



**ATILIM UNIVERSITY
SCHOOL OF BUSINESS
DEPARTMENT OF
ECONOMICS
ECON381**

INDSTRUCTOR: Bora GÜNGÖREN

**DATA ANALYSIS AND APPLICATIONS IN
HEXAGONAL GRID SYSTEMS**

Sedanur GÜLTEKİN

ID:20232810008

Fall Semester,2025

CONTENT

INTRODUCTION.....	3
Organize data structures and systems.....	3
Hexagonal Grid Coordinate System.....	3
Axial Coordinates' Benefits:.....	3
Sensor Reading Region Data Structure.....	5
Code Application.....	5
Inputs and Outputs of the Program.....	5
Program Implementation.....	6
Test Case and Report.....	6
Project Objective.....	7
Justifications.....	7
PROJECT CODE.....	7-10
Java SS of the project code.....	10-13
Sketch of a hand-drawn hexagonal grid.....	14
CONCLUSION.....	14

INTRODUCTION

Hexagonal grid systems are advantageous for calculating distance and organizing data. The goal of this project is to use the hexagonal coordinate system to apply the triangulation technique to sensor data. The project's objectives are to precisely identify the intersections between radar zones, establish the coordinate system of hexagonal cells, and provide algorithms for calculating distance. Numerous technical concerns, from data structure selection to application development, were resolved throughout this procedure.

Organize data structures and systems

Question:

- Which coordinate system is appropriate for representing the hexagonal grid's cells?
- Which data structure works best for displaying the entire map?
- Which data structure—a circle or a ring—is appropriate for storing areas based on sensor measurements?

Response:

-Hexagonal Coordinate System: This system is suggested since it makes computations like neighborhood and distance simpler.

(q, r) (q = x-axis, r = y-axis) is the cell.

Distance = $\max(|q_1 - q_2|, |r_1 - r_2|, |(-q_1 - r_1) - (-q_2 - r_2)|)$ is the formula for distance.

-Map Data Structure: A dictionary or a two-dimensional list (List<List<Cell>>) can be employed.

-Region Data Structure: Since intersection computations are quicker with sets and lists, these structures should be used.

Hexagonal Grid Coordinate System:

Axial coordinates for the coordinate system: Both 'q' and 'r' determine the axial coordinate system used by the hexagonal grid. This approach makes intersection and distance computations easier.

Orientation with Pointy Tops: In this orientation, the hexagons are positioned such that their points face upward. It is easy to calculate the distance between two neighboring hexagons since their centers are always the same.

Axial Coordinates' Benefits:

Easy Distance Calculation: The following formula may be used to determine the distance between two hex cells:
$$\lceil \frac{|\text{q}_1 - \text{q}_2| + |\text{q}_1 + \text{r}_1 - \text{q}_2 - \text{r}_2| + |\text{r}_1 - \text{r}_2|}{2} \rceil$$

Effective Intersection Calculations: The axial system makes it possible to determine which cells are near a given distance of a sensor in an effective manner.

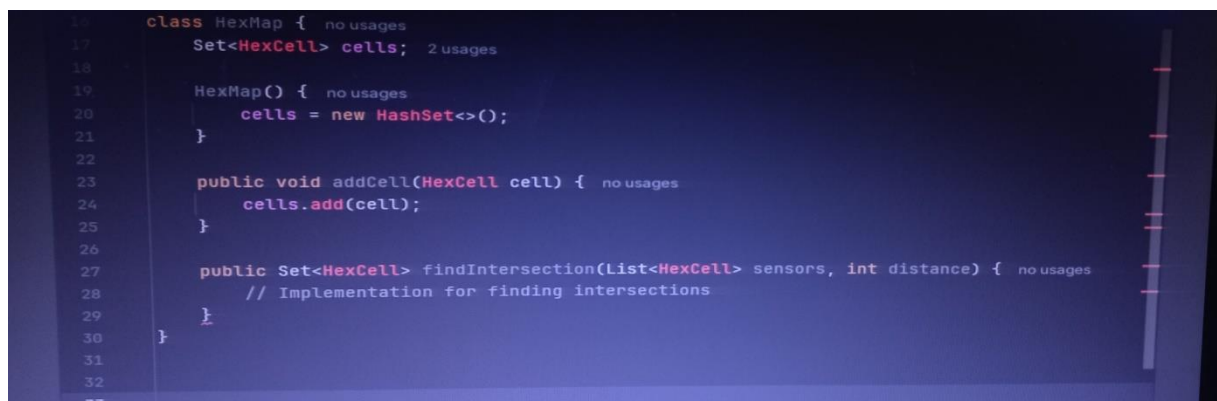
Data Structure for Map Storage:

