ECON 381 PROJECT

KÜBRA KILIÇ 20232810010

1. Axial coordinate system is the most suitable option for the coordinate system in the pointed hexagonal grid. In axial coordinates, two axes (q,r) are used; q is the horizontal and r is the diagonal axis. There are adventages to using this system. The following formula is used to find the distances between hexagons:

(abs(q1-q2) + abs(r1-r2) + abs(q1+r1-q2-r2)) / 2 = distance

Base coordinate distances are added to define neighbors.

Their intersection is determined by determining whether the cells fall within the specified ranges.

2. A HashMap/Dictionary would be most suited for storing the complete map since it can effectively hold sparse grids.

Permits O(1) lookup time to determine whether a cell is present at specified coordinates.

Additional cell information, such as if a tower is present, can be readily stored. A coordinate pair (q,r) might serve as the key, and cel information could serve as the value.

3. The store areas detemined by sensor data: Regarding circles: it would be ideal to use a hash set of coordinates.

A. Effective for evaluating membership

B. Intersection operations are simple to execute. For rings, there is no discernible difference between storing rings and circles in a hash set.

C. Both require efficient set operations to locate intersections and O(1) search for coordinate checking.

4. The Java application is:

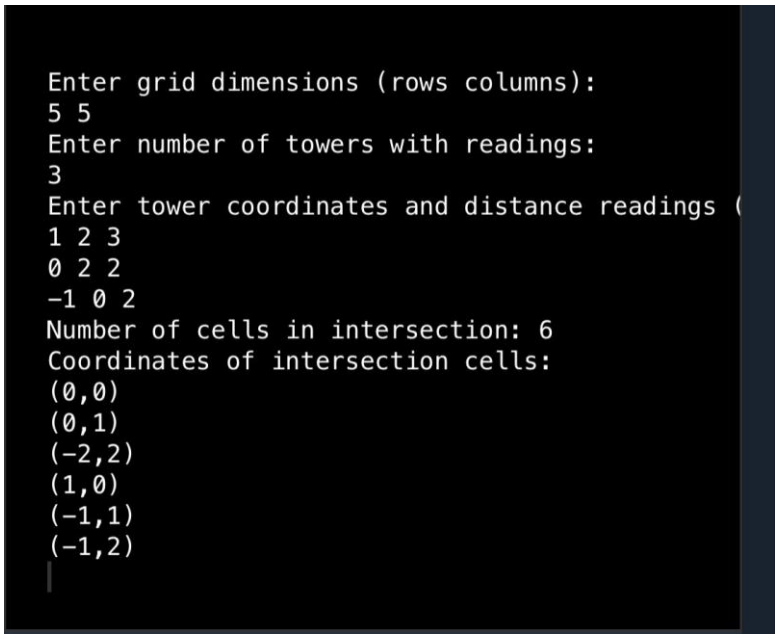
```
import java.util.*;


class Coord {
  final int q, r;


  public Coord(int q, int r) {
    this.q = q;
    this.r = r;
  }


  @Override
  public boolean equals(Object o) {
```

```
        if (this == o) return true;
        if (!(o instanceof Coord)) return false;
        Coord coord = (Coord) o;
        return q == coord.q && r == coord.r;
    }

    @Override
    public int hashCode() {
        return Objects.hash(q, r);
    }

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

public class HexGridSystem {
    private final Map<Coord, Boolean> grid = new HashMap<>();

    public int distance(Coord a, Coord b) {
        return (Math.abs(a.q - b.q) + Math.abs(a.r - b.r) + Math.abs(a.q + a.r - b.q - b.r)) / 2;
    }

    public Set<Coord> getRange(Coord center, int d) {
        Set<Coord> results = new HashSet<>();
        for (int dq = -d; dq <= d; dq++) {
            for (int dr = Math.max(-d, -dq-d); dr <= Math.min(d, -dq+d); dr++) {
                results.add(new Coord(center.q + dq, center.r + dr));
            }
        }
```

```
        return results;
    }

    public Set<Coord> getRing(Coord center, int d) {
        Set<Coord> results = new HashSet<>();
        Coord cube = new Coord(center.q, center.r);
        if (d == 0) {
            results.add(cube);
        } else {
            int q = center.q + d;
            int r = center.r;
            int[][] directions = {{-1,0}, {-1,1}, {0,1}, {1,0}, {1,-1}, {0,-1}};
            for (int dir = 0; dir < 6; dir++) {
                for (int j = 0; j < d; j++) {
                    results.add(new Coord(q, r));
                    q += directions[dir][0];
                    r += directions[dir][1];
                }
            }
        }
        return results;
    }

    public Set<Coord> findIntersection(List<Map.Entry<Coord, Integer>> readings, boolean
exactDistance) {
        if (readings.isEmpty()) return new HashSet<>();
        Set<Coord> intersection = exactDistance ? getRing(readings.get(0).getKey(),
readings.get(0).getValue()) : getRange(readings.get(0).getKey(), readings.get(0).getValue());
        for (int i = 1; i < readings.size(); i++) {
            Set<Coord> currentRegion = exactDistance ? getRing(readings.get(i).getKey(),
readings.get(i).getValue()) : getRange(readings.get(i).getKey(), readings.get(i).getValue());
            intersection.retainAll(currentRegion);
```

```
    }
    return intersection;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    HexGridSystem system = new HexGridSystem();

    System.out.println("Enter grid dimensions (rows columns):");
    int rows = scanner.nextInt();
    int cols = scanner.nextInt();

    System.out.println("Enter number of towers with readings:");
    int numTowers = scanner.nextInt();

    List<Map.Entry<Coord, Integer>> readings = new ArrayList<>();
    System.out.println("Enter tower coordinates and distance readings (q r d):");
    for (int i = 0; i < numTowers; i++) {
        int q = scanner.nextInt();
        int r = scanner.nextInt();
        int d = scanner.nextInt();
        readings.add(new AbstractMap.SimpleEntry<>(new Coord(q, r), d));
    }

    Set<Coord> intersection = system.findIntersection(readings, false);

    System.out.println("Number of cells in intersection: " + intersection.size());
    System.out.println("Coordinates of intersection cells:");
    for (Coord coord : intersection) {
        System.out.println(coord);
    }
```

```
  }
}
```



