COMP20050 - Software Engineering Project II

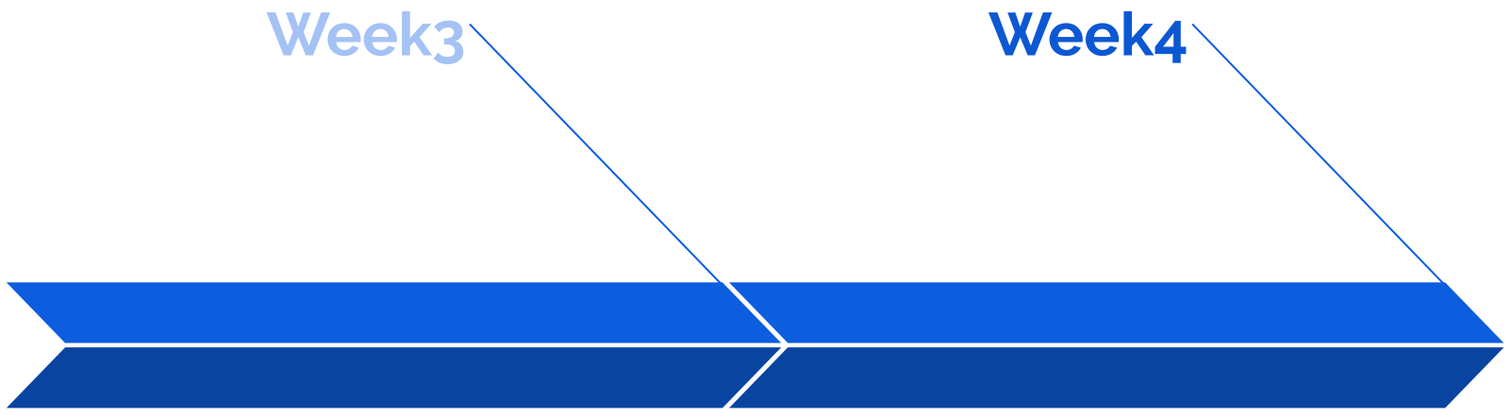
Hexagonal Grids: Design and Implementation

Ravi Reddy Manumachu
ravi.manumachu@ucd.ie

UCD School of Computer Science.

Scoil na Ríomheolaíochta
UCD.

COMP20050 - Week 3 & 4



Teamwork and Agile

Scrum Software Development Methodology

Hexagonal Grids

Introduction to JavaFX

Outline (Learning Objectives)

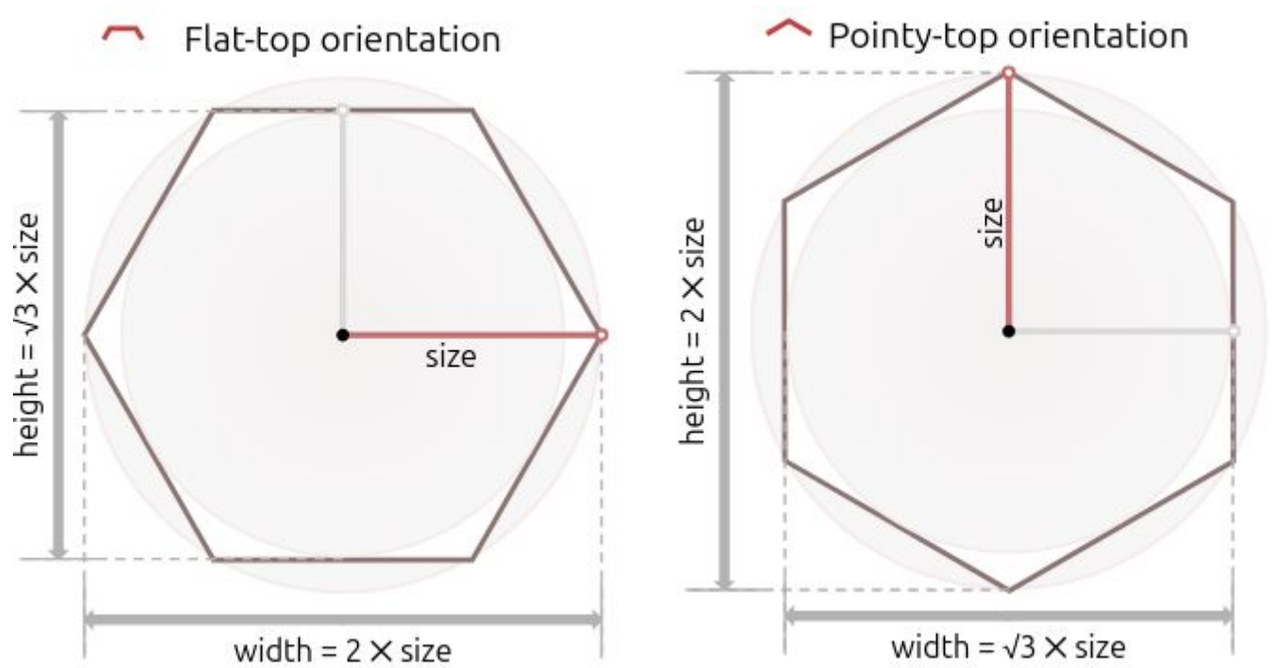
- Understand the **orientation** and **coordinate systems** used for hexagonal grids.
- Understand a **Java implementation** of hexagonal grids constructed using cubic coordinate system.

Resources

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

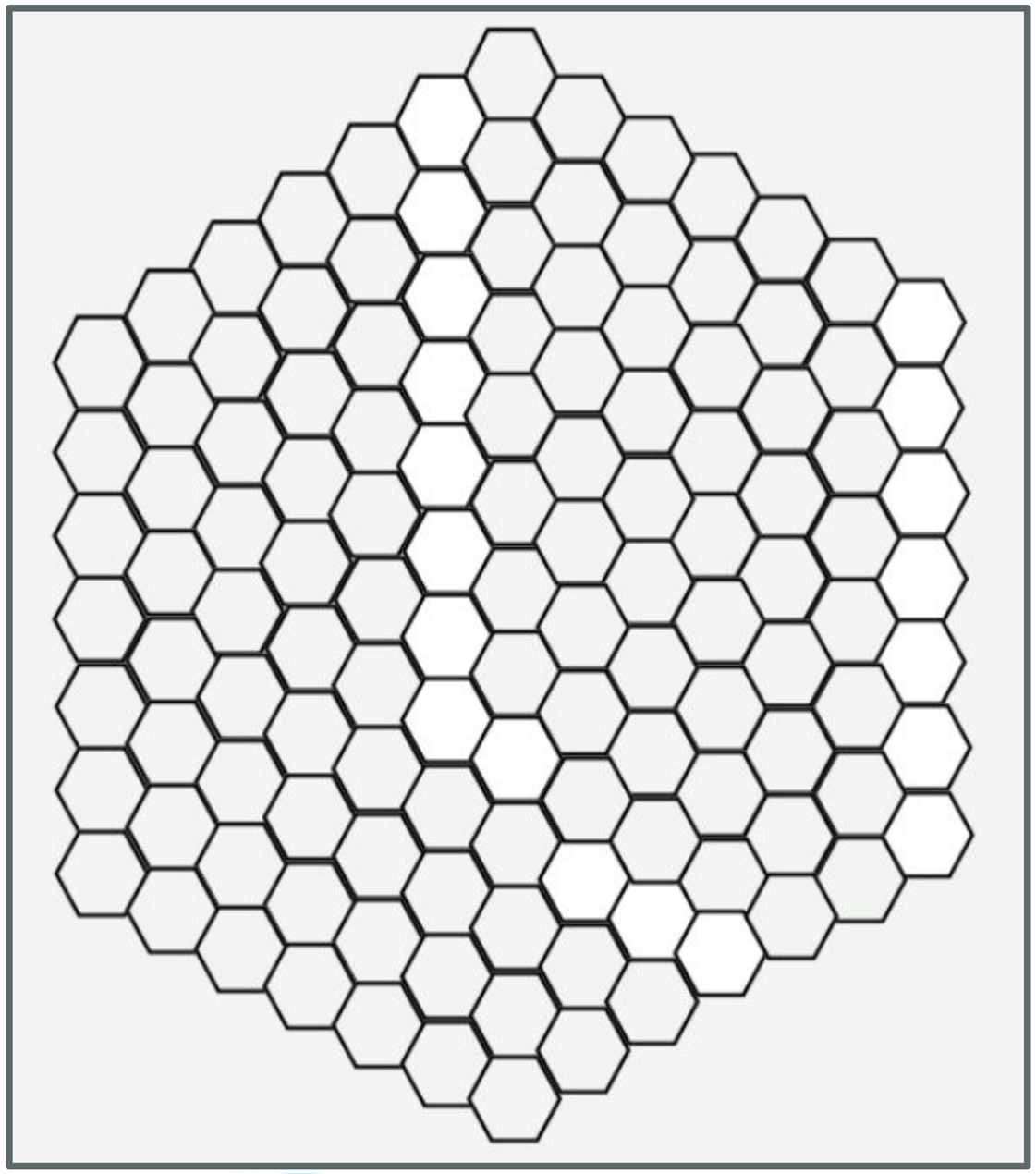
https://www.redblobgames.com/grids/hexagons/implementation.html

