Black Box: Sprint 1
Implementation and testing

Since establishing the initial goals of our project and devising a plan structured around sprints, it's evident that we have deviated from the intitial sprint outlined to concentrate on enhancing the display and expanding our understanding of swing and GUI packages within the limited timeframe.

With that said, great progress has been made in implementing the board and atoms. This includes displaying the hexagons, storing their location, adding/removing atoms and calculating their subsequent circle of influcences (neighbours).

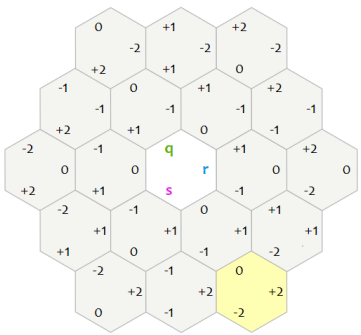
With doing so we created some unit tests to help us move on knowing everything is working fine.

Drawing of Board breakdown:

-Create inner hexagons:
This was done by utilising the properties of the unit circle, essentially mapping the vertices of the hexagon on to the circumference of a circle (with radius HEX_SIZE) that encloses the hexagon. Connecting the vertices gives the finished product.

```
private Path2D createHexagon(int x, int y) {
    Path2D path = new Path2D.Double();
    double angleStep = Math.PI / 3;//60 degrees
    double startAngle = Math.PI / 6;//30 degrees ~ pointy top
    for (int i = 0; i < 6; i++) {
        double angle = startAngle + (i * angleStep);
        double x1 = x + HEX_SIZE * Math.cos(angle);
        double y1 = y + HEX_SIZE * Math.sin(angle);
        if (i == 0) path.moveTo(x1, y1);//start point
        else path.lineTo(x1, y1);//draw to next vertex
    }
    path.closePath();
    return path;
}
```

-Align hexagons in desired pattern/location:
We got the idea for this on a website (https://www.redblobgames.com/grids/hexagons/) which outlines the usage of axial coordinates q,r,s:



s is redundant as it can be created using q and r, due to this we left it out leaving us with q and r.

Black Box: Sprint 1
Implementation and testing

```
//Draw the hexagons
for (int q = -radius; q <= radius; q++) {
    int qminus = -q - radius;
    int qplus = -q + radius;
    int r1 = Math.max(-radius, qminus);
    int r2 = Math.min(radius, qplus);
    for (int r = r1; r <= r2; r++) {

        Point point = axialToPixel(q,r);
        Path2D hexagon = createHexagon(point.x, point.y);
        g2d.draw(hexagon);

        String coordText = q + "," + r;
        int textWidth = metrics.stringWidth(coordText);
        int textHeight = metrics.getHeight();
        g2d.drawString(coordText,  x: point.x - textWidth / 2,  y: point.y + textHeight / 4);
    }
}
```

We created this code which iterates through all of the hexagon coordinates, using a helper method it converts these “axial” coordinates to pixel needed for the createHexagon() method to know where to draw each internal hexagon.
Some code was also added to show axial coordinates within each hexagon to aid us in implementation.

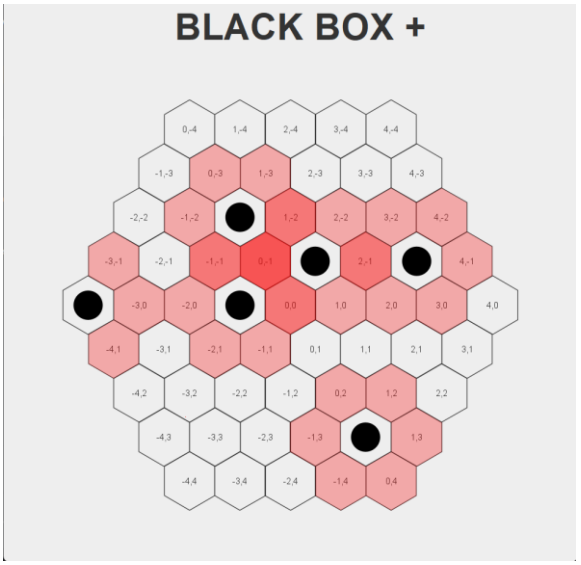
-Add and store atoms:
A mouse listener is used in the constructor to retrieve “clicked points”, these points are then converted to axial using the pixelToAxial(). Knowing now within which hexagon it was clicked it can determine if its withing the grid and if there is already an atom there or not.

```
addMouseListener(new MouseAdapter() {
    ± artemkuc44
    @Override
    public void mouseClicked(MouseEvent e) {
        super.mouseClicked(e);
        Point clickedPoint = e.getPoint();//gets pixel coord

        Point hexCoord = pixelToAxial(clickedPoint.x, clickedPoint.y);//convert to axial
        //System.out.println(hexCoord);
        if (containsElement(hexCoordinates,hexCoord)) {//if click within board
            Atom existingAtom = findAtomByAxial(hexCoord);//try find atom in atoms arrayList
            if (existingAtom != null) {
                // Atom exists, so remove it
                atoms.remove(existingAtom);
            }
            else if(atoms.size() != MAX_ATOMS) {
                // Atom doesn't exist, create and add to arraylist;
                Atom newAtom = new Atom(hexCoord);
                atoms.add(newAtom);
                updateNeighbors();
            }

            repaint(); // Repaint after every mouse click
        }
    }
});
```