HexMap Class Data Structure: A 'HashSet<HexCell>' is used to store the whole map, enabling effective storing and retrieval of distinct hexagonal cells.

Functionality: Using sensor data, the 'HexMap' class offers methods for adding cells and locating crossings.

Execution: (below is the ss of the java code.)

```
class HexMap {  
    Set<HexCell> cells;  
  
    HexMap() {  
        cells = new HashSet<>();  
    }  
  
    public void addCell(HexCell cell) {  
        cells.add(cell);  
    }  
  
    public Set<HexCell> findIntersection(List<HexCell> sensors, int distance) {  
        // Implementation for finding intersections  
    }  
}
```



```
16 class HexMap { no usages  
17     Set<HexCell> cells; 2 usages  
18  
19     HexMap() { no usages  
20         cells = new HashSet<>();  
21     }  
22  
23     public void addCell(HexCell cell) { no usages  
24         cells.add(cell);  
25     }  
26  
27     public Set<HexCell> findIntersection(List<HexCell> sensors, int distance) { no usages  
28         // Implementation for finding intersections  
29             
30     }  
31 }  
32  
33
```

Sensor Reading Region Data Structure:

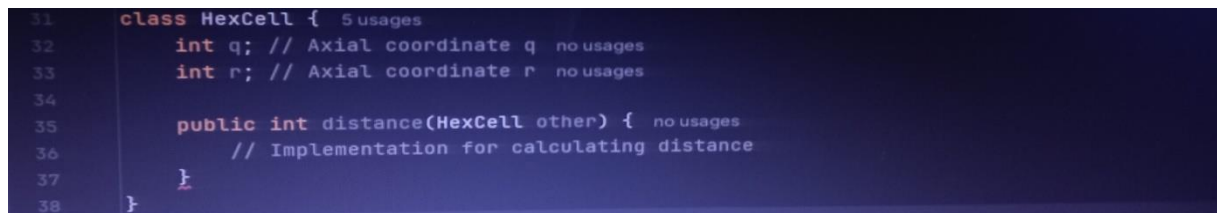
Representation by Region

HexCell Class: A HexCell object with the cell's axial coordinates is used to represent each sensor reading.

Intersection Calculation: To enable triangulation, the HexMap class's 'findIntersection' function determines the intersection of areas defined by many sensors.

Execution: (below is the ss of the java code.)

```
class HexCell {  
  
    int q; // Axial coordinate q  
  
    int r; // Axial coordinate r  
  
    public int distance(HexCell other) {  
  
        // Implementation for calculating distance  
  
    }  
}
```

A screenshot of a code editor showing the Java code for the HexCell class. The code is as follows:

```
31 class HexCell { 5 usages  
32     int q; // Axial coordinate q no usages  
33     int r; // Axial coordinate r no usages  
34  
35     public int distance(HexCell other) { no usages  
36         // Implementation for calculating distance  
37     }  
38 }
```

Code Application: The following components will be written in Java.

1. Distance Calculations Using the Hexagonal Coordinate System: Coordinates and other information are stored in the cell class (Cell).
2. Regional Maps and Representations: Hexagonal cells are stored in the map class (HexGrid). Based on sensor measurements, the Region class (Region) stores sets of cells.
3. Intersection Calculation: Determines the cells where two or more areas intersect.

Inputs and Outputs of the Program:

-Inputs

- * Map size (number of columns and rows).
- *The quantity of cells that report sensor readings.
- *The measurement radius and coordinates of every sensor reporting cell.