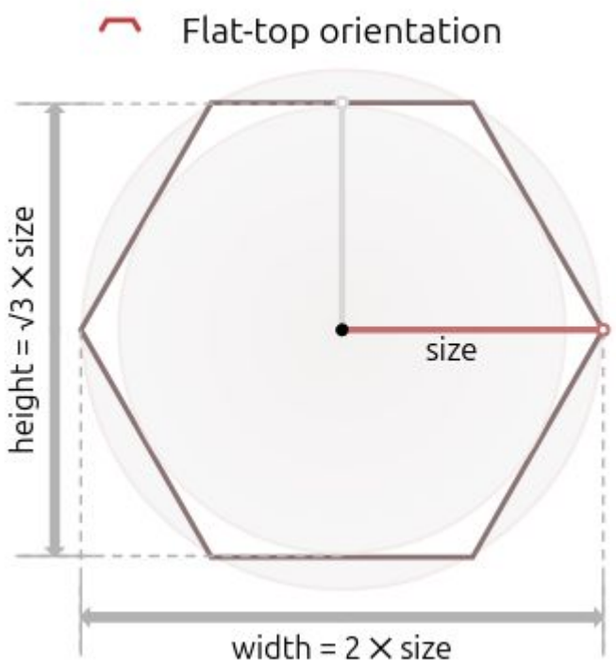
Hexagon Orientation



- **Flat top** or **Pointy top** orientations of hexagons in 2D games.

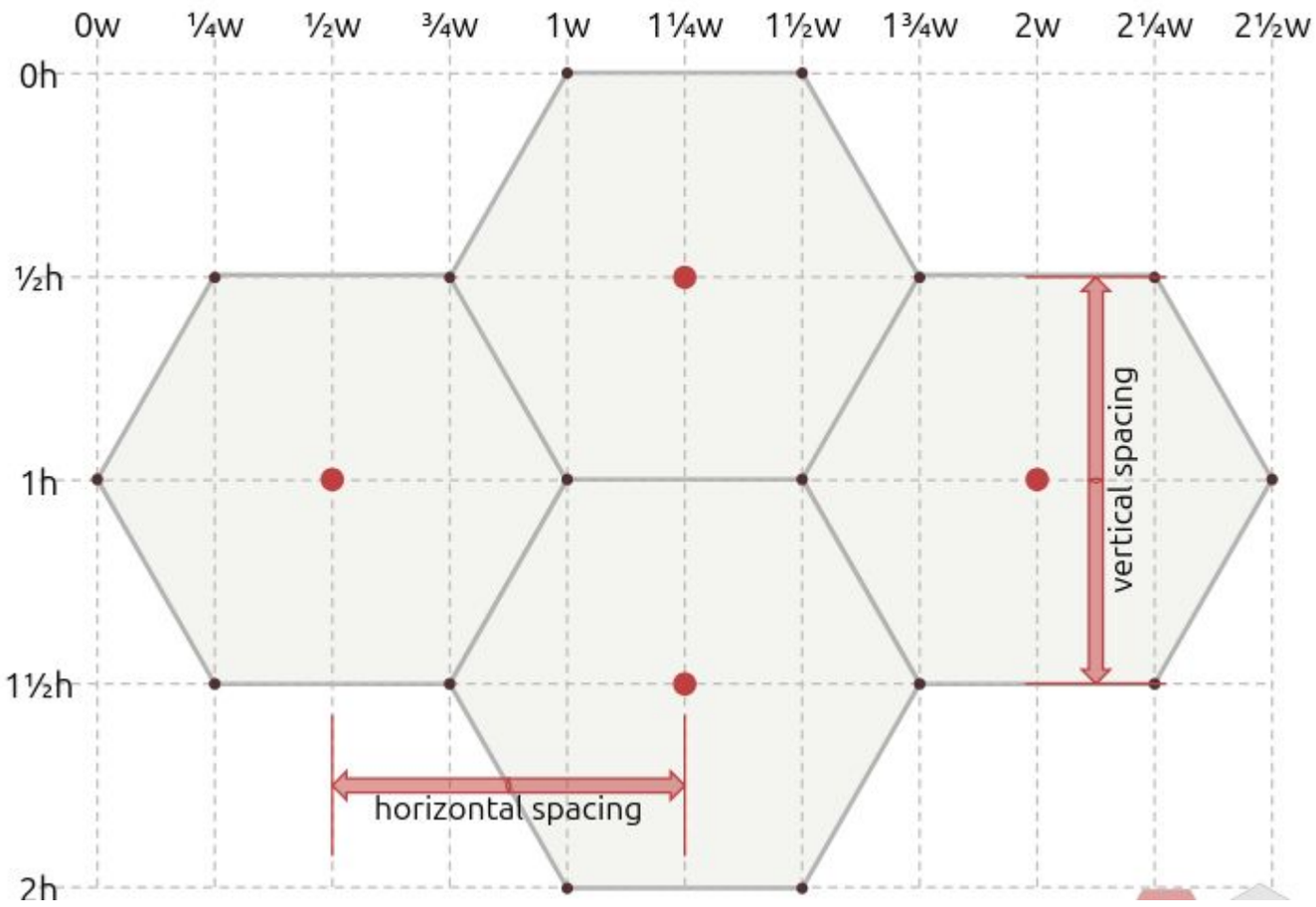
HexOust Hexagons





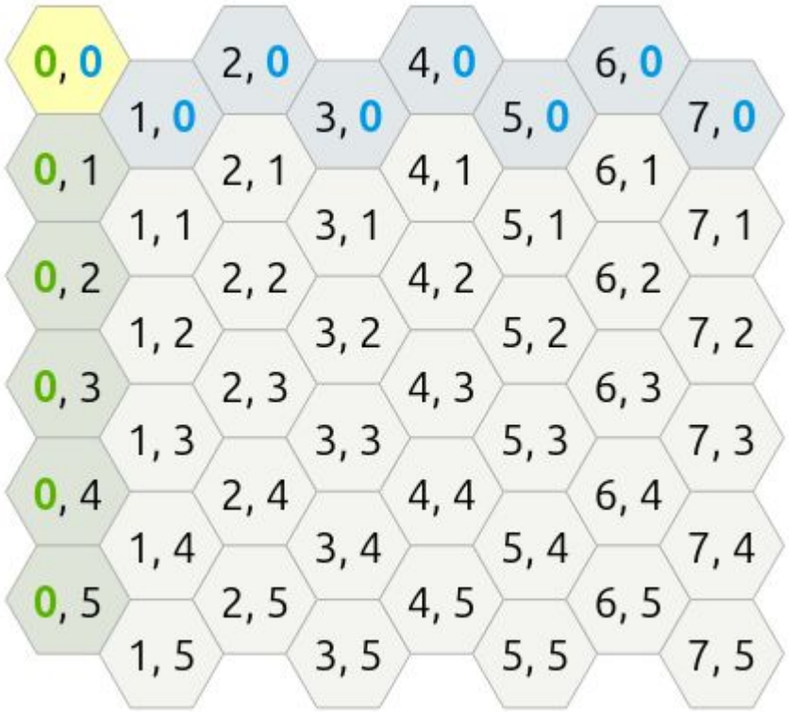
- HexOust hexagons have **flat top** orientation.
- All hexagons have the same **size**, **width**, and **height**.
- **size** = Radius of the outer circle
  width = 2 * **size**
  height = √3 * **size**

