handles any extra squares left at the end of the line to maintain consistent spacing.

How It Works

1. **Initialization**:
   - chambers: Defines how many sections the line is divided into (4 in this case).
   - chamberSize: Calculates the size of each chamber by dividing the line's length by the number of chambers.
   - counter: Tracks Karel's position along the line.
2. **While Loop**:
   - Iterates through the line until Karel has moved across its full length (counter < length).
   - **Conditions for Placing Beepers**:

     counter % chamberSize == 0

     This condition ensures that beepers are placed at the start of each chamber.

     - The modulo operator (%) checks if the current position (counter) is divisible by the chamber size.
     - If counter % chamberSize == 0, it means Karel is at the beginning of a new chamber.

- **Example**: For a line of length 10 divided into 4 chambers (chamberSize = 2):
    - **Chamber Boundaries**: Positions 0, 2, 4, 6, 8.
    - Beepers are placed at these positions to mark the start of each chamber.

2. counter >= (chambers * chamberSize)

This condition handles any leftover squares that don't fit evenly into the defined chambers.

- **What does it do?**
    - It checks if the current position (counter) is beyond the last calculated chamber boundary.
    - If the counter is greater than or equal to the total space allocated for the chambers (chambers * chamberSize), it places a beeper.
- **Why is it important?**
    - In cases where the line length isn't perfectly divisible by the number of chambers, this condition ensures no part of the line is neglected.
    - It adds a beeper in any remaining squares at the end of the line.

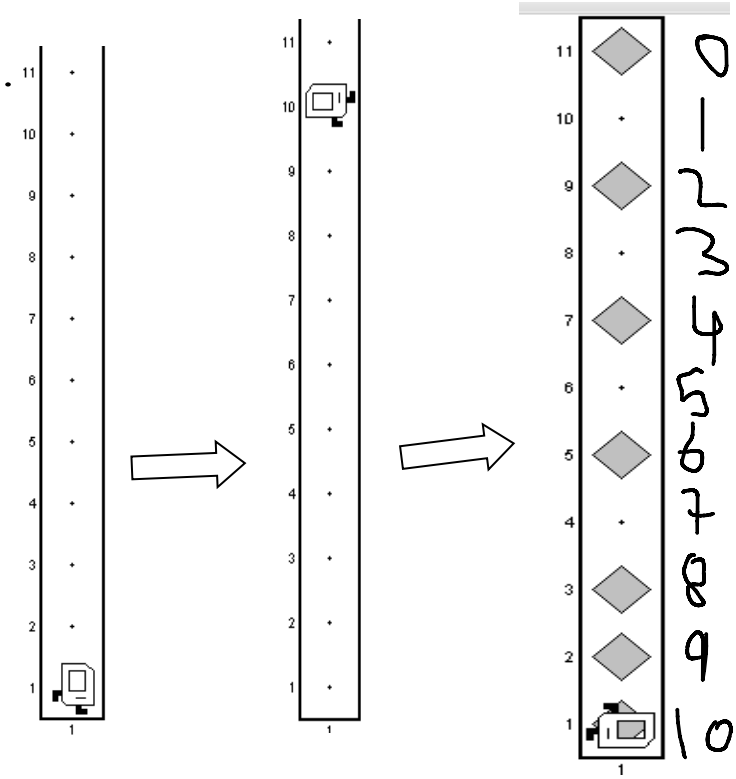
Combined Effect

The two conditions work together to ensure:

1. Beepers are placed at regular intervals to mark the boundaries of chambers.
2. Any leftover squares are accounted for, maintaining a consistent pattern across the entire line.

- **Moving Forward**:
    - After each action, the counter is incremented, and Karel moves to the next square using moveAndCount().

