Period Project

In this project, you are expected to do triangulation on a hexagonal grid for an application similar to the controversial ShotSpotter system. To understand how triangulation works, you can watch this short news video: https://youtu.be/USNJ2eOme8E .

Our project is working for observation towers located in a large park. These watchtowers can be used to monitor fires or other events of interest (not necessarily gunshots).

The positions of these towers are arranged so that they are in the center of the cells in a hexagonal grid system.

Figure 1: A beautiful observation tower

Figure 2: Hexagonal grid projected on the geography of the national park

The special type of sensor we will use in this project cannot detect the exact location or direction of the source, but it can estimate how many cells away the source is.

This also requires some kind of triangulation. For example, in the image below, two towers (indicated in darker colors) each reported that it detected that something was inside 3 cells (d = 1, 2, or 3). Each reported area is illustrated with a more explicit version of the tower. Therefore, the cells in which the two towers are within 3 cells are intersections (indicated in green).

If there was more reporting from two towers, we could do more intersections and triangulate the source better.

Points to be considered:

a distance within 3 cells means that the sensor actually detects something within 3 times the horizontal space. So we approximately form a circle with a set of hexagons.

An exact 3-cell distance means that the sensor detects that it is within 3 times the horizontal space, but not within 2 times the horizontal space at the same time. This time we approximately form a ring with a set of hexagons.

As a result, we can encounter one of three situations:

There is no intersection (this means a false positive),

A cell (exact triangulation), or

A list containing more than one cell (one region).

Figure 3: Intersection of two regions

Questions

What kind of coordinate system can we use to specify cells in a hexagonal grid with point ends? If there are alternatives, which one provides the easiest method for calculating distances or making intersections in the December shown above?

Which data structure is more suitable for storing the entire map?

Which data structure is more suitable for storing a region defined by sensor reading? Does it matter if the area is a circle or a ring?

Perform the following operations with Java:

Coordinate system, map and intersection finding. Your program should receive the following inputs:

* An indication of the number or size of cells on the map (e.g. rows, columns, etc.)

The number of cells that respond to radar

The coordinates of the cells that respond to the radar (repeat the specified number until it is completed)

Your program should give the following outputs:

The number of cells in the intersection,

The coordinates of the cells at the intersection.

Your report should also include a test sample:

A sketch marking the towers, zones and intersection (it may have been hand-drawn and photographed).

A screenshot showing that your program is running while receiving the inputs.

A screenshot showing that your program is running while showing the outputs.

Submit your code and report via Github.

Oct - Background on Hexagonal Grids and Distances To understand the graph, you need to know that the size of a hexagon is usually defined according to the radius of the circle it fits into (the outer circle).

And, in a point-Decked layout, the distance between the centers of two cells defines the horizontal area. In a flat-ended layout, the horizontal and vertical space definitions change places. Also, remember that the horizontal area between neighboring centers is always the same. Dec.

The figures above are taken from a really good tutorial about the use of hexagonal grids in game programming: https://www.redblobgames.com/grids/hexagons /. You are free to read this tutorial, especially the sections related to geometry, neighbors, distances, December, rings and map storage. Also, keep in mind that some concepts related to hexagonal grids are related to presentation with computer graphics, and you do not need to produce graphics in this semester's project. Therefore, you don't need to understand everything about hexagonal grids.

Figure 4: The size of a hexagon

Figure 5: Horizontal area in the dot-tipped layout

https://www.redblobgames.com/grids/hexagons/

```
// Java program for triangulation on a hexagonal grid

import java.util.*;


// Class to represent a hexagonal grid

class HexGrid {

  private int gridRows;


  private int gridCols;

  private Set<String> radarResponses;


  public HexGrid(int rows, int cols) {

    this.gridRows = rows;

    this.gridCols = cols;
    this.radarResponses = new HashSet<>();
  }

  // Add radar response for a given coordinate
  public void addRadarResponse(int q, int r) {
    radarResponses.add(encode(q, r));
  }

  // Calculate intersection of radar responses
  public Set<String> calculateIntersection(int radius) {
    Map<String, Integer> cellCounts = new HashMap<>();

    for (String coord : radarResponses) {
        int[] qr = decode(coord);
        for (int dq = -radius; dq <= radius; dq++) {
            for (int dr = Math.max(-radius, -dq - radius); dr <= Math.min(radius, -dq + radius); dr++) {
                int ds = -dq - dr;
                if (Math.abs(dq) + Math.abs(dr) + Math.abs(ds) <= radius) {
                    String neighbor = encode(qr[0] + dq, qr[1] + dr);
                    cellCounts.put(neighbor, cellCounts.getOrDefault(neighbor, 0) + 1);
                }
            }
        }
    }

    // Only keep cells that are within the range of all sensors
    Set<String> intersection = new HashSet<>();
    for (Map.Entry<String, Integer> entry : cellCounts.entrySet()) {
        if (entry.getValue() == radarResponses.size()) {
            intersection.add(entry.getKey());
        }
        }

        return intersection;
    }

    // Encode coordinates as a string for storage
    private String encode(int q, int r) {
        return q + "," + r;
    }

    // Decode a string back to coordinates
    private int[] decode(String coord) {
        String[] parts = coord.split(",");
        return new int[]{Integer.parseInt(parts[0]), Integer.parseInt(parts[1])};
    }

    public void printIntersection(Set<String> intersection) {
        System.out.println("Number of cells in intersection: " + intersection.size());
        System.out.println("Coordinates of intersection:");
        for (String cell : intersection) {
            System.out.println(cell);
        }
    }
  }

  // Main class to run the program
  public class HexagonalTriangulation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of rows and columns for the grid:");
    int rows = scanner.nextInt();
    int cols = scanner.nextInt();

    HexGrid grid = new HexGrid(rows, cols);

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

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

    System.out.println("Enter the sensing radius:");
    int radius = scanner.nextInt();

    Set<String> intersection = grid.calculateIntersection(radius);
    grid.printIntersection(intersection);
  }
```