BSP tree

Our original idea regarding the implementation for the BSP code used a Node based approach, with each Node containing information on a **Bounding Box.** This bounding box kept track of partitionable space using limits. Before any tree is constructed the limits are infinite. Below is a basic implementation of this bounding box idea:

```
1  public class B_Box {
2
3      private double xmin;
4      private double xmax;
5      private double ymin;
6      private double ymax;
7
8      public B_Box() {
9          xmin = (-1)*Double.MAX_VALUE;
10         xmax = Double.MAX_VALUE;
11         ymin = (-1)*Double.MAX_VALUE;
12         ymax = Double.MAX_VALUE;
13     }
14
15     public B_Box(double xmin, double xmax, double ymin, double ymax) {
16         this.xmin = xmin;
17         this.xmax = xmax;
18         this.ymin = ymin;
19         this.xmax = xmax;
20     }
21
22 //  public void B_Box_Bounds(double ) {
23 //
24 //  }
25
26
27
28     public void set_xmin(double min) {
29         xmin = min;
30     }
31
32     public void set_xmax(double min) {
33         xmax = min;
34     }
35
36     public void set_ymin(double min) {
37         ymin = min;
38     }
39
40     public void set_ymax(double min) {
41         xmin = min;
42     }
43
44     public String toString() {
45         String output = "";
46         output = "Min x-limit: " + xmin + " Max x-limit: " + xmax + " Min y-limit: " + ymin + " M
47         return output;
48     }
49 }
50
```

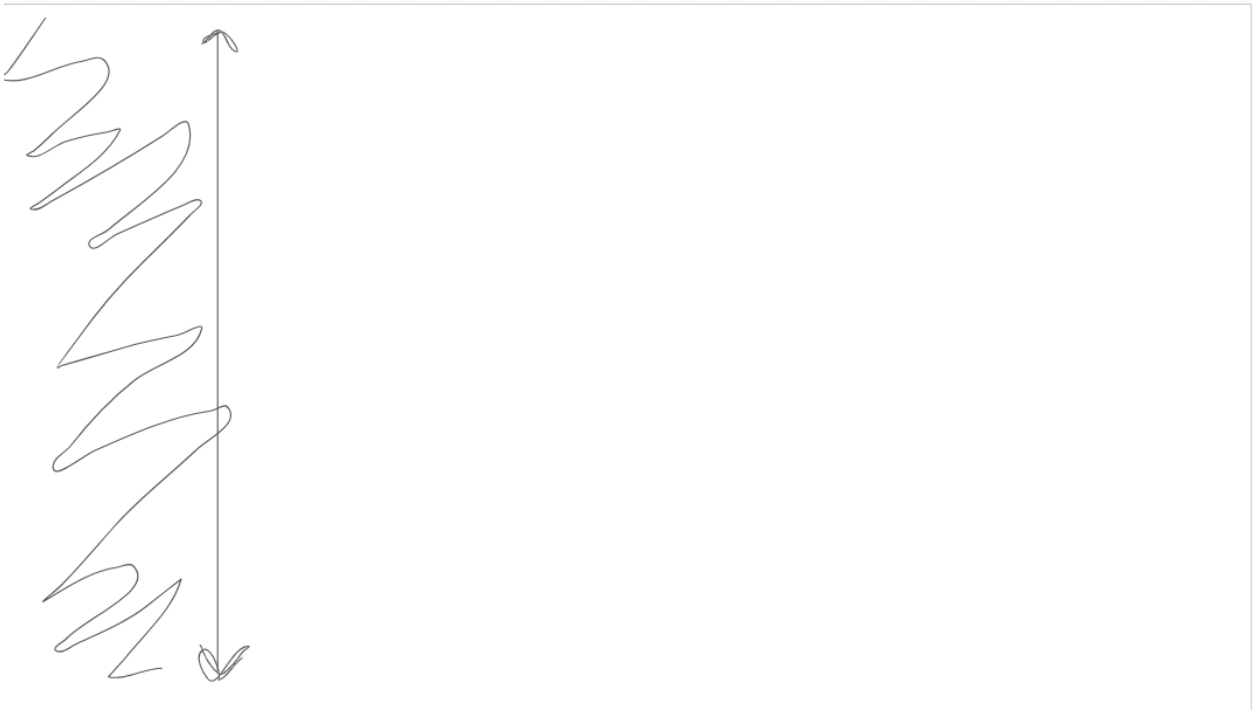
Then we also wrote a class for the BSP tree itself, using a node based approach. Below is that basic implementation:

```
1
2  public class BSP {
3
4      private Node root;
5      private int depth;
6
7
8      private class Node{
9          private int x;
10         private int y;
11         private boolean direction;
12         private boolean partition_direction;
13         private B_Box remaining_space;
14
15         private Node left,right;
16
17         public Node(int x, int y, boolean direction, boolean partition_direction) {
18
19             this.x = x;
20             this.y = y;
21             this.direction = direction;
22             this.partition_direction = partition_direction;
23
24         }
25
26         public double get_x() {
27             return x;
28         }
29
30         public double get_y() {
31             return y;
32         }
33
34         public String toString() {
35             return "Node on level" + depth + "Attributes: " + x + " , " + y;
36         }
37     }
38
39     public BSP(){
40         root = null;
41         depth = 0;
42     }
43
```