Example:



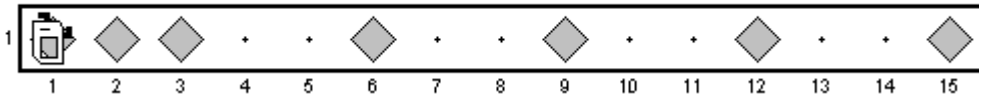
```
Height : 11
chamberSize = 2
The Full Size of not neglected cells =
Chambers * chamberSize = 8
Placed a beeper at Counter : 0
Placed a beeper at Counter : 2
Placed a beeper at Counter : 4
Placed a beeper at Counter : 6
Placed a beeper at Counter : 8
Placed a beeper at Counter : 9
Placed a beeper at Counter : 10
Height : 11
Total Moves : 22
```

Chart Representation for fillAlternatingLineWithNeglected (1x11 Example)

Step	Explanation	Details
Step 1: Divide the Line	Divide the line into 4 chambers based on its length.	- Line Length: 11 - Chambers: 4 - Chamber Size: 11 / 4 = 2 (chamber with beeper spans 2 positions).
Step 2: Place Beepers	Place beepers at the boundaries of the chambers and in the neglected squares.	- **Chamber Boundaries**: Beepers placed at positions 0, 2, 4, 6. - **Neglected Squares**: Beepers placed at positions 8, 9, 10.
Step 3: Result	Visual representation of the line with beepers.	**Pattern**: BBB.B.B.B.B - First chamber spans positions 0–2 (BBB). - Subsequent chambers span positions 3–7 (B). - Neglected squares span positions 8–10 (B.B).

The Same goes for Other Cases:

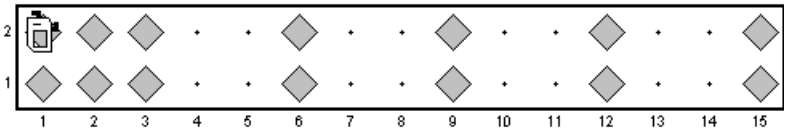
- **Height is one**




```
Height is one
Width : 15
chamberSize With beepers = 3
The Full Size of not neglected cells =
Chambers * chamberSize = 12
Placed a beeper at Counter : 0
Placed a beeper at Counter : 3
Placed a beeper at Counter : 6
Placed a beeper at Counter : 9
Placed a beeper at Counter : 12
Placed a beeper at Counter : 13
Placed a beeper at Counter : 14
Height : 1
Total Moves : 14
```

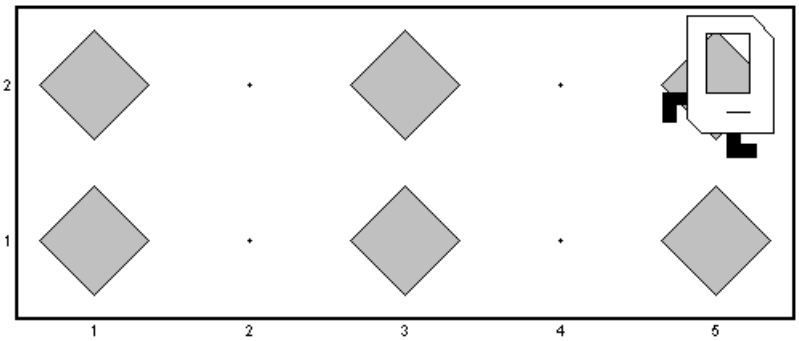
- **Height = 2**

Width > 9



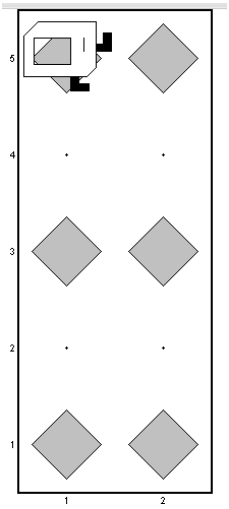

```
//height 2 special case
if (IsHeightTwo && width > 2 && width <= 9) {
    turnAround();
    fillAlternatingLine();
    turnRight();
    moveAndCount();
    turnRight();
    fillAlternatingLine();
}
else if(IsHeightTwo && width < 9) {
    turnAround();
    fillAlternatingLineWithNeglected(width);
    turnRight();
    moveAndCount();
    turnRight();
    fillAlternatingLineWithNeglected(width);
}
```

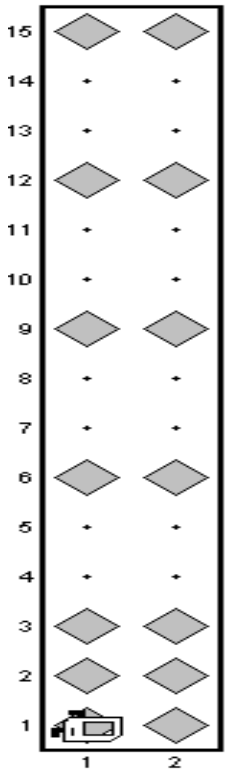
Width < 9



- **Width = 2**
  we just call fill Alternating method one for one column then Move to the second line and call it again