Spacing Between Flat Top Hexagons

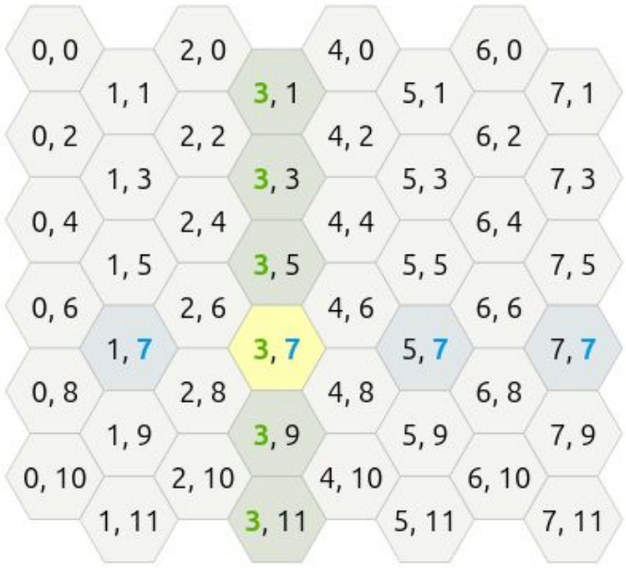


- Horizontal distance between adjacent hexagons (centers) = 3/2 * **size**.
- Vertical distance between adjacent hexagons (centers) = √3 * **size**.

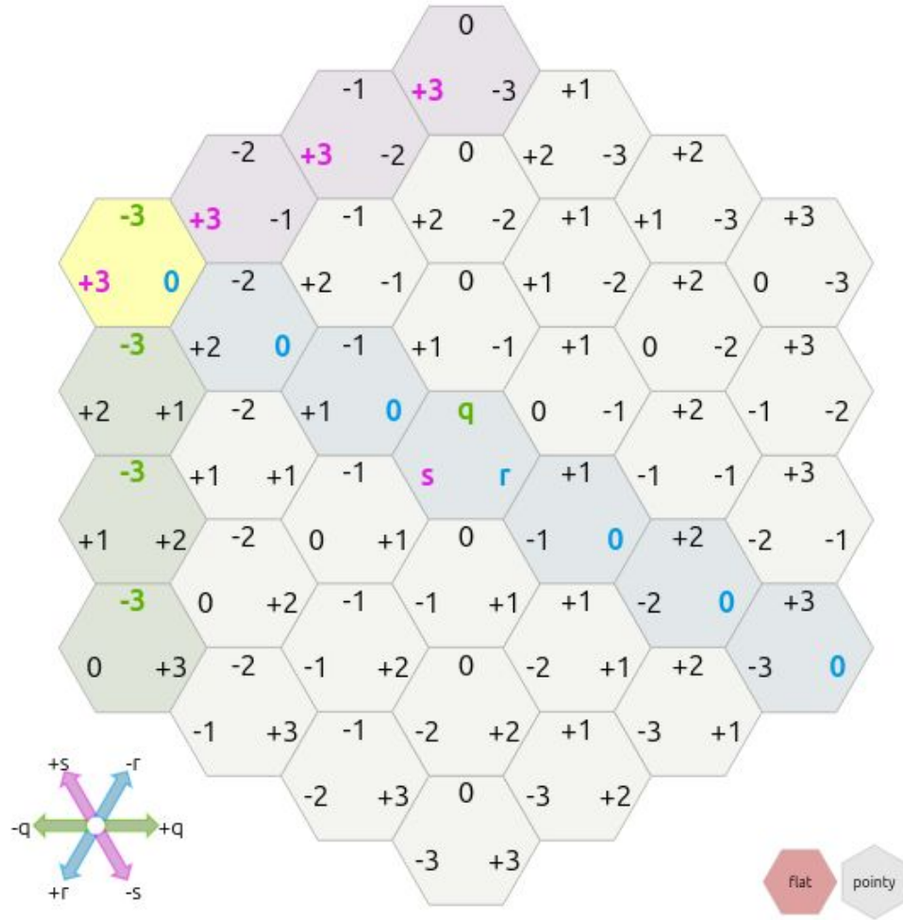
Flat Top Hexagonal Grid: Coordinate Systems



Offset



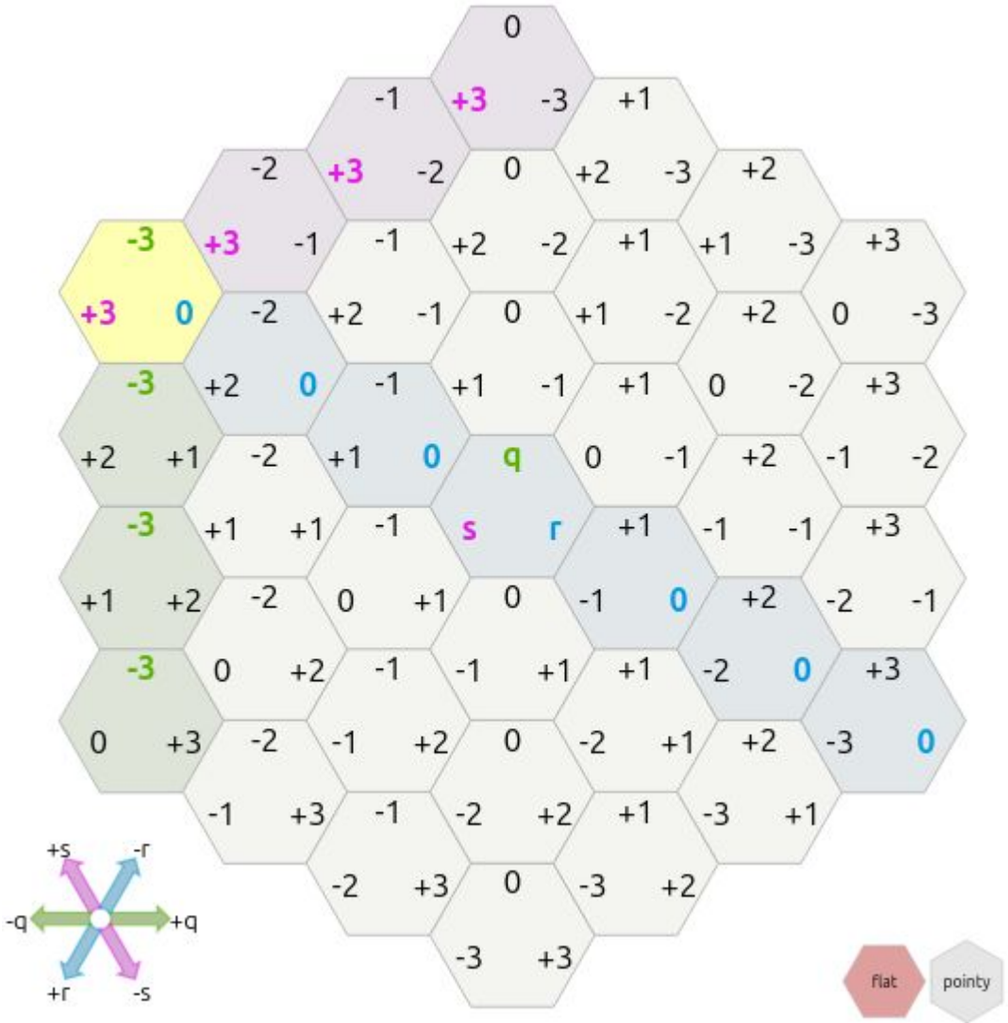
Doubled



Cube/axial

- There are several coordinate systems that you can use for a hexagonal grid with different tradeoffs for storage and complexity of algorithms.
- A good coordinate system can simplify storage and algorithms such as finding neighbours, intersecting ranges, rotation, and reflection.

