

CHAPTER V

IMPLEMENTATION AND TESTING

5.1. Implementation

5.1.1. User Interface Design



Figure 5 : User Interface Guillotine Glass Cutting Program

1. User input main glass length and main glass width with number.
2. Then choose one of the button
Get File – for processing only guillotine cut
Optimum Area – for processing guillotine cut and generating optimum solution
3. After button clicked then open file window will shown

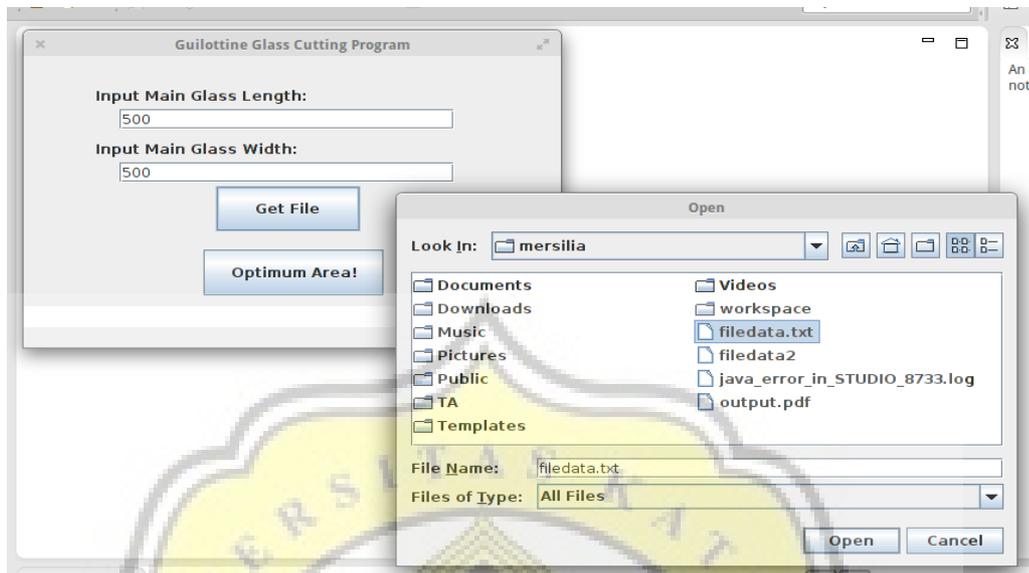


Figure 6 : *OpenFile Window*

5.1.2. Input Data Processing

In this part of the program will read the data in .txt file then change data from String data type into Integer data type

This is example of .txt file writing format:

rectangle length ,rectangle width , rectangle ID number



Figure 7 : *.txt file format and example*



```

*OpenFileOrigin.java
int idData=0;
i = 0;
//read all data in file txt
while(i < data.length)
{
    BufferedReader br = new BufferedReader (new FileReader(file));
    String line=null;

    while ((line = br.readLine())!=null)
    {
        String tmps2[]=line.split(",");
        p = Integer.parseInt(tmps2[0]);
        l = Integer.parseInt(tmps2[1]);
        idData= Integer.parseInt(tmps2[2]);
        if(data[i] == idData)
        {
            System.out.println(data[i]+"=="+idData+" with length = "+p+" width = "+l);

            RectangularPart baru = new RectangularPart(p,l,idData);
            coba.processGuillotineCut(baru.getLengthRectPart(),baru.getWidthRectPart
            (),baru.getIDRectPart());
            break;
        }
        else
        {
            System.out.println(data[i]+" ?? "+idData );
        }
    }
    i++;
}

```

Figure 8 : read txt file and change it to integer data

data.length- data is an array of id number per line, and *data.length* is count how many ID number/line in txt file chosen before.

BufferedReader read file from

```
java.io.File file = fileChooser.getSelectedFile();
```

it read per line and then in each line String data separated by `line3.split(",")` for the length,width,ID number then save it temporary at array `tmps2[]`

String data changed into Integer data with `Integer.parseInt` then save length in `p`, width in `l`, ID data in `iddata`.

5.1.3. Genetic Algorithm Implementation

Genetic algorithm handle data shuffling, waste area generating, and found the optimum result from the 1000 generation

5.1.3.1. Data Shuffling

Data shuffling will be used only if the user chooses the button "Optimum Area!" in the user interface. (Figure 5) because the "Get file" button only implements a guillotine cut.

The method `shuffleArray(data)` handles the shuffling operation for shuffling the sequence of ID data in the array data.

```

static void shuffleArray(int[] ar)
{
    Random rnd = ThreadLocalRandom.current()
    for (int i = ar.length - 1; i > 0; i--)
    {
        int index = rnd.nextInt(i + 1);
        int a = ar[index];
        ar[index] = ar[i];
        ar[i] = a;
    }
}

```

Data shuffling is in the `OpenFileGenerate.java`

5.1.3.2. Waste Area Generating

Genetic algorithm processing 1000 times for this project (processL), each process will read again the .txt file then swapped data length and width by `shuffleArray(revert)` before executing `shuffleArray(data)` for the next process. Swap process makes the combination of rectangle position rotating 90 degrees.

```

*OpenFileGenerate.java ✖

int processL = 1000;
int j = 0;
int dot;

while(j < processL)
{
    MainSheet coba=new MainSheet(MainLength,MainWidth);

    i = 0;
    set=0;
    while(i < data.length)
    {
        BufferedReader br = new BufferedReader (new FileReader(file));
        String line=null;

        while ((line = br.readLine())!=null)
        {
            String tmps2[]=line.split(",");
            int[] revert = new int[2];
            revert[0] = Integer.parseInt(tmps2[0]);
            revert[1] = Integer.parseInt(tmps2[1]);

            shuffleArray(revert);

            p = revert[0];
            l = revert[1];

            idData= Integer.parseInt(tmps2[2]);
            //p, l, idData