- Height > 9
- Height <=9

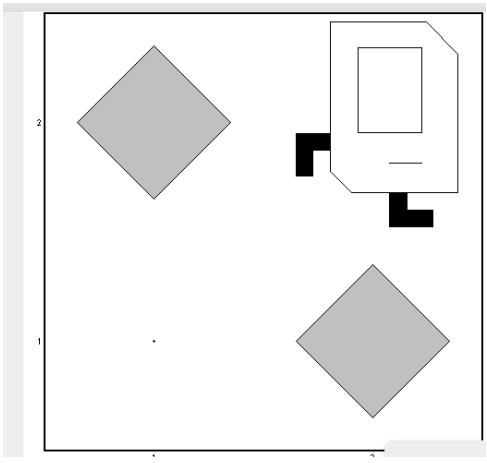




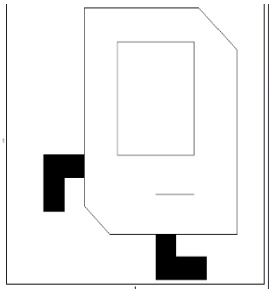
```
// when width = 2 special case
if (width == 2 && !IsHeightOne) {
    set_height();
    if (height >= 2 && height <= 9) {
        turnAround();
        fillAlternatingLine();
        turnRight();
        moveAndCount();
        turnRight();
        fillAlternatingLine();
    } else {
        turnAround();
        fillAlternatingLineWithNeglected(height);
        // go to the second column
        turnRight();
        moveAndCount();
        turnRight();
        goToWall();
        turnRight();
        turnRight();
        //fill the second column
        fillAlternatingLineWithNeglected(height);
    }
}
```

Other Cases :

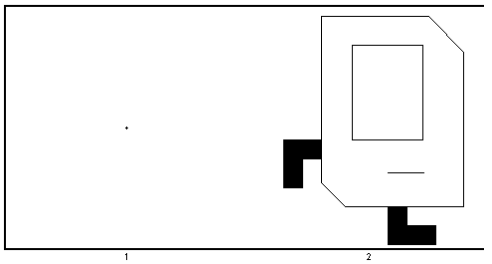
2x2



1x1



1x2



Base Cases Code And Helper Methods:

BuildFourChambers

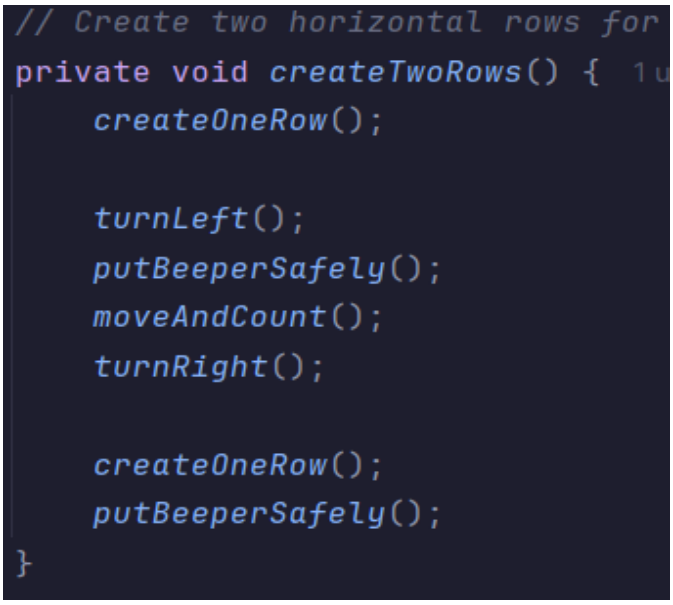
```
private void buildFourChambers() {  1 usage   Abdallah Abu Hussein +1 *
    if (width % 2 == 0) {
        goToMid(width);
        createTwoColumns();
    } else {
        goToMid(width);
        createOneColumn();
    }
    if (this.height != 2 && this.height % 2 == 0) {
        goToMid(height);
        turnLeft();
        goToWall();
        turnAround();
        createTwoRows();
    } else {
        goToMid(height);
        turnLeft();
        goToWall();
        turnAround();
        createOneRow();

    }
}
```

Explanation:

- Calculates the height of the grid.
- Handles special cases for height = 1 or 2 using checkIfHeightIsNotOneOrTwo.
- For general cases, counts the number of steps from the bottom to the top to determine height.