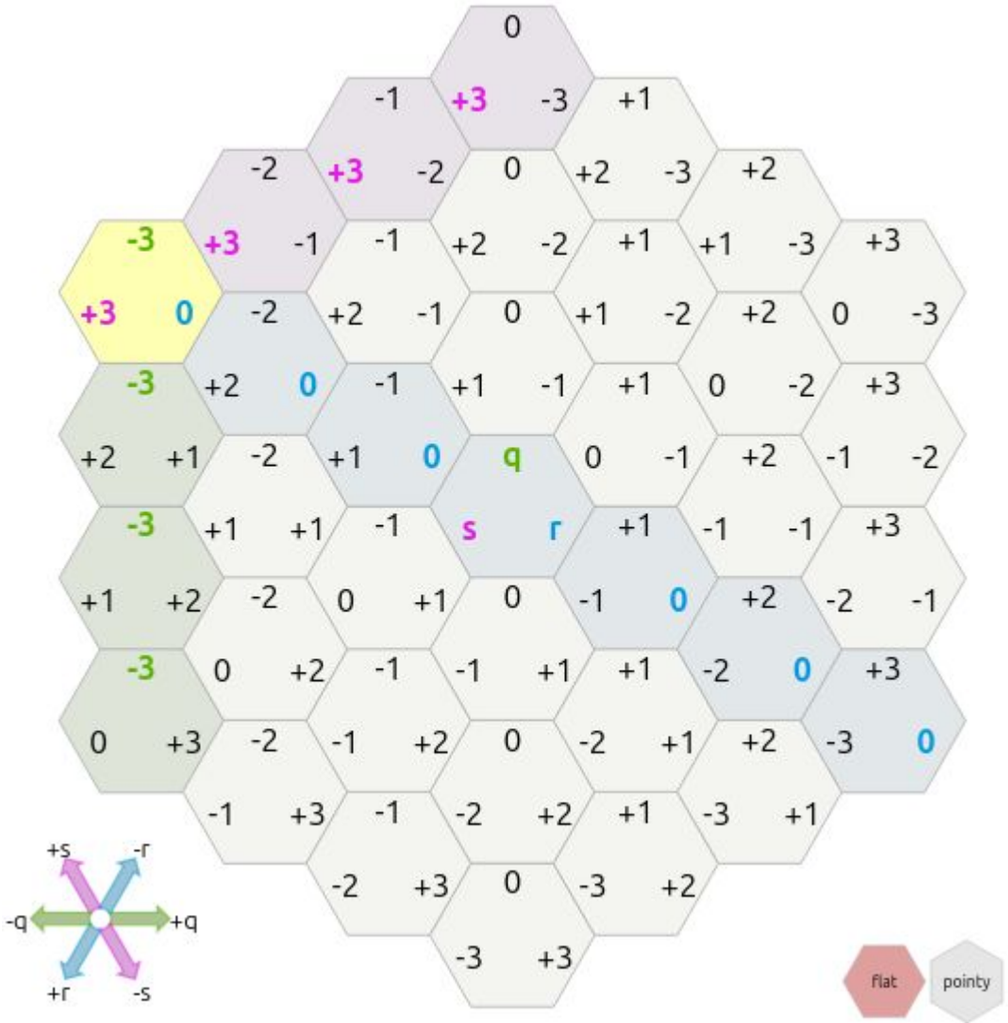
Cube Coordinate System (1/2)



Cube/axial

- I would recommend **cube/axial coordinate system**. It simplifies algorithms such as distances and finding neighbours.
- *However, it is not mandatory that you use this coordinate system in this project.*

Cube Coordinate System (2/2)



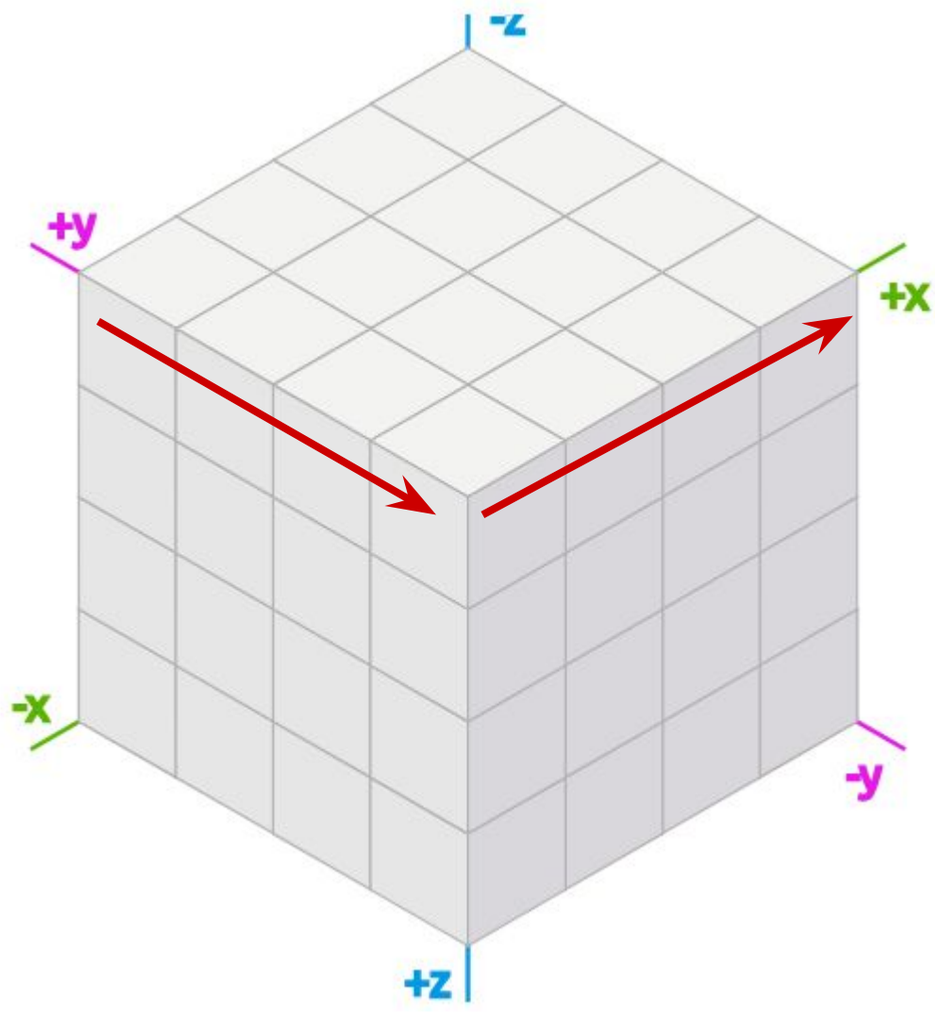
Cube/axial

- Hexagonal grids are represented by three primary axes (**q**, **r**, **s**) unlike square grids given by (**x**,**y**).

- (**q**, **r**, **s**) are three hexagonal directions 120° apart.

- The property/constraint **eliminates redundancy** and **ensures consistency**.