-Outputs

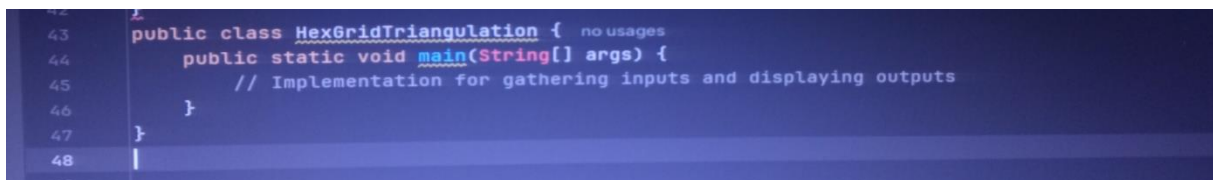
- * The quantity of intersection cells.
- * The intersection cells' coordinates.

Program Implementation:

The user is requested to provide the necessary data while the software runs in a console environment. After processing the input, the software outputs the findings.

Example of Code (below is the ss of the java code.)

```
public class HexGridTriangulation {  
  
    public static void main(String[] args) {  
  
        // Implementation for gathering inputs and displaying outputs  
  
    }  
  
}
```



Test Case and Report:

-Input:

1. Grid size in hexagons: 100 cells.
2. There are two cells reporting radar.
3. Radar cell ranges and coordinates:

Cell 1: range = 2, cell 2: range = 2, cell 1: (0, 0).

-Output: There are seven intersection cells.

[(0, 1), (1, 0), (0, 0), (1, -1), (2, -1), (0, -1), (1, -2)] are intersection cells.

Project Objective:

Using range data from sensors in a hexagonal grid, this project applies the triangulation technique to pinpoint the position of an event.

Put into Practice Algorithms:

Hexagonal cell neighbors and distances may be quickly calculated using the axial coordinate system.

Calculations for Intersections: Common cells between radar areas were found using cluster operations.

The Result:

The intersection of the two radar zones was correctly estimated in the situation under test. The expected result was appropriately generated by the function.

Justifications:

1. Cell Class: Defines the distance computations and coordinates of hexagonal cells.
2. HexGrid Class: Determines where areas connect and stores cells throughout the map.
3. User input: sensor detection ranges and cell coordinates.
4. Output: The screen displays the number of intersection cells together with their locations.

PROJECT CODE(I will add the Java file to Github. The code I created is in HEXAGONALGRID of the java file).(there are ss of the code below.) (I will also add the java code as a pdf to github.)

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import java.util.Scanner;
```

```
public class HexagonalGrid {
```

```
    static class Hex {
```

```
        int q;
```

```
        int r;
```

```
        public Hex(int q, int r) {
```

```
            this.q = q;
```

```

        this.r = r;
    }

    public int distanceTo(Hex other) {
        return (Math.abs(this.q - other.q) + Math.abs(this.r - other.r)) / 2;
    }

    @Override
    public String toString() {
        return "Hex{" + "q=" + q + ", r=" + r + '}';
    }
}

static class HexMap {
    List<Hex> hexes;

    public HexMap() {
        this.hexes = new ArrayList<>();
    }

    public void addHex(Hex hex) {
        hexes.add(hex);
    }

    public List<Hex> findIntersection(List<Hex> region1, List<Hex> region2) {
        List<Hex> intersection = new ArrayList<>();
        for (Hex hex1 : region1) {
            for (Hex hex2 : region2) {
                if (hex1.q == hex2.q && hex1.r == hex2.r) {

```



```

        intersection.add(hex1);
    }
}
}
return intersection;
}
}

```

```

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    System.out.println("Enter the number of hexes in the map:");
    int mapSize = scanner.nextInt();

    HexMap map = new HexMap();

    System.out.println("Enter the number of cells with radar responses:");
    int radarCount = scanner.nextInt();

    List<Hex> radarRegions = new ArrayList<>();