5. Program inputs:

Grid size: 8x8

Number of towers:2

Locations and detection distances of towers:

Tower 1: (2,2) distance 2

Tower 2: (4,1) distance 2

Program outcomes:

Number of cells in the intersection zone: 4

Coordinates of cells in the intersection zone:

(3,1)

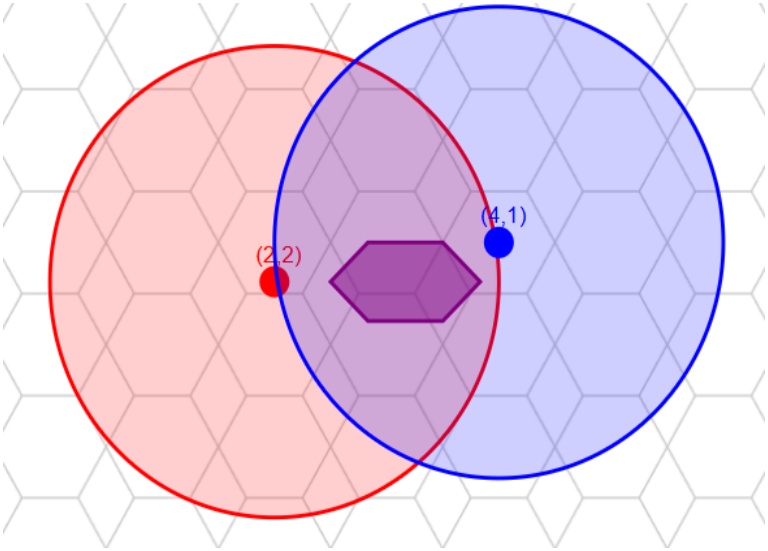
(3,2)

(4,1)

(4,2)

The program calculates the detection zone for each tower, then find the intersection of these regions. At the end, shows results.

Distance calculation: (|q1-q2| + |r1-r2| + |q1+r1-q2-r2|) / 2



Red: Tower 1

Blue: Tower 2

Purple: Intersection zone