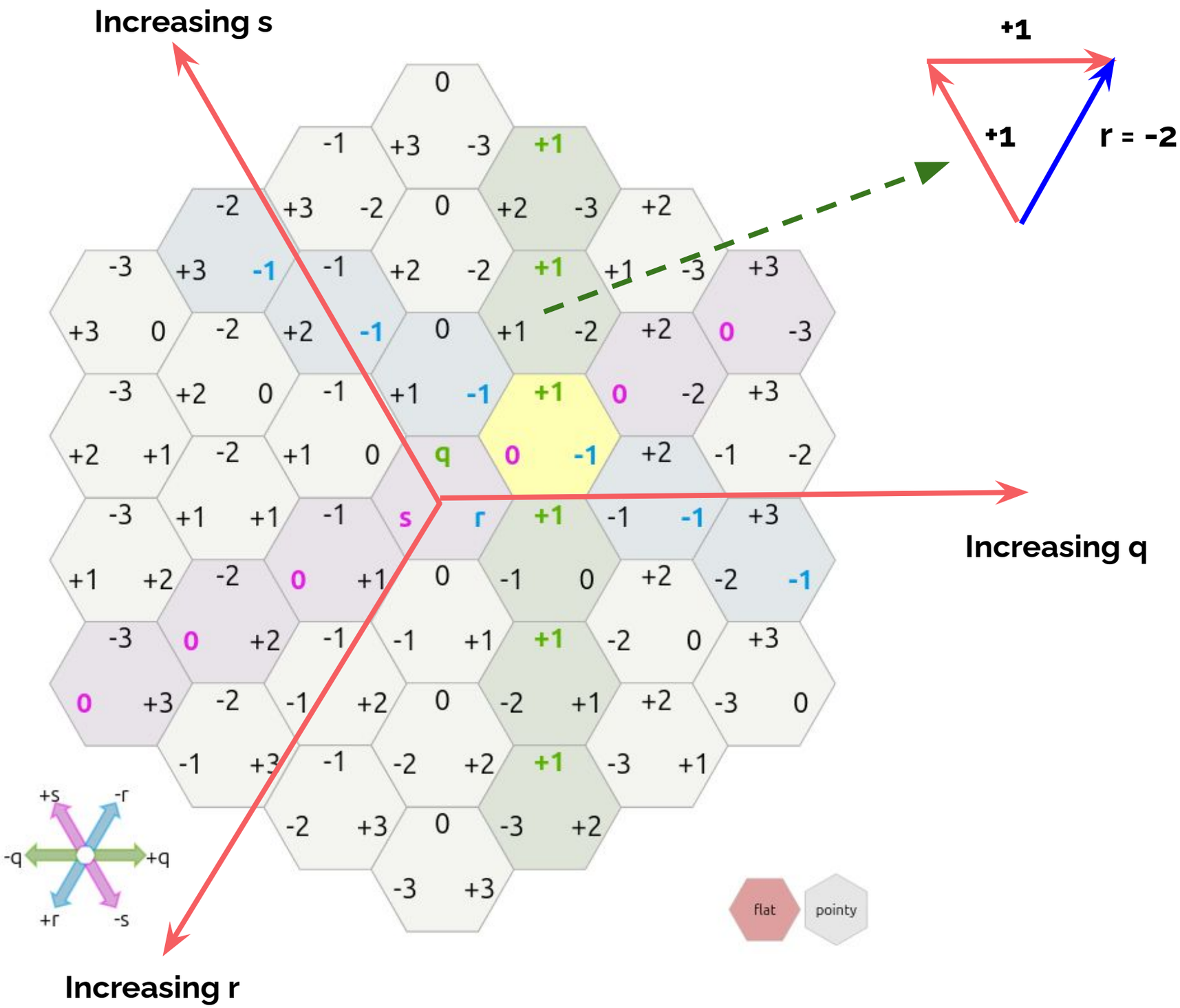
q + r + s = 0



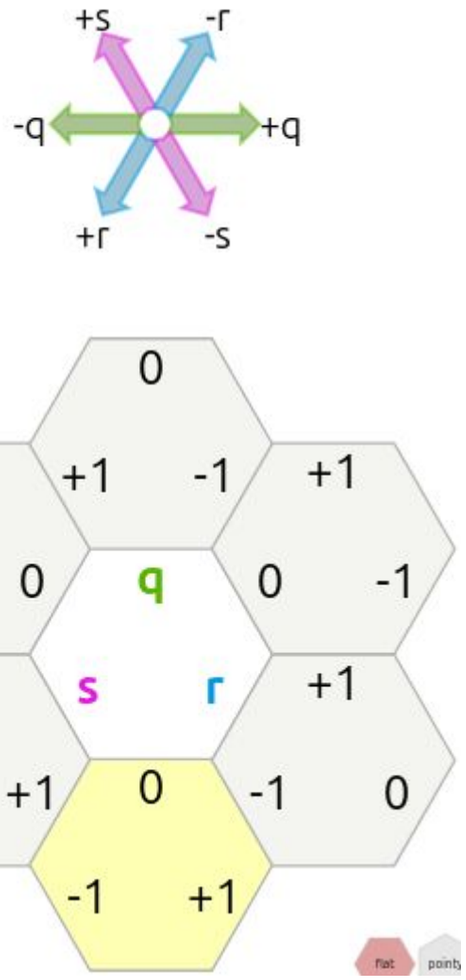
q, r, s represented by x, y, z

- In 3D, moving along the red arrows takes you from **+y** to **+x**.
- The z-coordinate will be 0 and is independent .
- When you squeeze 3D into 2D, one dimension is lost. The three axes (**q**,**r**,**s**) are not independent.

q + r + s = 0


Increasing s
+1
+1
r = -2
Increasing q
Increasing r
+s
-r
-q
+q
+r
-s
flat
pointy

Cube Coordinate System: Neighbours

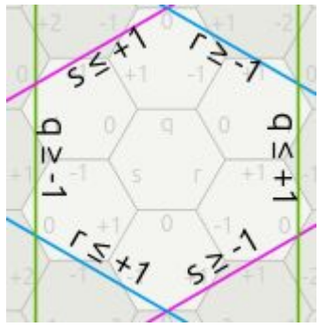


Neighbours in Cube

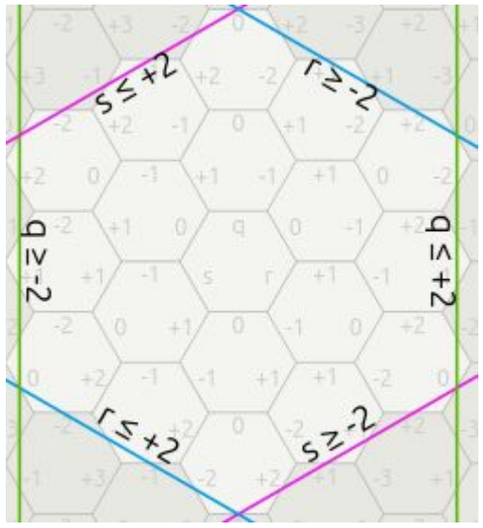
- Finding the **neighbours** of a hexagon is **easy with cube coordinates**.
  - Helpful in board games to find paths connecting hexagons.

- Neighbours:
  - (1, 0, -1)
  - (1, -1, 0)
  - (0, -1, 1)
  - (-1, 0, 1)
  - (-1, 1, 0)
  - (0, 1, -1)