Each node contains a link to the B_Box class, along with integers for the coordinates of the partition point, a direction boolean for a vertical or horizontal partition, a partition direction boolean variable which specifies which space in reference to the partition is still partitionable, and 2 Nodes for the left and right children. Below will be a basic walkthrough of how this implementation work:

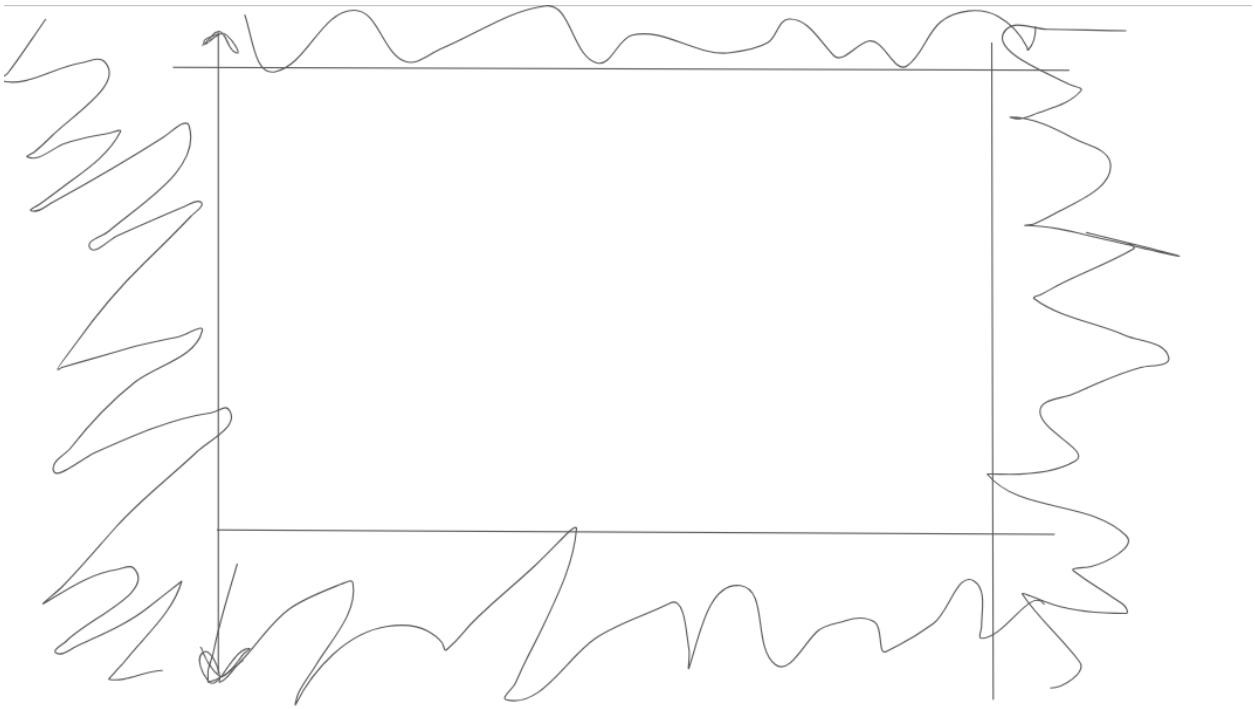
No Nodes: Bounding Box limits: $(-\infty,\infty)U(-\infty,\infty)$