Methods For Building Rows

```
private void createOneRow() {  3 usages
    int counter = 1;
    while (counter < this.width) {
        putBeeperSafely();
        moveAndCount();
        if (frontIsBlocked()) {
            turnAround();
        }
        putBeeperSafely();
        counter++;
    }
}
```



```
// Create two horizontal rows for
private void createTwoRows() {  1 u
    createOneRow();

    turnLeft();
    putBeeperSafely();
    moveAndCount();
    turnRight();

    createOneRow();
    putBeeperSafely();
}
```

- **createOneRow**

Details:

- Creates a single horizontal row by placing beepers across the width of the grid.
- Handles edge cases at the walls.

- **createTwoRows**

Details:

- Creates two horizontal rows for grids with even heights.
- Moves down one square after completing the first row to start the second.

Methods For Creating Columns

```
private void createOneColumn() {  1 usage  Abd
    int counter = 1;
    turnRight();
    while (frontIsClear() & height != 2) {
        counter++;
        putBeeperSafely();
        moveAndCount();
    }
    putBeeperSafely();
    this.height = counter;
    if (height == 2) {
        IsHeightTwo = true;
        return;
    }
}
```

```
private void createTwoColumns() {  1 usage  Abdallah
    int counter = 1;
    turnRight();
    while (frontIsClear()) {
        counter++;
        putBeeperSafely();
        moveAndCount();
    }

    this.height = counter;
    if (height == 2)
        IsHeightTwo = true;
    System.out.println("Height : " + this.height);
    putBeeperSafely();
    turnRight();
    moveAndCount();
    turnRight();
    while (frontIsClear()) {
        putBeeperSafely();
        moveAndCount();
    }
    putBeeperSafely();
}
```

- **createOneColumn**

Details:

- Creates a single vertical column by placing beepers across the height of the grid.
- Checks and handles the special case where height = 2.

- **createTwoColumns**

Details:

- Creates two vertical columns for grids with even widths.
- Places beepers in the first column, moves one square to the side, and repeats for the second column

Helper Methods:

```
void set_width() {  1 usage  Abdallah Abu Hussein +1 *
    int counter = 1;
    while (frontIsClear()) {
        move();
        counter++;
    }
    this.width = counter;
    System.out.println("Width : " + width);

}
```

```
void goToWall() {  4 usages  new *
    while (frontIsClear()) {
        moveAndCount();
    }
}
```

```
void putBeeperSafely() {  13 usages  Abdallah Abu Hussein +1
    if (!beepersPresent())
        putBeeper();
}

void moveAndCount() {  18 usages  Abdallah Abu Hussein +1
    move();
    movesCounter++;

}
```

```java
void goToMid(int size) {  4 usages  Abdallah Abu Hussein +1
    if (size % 2 == 0) {
        // For even sizes
        int mid = size / 2;
        turnAround();
        for (int i = 1; i <= mid; i++) {
            moveAndCount();
        }
        System.out.println("Even Size Midpoint Reached at Step: " + mid);
    } else {
        // For odd sizes
        int mid = (size / 2) + 1;
        turnAround();
        for (int i = 1; i < mid; i++) {
            moveAndCount();
        }
        System.out.println("Odd Size Midpoint Reached at Step: " + mid);
    }
}
```

- **goToWall**

Details:

- Moves Karel to the nearest wall in the current direction.
- Increments the move counter for each step.

- **set_width**

Details:

- Calculates the grid's width by counting the number of steps from the leftmost square to the rightmost wall.
- Sets the width variable for further processing.

- **putBeeperSafely**

Details:

- Places a beeper on the current square only if no beeper is already present.
- Ensures efficient use of beepers.

- **moveAndCount**

Details:

- Moves Karel forward by one square.
- Increments the movesCounter to keep track of Karel's total movement.

- **goToMid**

Details:

- Moves Karel to the midpoint of a given dimension (width or height).
- Handles even and odd dimensions separately:
    - **Even**: Stops at the exact center.
    - **Odd**: Stops just before the center.