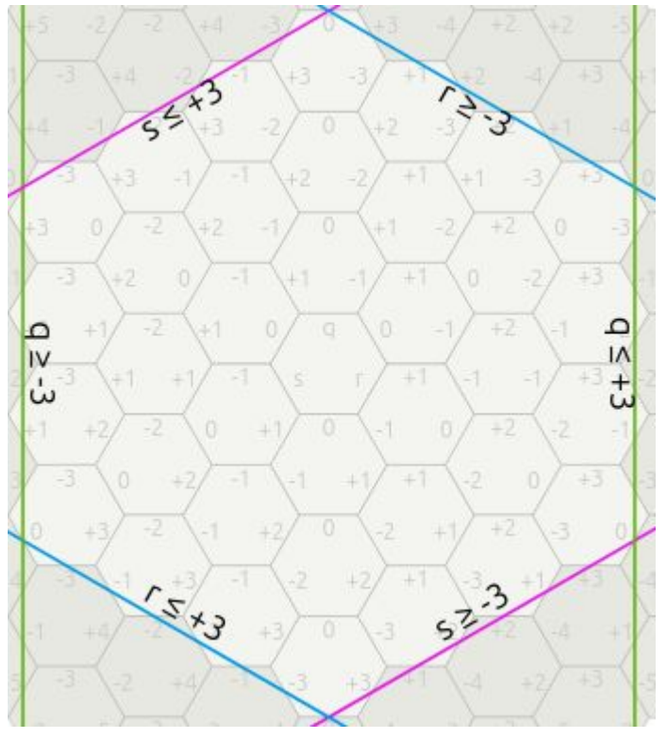
Hexagonal Grid Generation



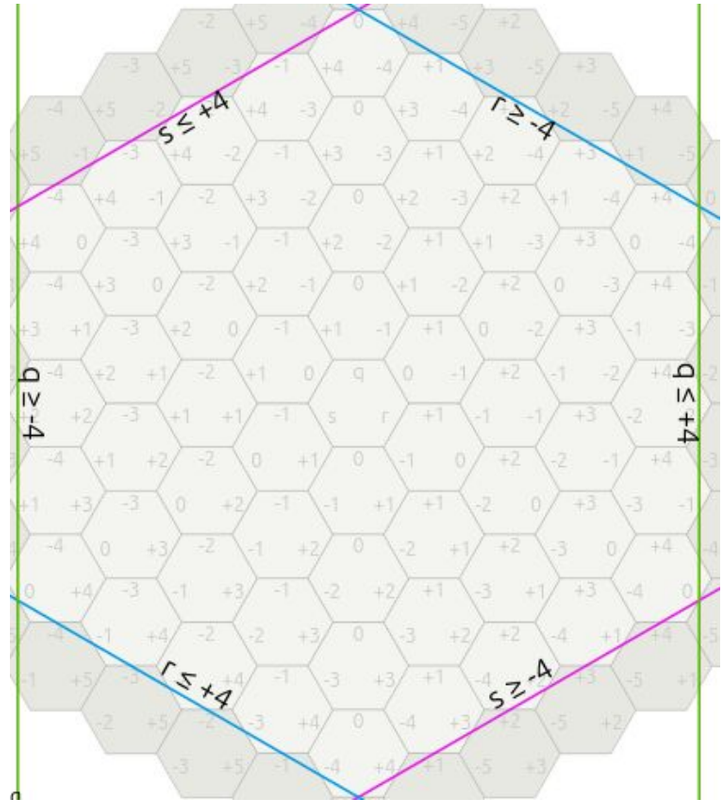
N = 2



N = 3



N = 4

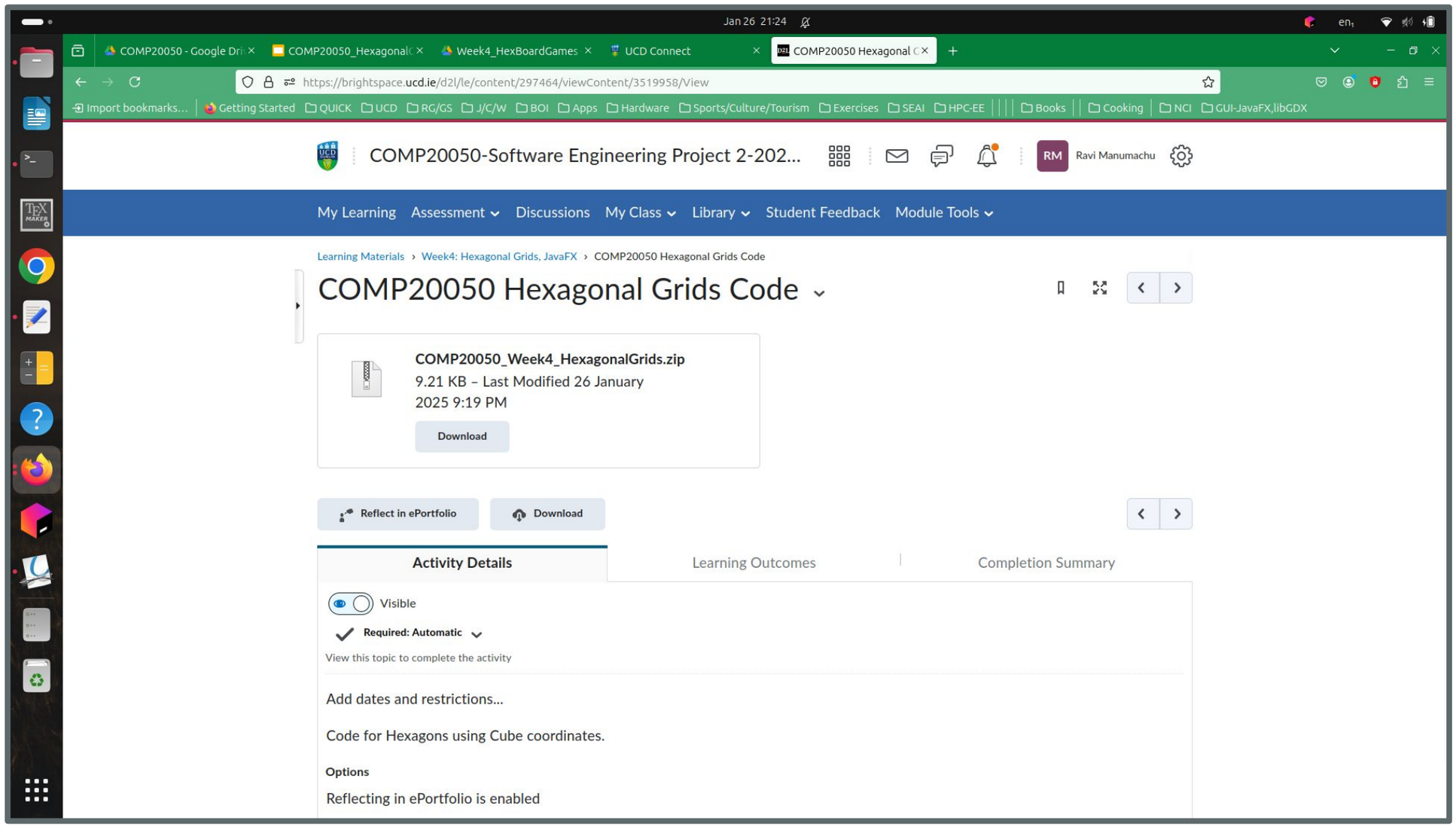


N = 5

- One can generating **base-N** (distance **N-1** from the center) hexagonal grids using a simple loop.
- Java code presented in later slides.

Hexagonal Grids: A Java Implementation

Hexagonal Grid Code



- The code presented in this lecture is on Brightspace above.

Flat Top Hexagonal Grid: Java Implementation

```
class Point
{
    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }
    public final double x;
    public final double y;
}
```

- A simple java class called **Point** for storing screen coordinates of a point in a 2D X-Y Cartesian coordinate system.

Hexagon: Java Implementation

```
class HexCube
{
    public HexCube(int q, int r, int s)
    {
        this.q = q;
        this.r = r;
        this.s = s;
        if (q + r + s != 0)
            throw new IllegalArgumentException(
                "q + r + s must be 0");
    }

    public final int q;
    public final int r;
    public final int s;
...
```

- **Cube coordinate system** is used to represent a Hexagon.
- Note the property in the constructor:

  q + **r** + **s** = 0

Hexagon Coordinate Arithmetic

```
public HexCube add(HexCube b)
{
        return new HexCube(q + b.q, r + b.r, s + b.s);
}

public HexCube subtract(HexCube b)
{
        return new HexCube(q - b.q, r - b.r, s - b.s);
    }
...
```

- Coordinate arithmetic is straightforward since Cube coordinates follow from 3D coordinates.

Distance Between Two Hexagons

```
public int length()
{
    return (int)((Math.abs(q) + Math.abs(r) + Math.abs(s)) / 2);
}

public int distance(HexCube b)
{
    return subtract(b).length();
}
```

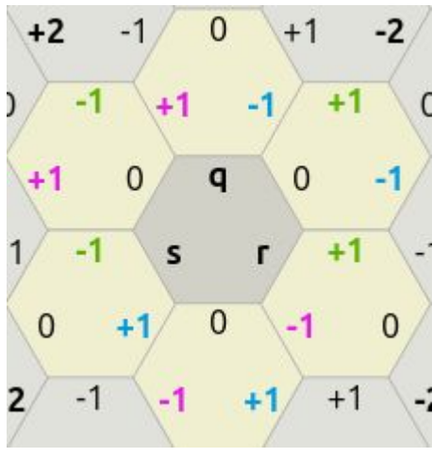
...