Node A: x = 1, y = 1, vert, right:

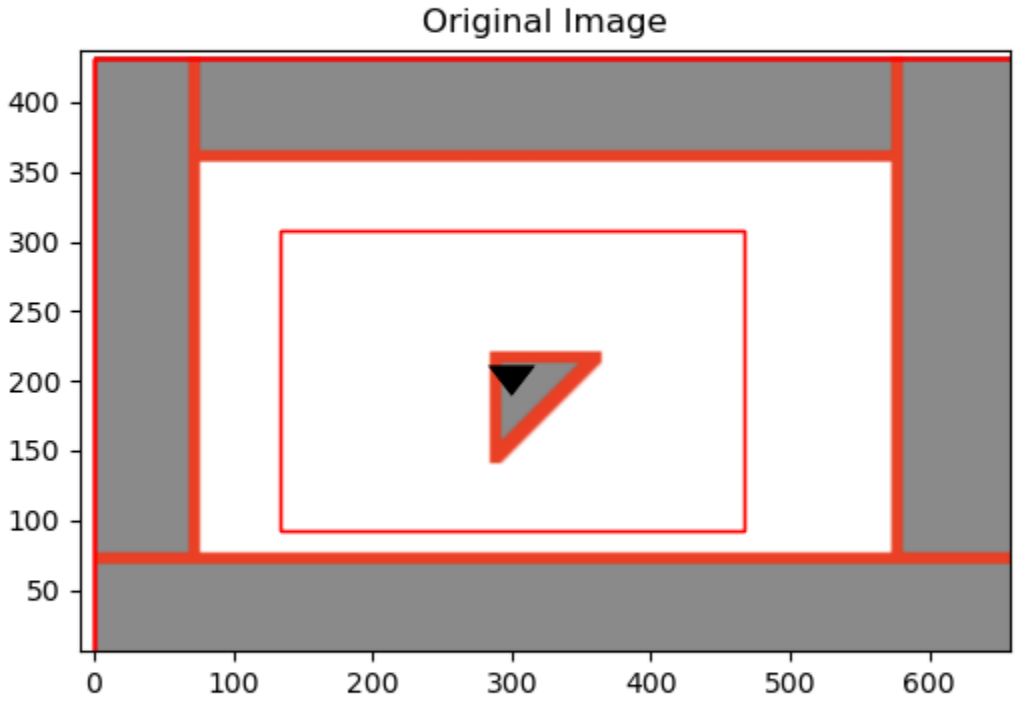


Node A: Bounding Box Limits: $(1,\infty)U(-\infty,\infty)$

This would continue until the following shape is achieved, with the squiggled sections being solid unpartitioned space:



This is where the issues arose with this approach for us. Because we were relying on a bounding box, we struggled to understand how to create bounds for a non-rectangular shape ie. the triangle in the center of the empty space. Through trial error we ended up switching our approach to python, getting the following visual shape and traversal:



```
Starting with the horizontal quadrant
Splitting horizontal into bottom-left and bottom-right
Splitting bottom-left into horizontal and vertical
Splitting horizontal into bottom-left and bottom-right
Splitting vertical into top-left and top-right
Splitting bottom-right into horizontal and vertical
Splitting horizontal into bottom-left and bottom-right
Splitting vertical into top-left and top-right
```

```
In-Order Traversal of BSP Tree:
Step 1: Quadrant: bottom-left
Step 2: Quadrant: horizontal
Step 3: Quadrant: bottom-right
Step 4: Quadrant: bottom-left
Step 5: Quadrant: top-left
Step 6: Quadrant: vertical
Step 7: Quadrant: top-right
Step 8: Quadrant: horizontal
Step 9: Quadrant: bottom-left
```

```
Step 10: Quadrant: horizontal
Step 11: Quadrant: bottom-right
Step 12: Quadrant: bottom-right
Step 13: Quadrant: top-left
Step 14: Quadrant: vertical
Step 15: Quadrant: top-right
```

Below is the Github repo for the updated python code:

https://github.com/LukeSheldon19/Homework-2-Question-1/tree/main