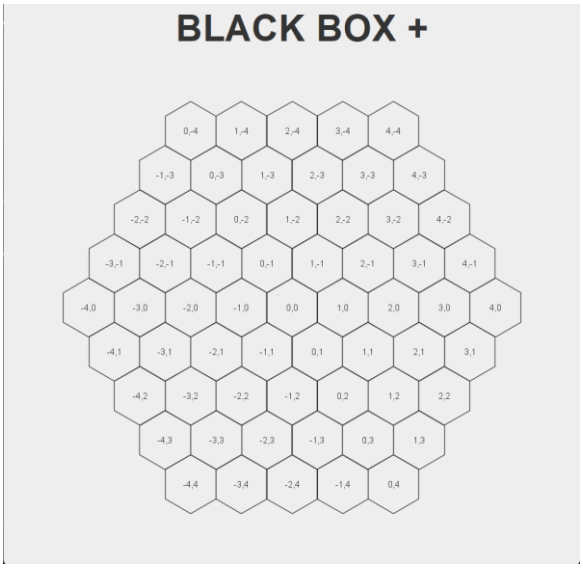
The final outcome can be seen below.

Black Box: Sprint 1
Implementation and testing

Atoms placed with COI



Mapped hex grid



Helper Functions:

-axialToPixel();

This method is really the very core of the whole operation, used to position each hexagon correctly about its centre. It precisely converts an axial coordinate to pixel.

```
private Point axialToPixel(int q,int r){
    int centerX = this.getWidth() / 2;
    int centerY = this.getHeight() / 2;

    int x = centerX + (int) (HEX_SIZE * Math.sqrt(3) * (q + r / 2.0));
    int y = centerY + (int) ((HEX_SIZE * 3 * r)/2);

    return new Point(x,y);

}
```

-pixelToAxial();

This method essentially reverses the latter (axialToPixel();), rounding to the closest hexagonal centre;

Black Box: Sprint 1
Implementation and testing

```
Point pixelToAxial(int x, int y) {
    double q;
    double r;

    x -= getWidth() / 2;
    y -= getHeight() / 2;

    q = (Math.sqrt(3) / 3 * x - 1.0 / 3 * y) / HEX_SIZE;
    r = (2.0 / 3 * y) / HEX_SIZE;

    //Round the coordinates to the nearest whole number to snap to the closest hex
    return new Point((int) Math.round(q), (int) Math.round(r));
}
```

-updateNeighbours();

This helper functions finds the circle of influence of hexagons(neighbours). This is done by iterating through the arrayList of atoms, calculating and adding the neighbours to an array within the atom class.

```
private static final Point[] DIRECTIONS = new Point[] {//Directions array used to compute circular dependency
        new Point( x: 0,  y: 1), new Point( x: 0,  y: -1), new Point( x: -1,  y: 0),
        new Point( x: 1,  y: 0), new Point( x: -1,  y: 1), new Point( x: 1,  y: -1)
};
```

```
public void updateNeighbors() {
    // Clear existing neighbors
    for (Atom atom : atoms) {
        atom.getNeighbors().clear();
    }
    // Recalculate neighbors
    for (Atom atom : atoms) {
        for (Point dir : DIRECTIONS) {
            Point neighborPoint = new Point( x: atom.getPosition().x + dir.x,  y: atom.getPosition().y + dir.y);
            if(containsElement(hexCoordinates,neighborPoint)){
                atom.addNeighbor(neighborPoint);
            }
        }
    }
}
```

For more information please see git repo link:

Git repository: https://github.com/jamie6084/SoftwareProject.git

Black Box: Sprint 1
Implementation and testing

Unit Testing

For the testing there were numerous areas we wanted to make sure worked.

1) Aim: Clicking inside the board to make sure something happened. (adding an atom).

   Procedure: Creates MouseEvent to simulate mouse click at certain points (450,450 for example) and calls mouseClicked. Then checks whether an atom was added to hex coordinate in relation to the pixel coordinates (450,450) by using the findAtomByAxial method.

2) Aim: Making sure the program knows whether a point is on the playing area or not.

   Procedure: Asserts true for points that are in the grid and false for points outside. Ensures the boundary logic in the "containsElement" method works as expected, differentiating between the points outside and inside the grid.

3) Aim: Checking to see if the game can locate a piece at a specific spot on the board.

   Procedure: Adds atom at a testing point and checks if "findAtomByAxial" can correctly locate it. It then checks that a non existing point returns null, making sure that the method can identify both present and absent Atoms.

4) Aim: Check to see if the game can figure out which spots are next to a piece as this is important in the circle of influence.

   Procedure: Adds atom and calls the "updateNeighbours", it then makes sure that the atoms neighbours match the expected points from the surrounding grid logic.

5) Aim: Make sure that when you click somewhere on the board, it correctly identifies the axial coordinate you are clicking on.

   Procedure: It tests multiple pixel coordinates to check if they are converted perfectly to their matching hex grid coordinates, this makes sure that the method "pixelToAxial" method works correctly.

Black Box: Sprint 1
Implementation and testing

What Next?

We are attempting to implement a hover function so that if the user is choosing a place to place atoms they can see its COI before placing and see overlapping COI creating an even greater angle of reflection.

Due to the lack of focus on the logic of the game and more on the dynamic display, we added unit testing that will test for edge cases and if a point is placed outside the grid or if the coi is outside the grid.

In the future, we hope to update this based on the implementation of rays we will have arrows that will show the direction of the ray-based on what side of the hex it will be shot from. This would be made easier with the hover already mentioned.

We have a basic layout of the game we want to create. The game will have one-player or two-player options. Two players will allow the user to place his atoms and have the other player find them, while one player will have the CPU randomly place the atoms in the grid using the mapped coordinates. The CPU will then play as the second player shooting rays at random giving the illusion of it guessing when in reality all it will be doing is acting as a second guess for the player suggesting where to place his ray next.