- **Manhattan distance** measures the sum of the absolute differences between the coordinates of the points.
- The distance between two points ($\mathbf{q_1}$,$\mathbf{r_1}$,$\mathbf{s_1}$) and ($\mathbf{q_2}$,$\mathbf{r_2}$,$\mathbf{s_2}$) on a hexagonal grid is half of the Manhattan distance.

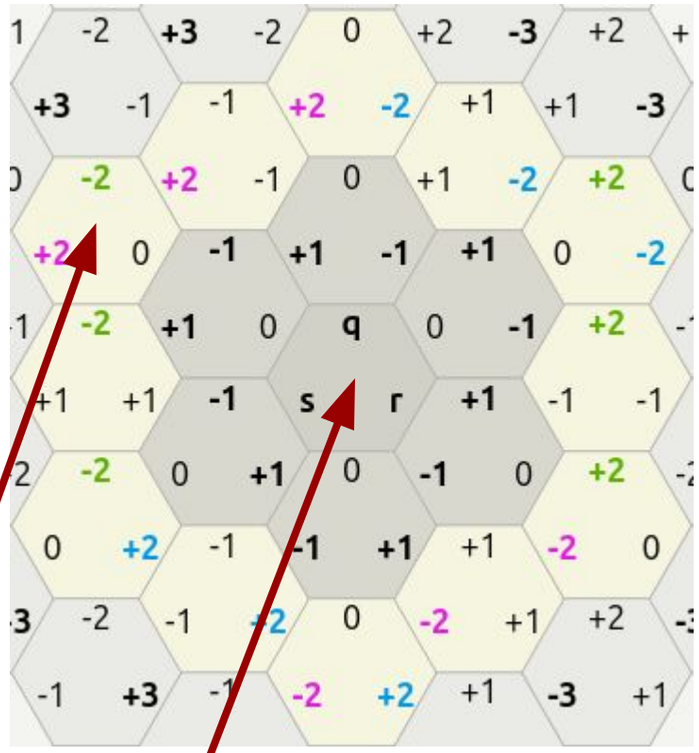
$$\text{Distance} = \frac{|q_1 - q_2| + |r_1 - r_2| + |s_1 - s_2|}{2}$$

Distance Between Two Hexagons

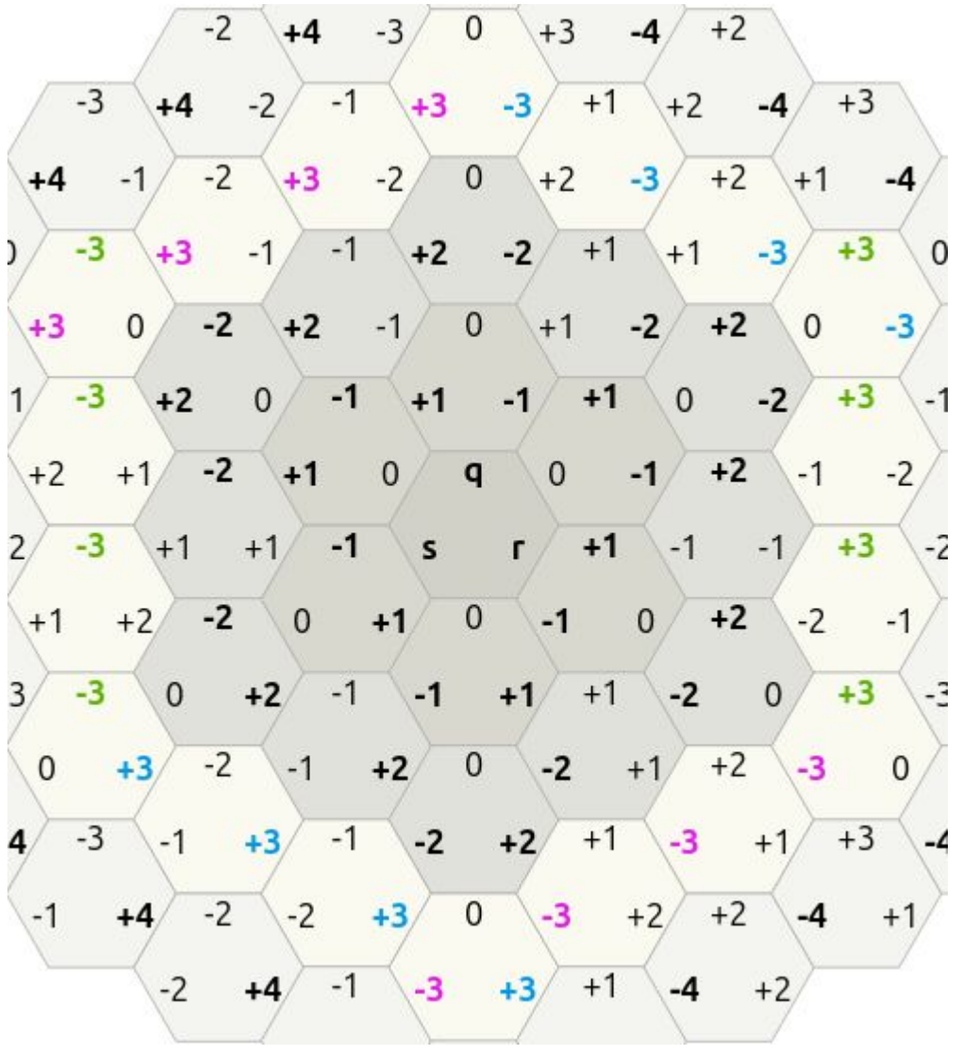
d = 1



d = 2



d = 3



A= (-2, 2, 0) O = (0, 0, 0)

$$d_{AO} = (|-2| + |2| + |0|) / 2$$
$$= 2$$

Flat Top Orientation

```
class Orientation
{
    public Orientation(double f0, double f1, double f2, double f3,
                       double b0, double b1, double b2, double b3,
                       double start_angle)
    {
...
```

- Used for the **flat top** layout.

Flat Top Layout

```
class Layout {
    public Layout(Orientation orientation, Point size, Point origin)    {
        this.orientation = orientation;
        this.size = size;
        this.origin = origin;
    }
    public final Orientation orientation;
    public final Point size;
    public final Point origin;
    static public Orientation flat = new Orientation(3.0 / 2.0, 0.0, Math.sqrt(3.0) / 2.0,
                                                     Math.sqrt(3.0), 2.0 / 3.0, 0.0, -1.0 / 3.0, Math.sqrt(3.0) / 3.0, 0.0);
...
```

- Layout is **Orientation** and two points represented by **size** and **origin**.

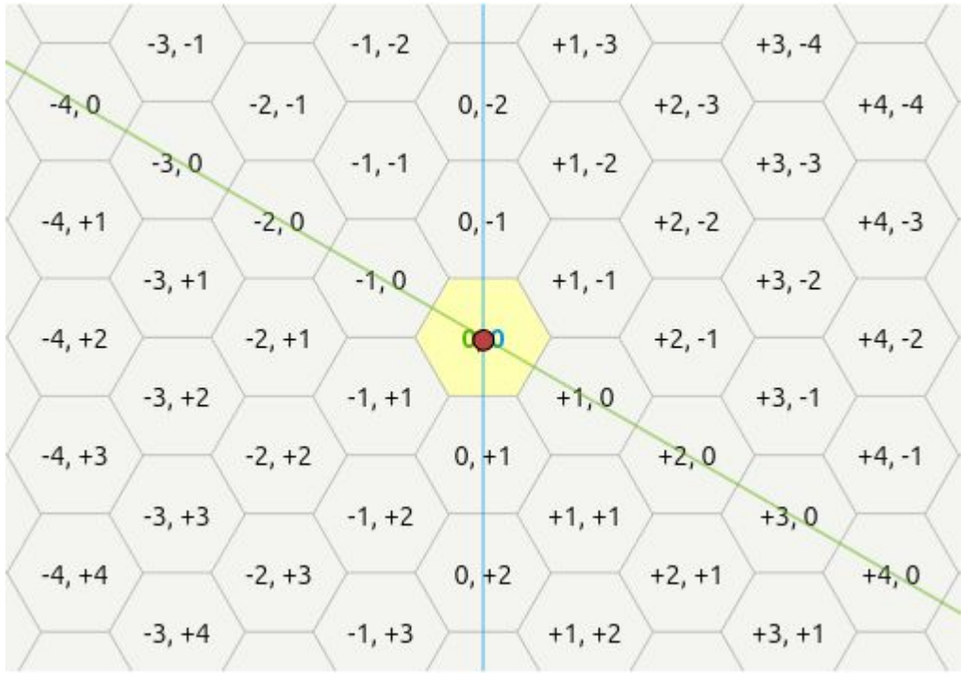
Hex to Pixel Conversion

```
public Point hexToPixel(HexCube h)
{
        Orientation M = orientation;
        double x = (M.f0 * h.q + M.f1 * h.r) * size.x;
        double y = (M.f2 * h.q + M.f3 * h.r) * size.y;
        return new Point(x + origin.x, y + origin.y);
}
```

- Converts HexCube center to a pixel location on the screen.