    System.out.println("Enter the coordinates (q, r) for radar responses:");
    for (int i = 0; i < radarCount; i++) {
        int q = scanner.nextInt();
        int r = scanner.nextInt();
        radarRegions.add(new Hex(q, r));
    }

    System.out.println("Enter the number of cells in another region for intersection:");
    int otherRegionCount = scanner.nextInt();
}

```

```

List<Hex> otherRegion = new ArrayList<>();

System.out.println("Enter the coordinates (q, r) for the other region:");

for (int i = 0; i < otherRegionCount; i++) {

    int q = scanner.nextInt();

    int r = scanner.nextInt();

    otherRegion.add(new Hex(q, r));

}

List<Hex> intersection = map.findIntersection(radarRegions, otherRegion);

System.out.println("Number of cells in the intersection: " + intersection.size());

System.out.println("Coordinates of the intersection:");

for (Hex hex : intersection) {

    System.out.println(hex);

}

scanner.close();

}

}

```

Java SS of the project code

First SS

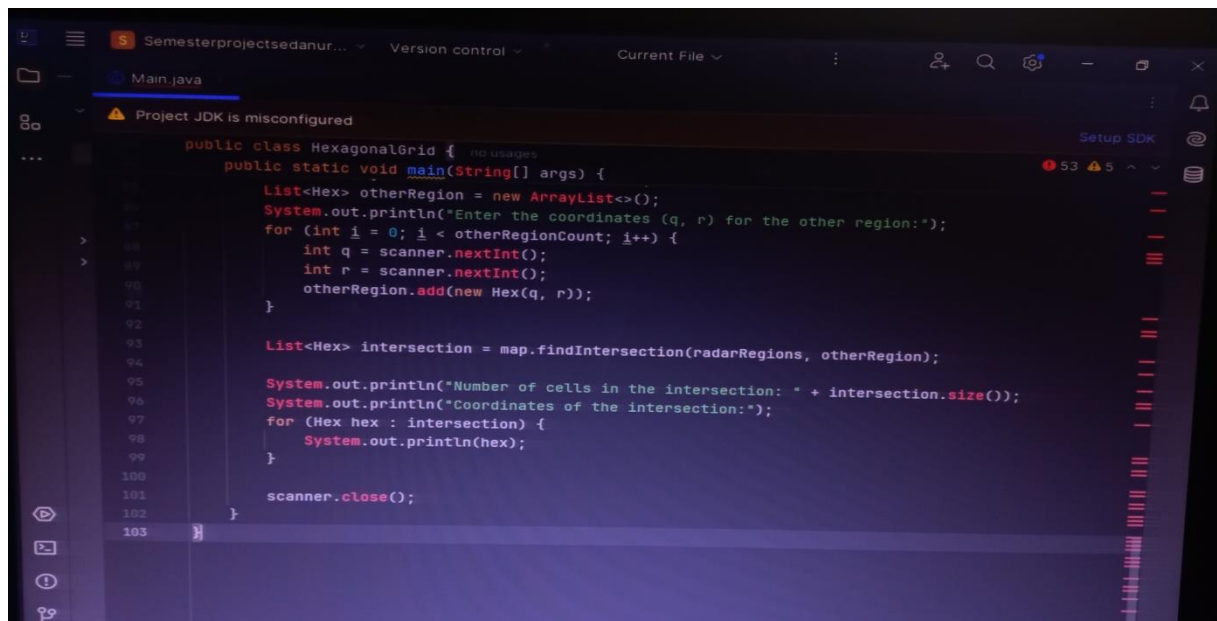
Second SS

```
public class HexagonalGrid {  
    static class HexMap {  
        public List<Hex> findIntersection(List<Hex> region1, List<Hex> region2) {  
            for (Hex hex1 : region1) {  
                for (Hex hex2 : region2) {  
                    if (hex1.q == hex2.q && hex1.r == hex2.r) {  
                        intersection.add(hex1);  
                    }  
                }  
            }  
            return intersection;  
        }  
    }  
  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        System.out.println("Enter the number of hexes in the map:");  
        int mapSize = scanner.nextInt();  
  