Pixel to Hex Conversion

```
public FractionalHexCube pixelToHex(Point p)
{
    Orientation M = orientation;
    Point pt = new Point((p.x - origin.x) / size.x,
                         (p.y - origin.y) / size.y);
    double q = M.b0 * pt.x + M.b1 * pt.y;
    double r = M.b2 * pt.x + M.b3 * pt.y;
    return new FractionalHexCube(q, r, -q - r);
}
```

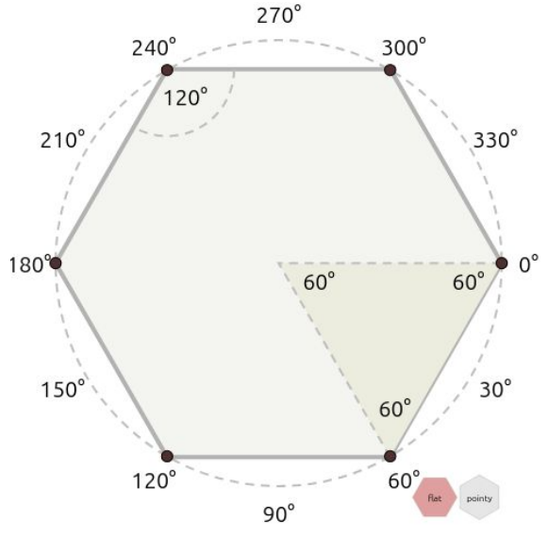


- **Allows converting a mouse click (pixel location) into a HexCube.**
- A mouse click gives you a **double** that must be converted to a HexCube center, which is an **integer**.
- **FractionalHexCube.hexRound()** will do the rounding.

Drawing a Hexagon

```
public ArrayList<Point> polygonCorners(HexCube h)
{
    ArrayList<Point> corners = new ArrayList<Point>(){{}};
    Point center = hexToPixel(h);
    for (int i = 0; i < 6; i++) {
        Point offset = hexCornerOffset(i);
        corners.add(new Point(center.x + offset.x, center.y + offset.y));
    }
    return corners;
}
```

- The loop calculates the 6 corner points for each hexagon relative to the center.
- With the flat top orientation, the corners are **0°**, **60°**, **120°**, **180°**, **240°**, **300°**.



Generating a Hexagonal Grid

```
int baseN = 6; /* 6 here is the distance of neighbouring hexagons from the center */
ArrayList<ArrayList<Point>> grid = new ArrayList<>();
for (int q = -baseN; q <= baseN; q++) {
      for (int r = -baseN; r <= baseN; r++) {
            for (int s = -baseN; s <= baseN; s++) {
                  if ((q + r + s) == 0) {
                     HexCube h = new HexCube(q, r, s);
                     ArrayList<Point> corners = flat.polygonCorners(h);
                     grid.add(corners);
                  }
            }
      }
}
```

- This loop creates a base-7 grid of hexagons. The grid is represented as a list of list of points (**ArrayList<ArrayList<Point>>**).
- Each hexagon is a ArrayList of six corner points, **ArrayList<Point>**.

Using Java AWT Graphics (Drawing Points)

```
@Override
 protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.BLUE);
        for (ArrayList<Point> hexagon : grid) {
              for (Point p : hexagon) {
                    int x = (int) Math.round(p.x);
                    int y = (int) Math.round(p.y);
                    g.fillOval(x - 5, y - 5, 10, 10);
              }
        }
...
```

- **fillOval()** method blows up the points for better visibility.

Drawing Lines Between Points

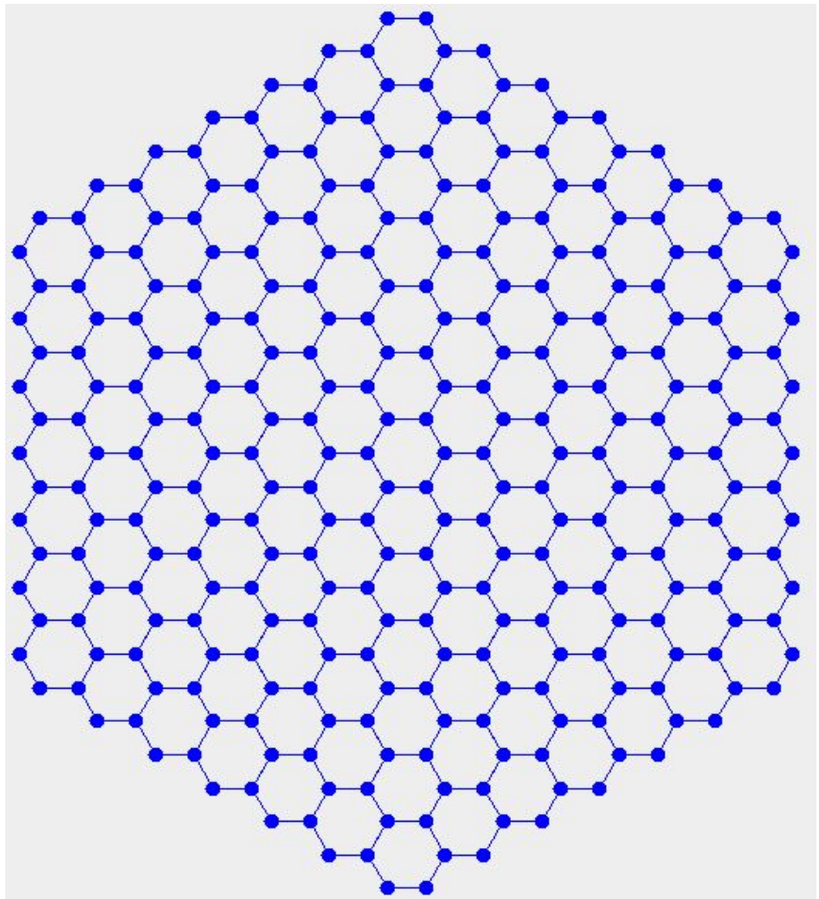
```
for (ArrayList<Point> hexagon : grid) {
        int p1xi, p1yi, p2xi, p2yi;
        Point p1 = hexagon.get(0), p2;
        for (int i = 1; i < hexagon.size(); i++)
        {
                p2 = hexagon.get(i);
                p1xi = (int) Math.round(p1.x);
                p1yi = (int) Math.round(p1.y);
                p2xi = (int) Math.round(p2.x);
                p2yi = (int) Math.round(p2.y);
                g.drawLine(p1xi, p1yi, p2xi, p2yi);
                p1 = p2;
        }

        p2 = hexagon.get(0);
        p1xi = (int) Math.round(p1.x);
        p1yi = (int) Math.round(p1.y);
        p2xi = (int) Math.round(p2.x);
        p2yi = (int) Math.round(p2.y);
        g.drawLine(p1xi, p1yi, p2xi, p2yi);
}
```

- **drawLine()** method draws a line between two points.

Running HexGrid Implementation

- The program accepts **size** (in pixels)and hexagonal grid center (**x**,**y**) from top left corner as inputs.

$ java HexGrid 25 400 400



Q&A

To follow...

Agile Software Development
Scrum Software Engineering
Framework