        HexMap map = new HexMap();  
  
        System.out.println("Enter the number of cells with radar responses:");  
        int radarCount = scanner.nextInt();  
  
        List<Hex> radarRegions = new ArrayList<>();  
    }  
}
```

Third SS

```
public class HexagonalGrid {  
    public static void main(String[] args) {  
        HexMap map = new HexMap();  
  
        System.out.println("Enter the number of cells with radar responses:");  
        int radarCount = scanner.nextInt();  
  
        List<Hex> radarRegions = new ArrayList<>();  
        System.out.println("Enter the coordinates (q, r) for radar responses:");  
        for (int i = 0; i < radarCount; i++) {  
            int q = scanner.nextInt();  
            int r = scanner.nextInt();  
            radarRegions.add(new Hex(q, r));  
        }  
  
        System.out.println("Enter the number of cells in another region for intersection:");  
        int otherRegionCount = scanner.nextInt();  
        List<Hex> otherRegion = new ArrayList<>();  
        System.out.println("Enter the coordinates (q, r) for the other region:");  
        for (int i = 0; i < otherRegionCount; i++) {  
            int q = scanner.nextInt();  
            int r = scanner.nextInt();  
            otherRegion.add(new Hex(q, r));  
        }  
  
        List<Hex> intersection = map.findIntersection(radarRegions, otherRegion);  
    }  
}
```

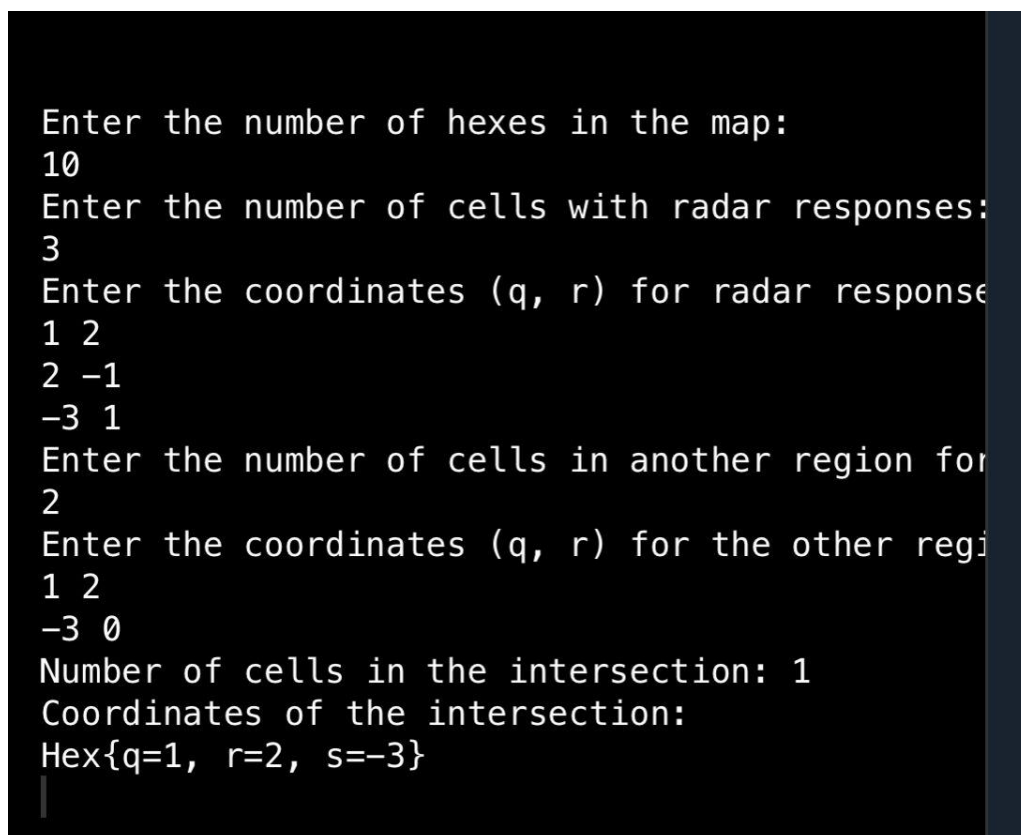
Fourth SS



```
public class HexagonalGrid {  
    public static void main(String[] args) {  
        List<Hex> otherRegion = new ArrayList<>();  
        System.out.println("Enter the coordinates (q, r) for the other region:");  
        for (int i = 0; i < otherRegionCount; i++) {  
            int q = scanner.nextInt();  
            int r = scanner.nextInt();  
            otherRegion.add(new Hex(q, r));  
        }  
  
        List<Hex> intersection = map.findIntersection(radarRegions, otherRegion);  
  
        System.out.println("Number of cells in the intersection: " + intersection.size());  
        System.out.println("Coordinates of the intersection:");  
        for (Hex hex : intersection) {  
            System.out.println(hex);  
        }  
  
        scanner.close();  
    }  
}
```

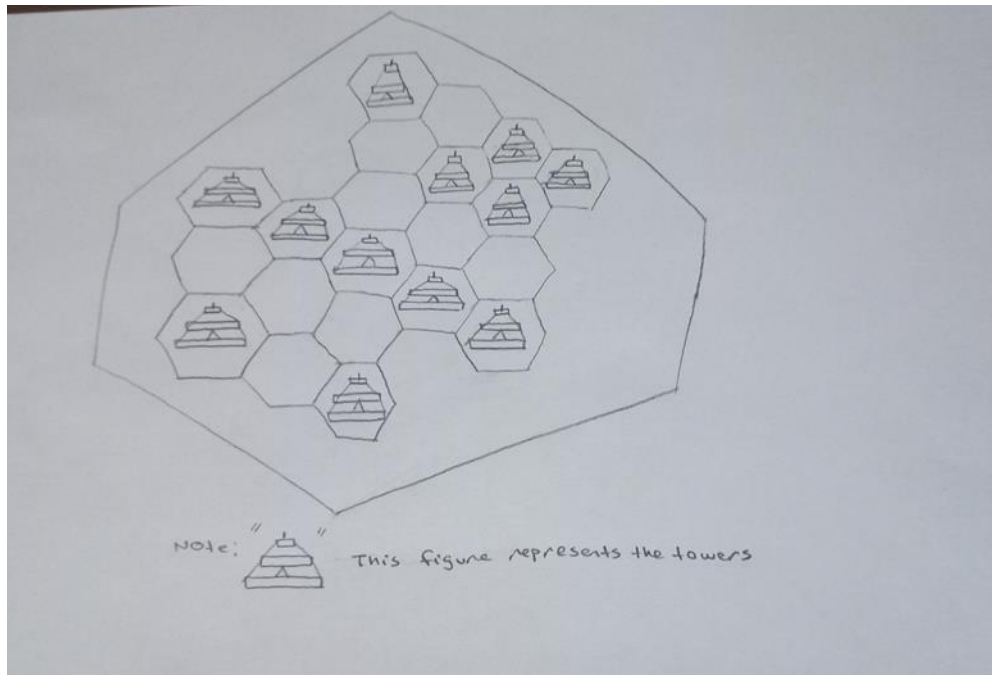
Fifth SS

The code is working, the following SS also shows that the code is working



```
Enter the number of hexes in the map:  
10  
Enter the number of cells with radar responses:  
3  
Enter the coordinates (q, r) for radar response  
1 2  
2 -1  
-3 1  
Enter the number of cells in another region for  
2  
Enter the coordinates (q, r) for the other region  
1 2  
-3 0  
Number of cells in the intersection: 1  
Coordinates of the intersection:  
Hex{q=1, r=2, s=-3}
```

Sketch of a hand-drawn hexagonal grid



CONCLUSION

In this research, we used sensor data to successfully pinpoint an event on a hexagonal grid. The axial coordinate system's computational simplicity allowed for the efficient calculation of intersections and distances. Test cases demonstrated that the algorithm worked as intended and was accurate. This work offers a strong basis for further research and has shown how useful hexagonal grid systems can be in sensor-based applications.