```

Figure 9 : swapping length and width

and then after width and length swapping, *idData* saved ID number from .txt file. This program read .txt file, shuffling ID number then save in array *data*, read data .txt again, swap the length and width of rectangle and saved ID number in array *idData*, execute the guillotine cut and then save the waste area total.

```

*OpenFileGenerate.java ✖

idData= Integer.parseInt(tmps2[2]);
//p, l, idData
if(data[i] == idData)
{
    temp[1][0] = idData;
    setIdandNumber(j,idData);
    temp[1][1] = p;
    setIdandLength(j,p);
    temp[1][2] = l;
    setIdandWidth(j,l);
    set=set+1;

    RectangularPart baru = new RectangularPart(p,l,idData);
    coba.processGuillotineCut(baru.getLengthRectPart(),baru.g
    baru.getIDRectPart());

    break;
}
else
{
    br.close();
    i++;
}
waste=coba.getEmptySheet();
setWasteArea(waste);
idsaver=coba.saveid;

setprocessNumber(j,waste);
shuffleArray(data);

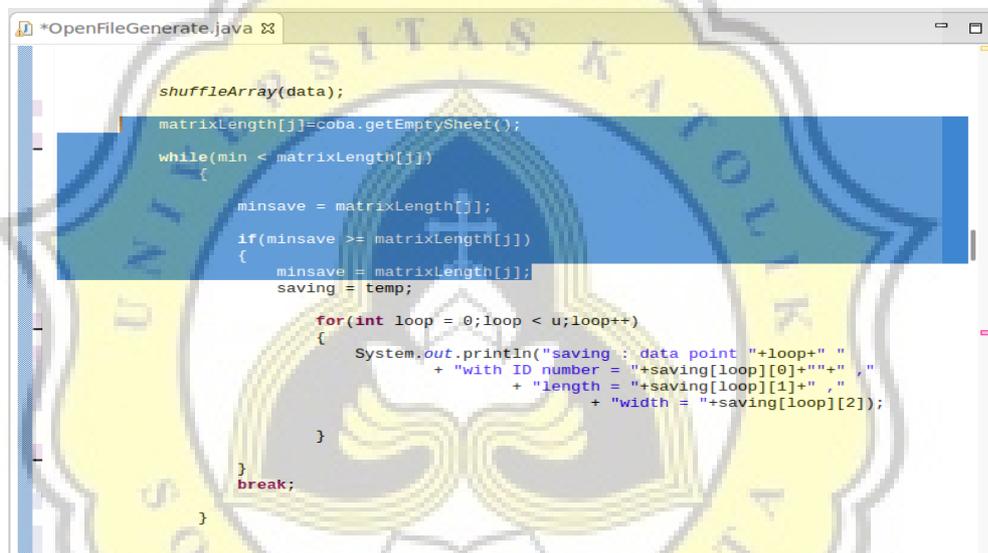
```

Figure 10 : execute guillotine cut and save waste area

Waste area get by execute *getEmptySheet()* from guillotine cut algorithm. `matrixLength[j]=coba.getEmptySheet();`
matrixLength[] length declaration must equal or higher than the *processL* length so all of the waste area can be saved.

Next is generating waste area, by search the lowest number of waste area.

`minsave=0;`



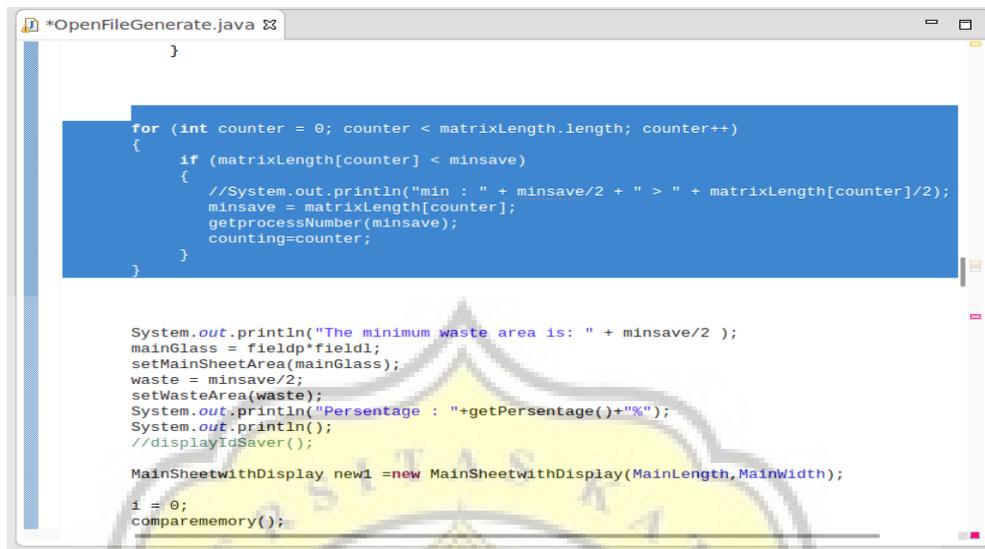
```

shuffleArray(data);
matrixLength[j]=coba.getEmptySheet();
while(min < matrixLength[j])
{
    minsave = matrixLength[j];
    if(minsave >= matrixLength[j])
    {
        minsave = matrixLength[j];
        saving = temp;
        for(int loop = 0;loop < u;loop++)
        {
            System.out.println("saving : data point "+loop+" "
                + "with ID number = "+saving[loop][0]+" "+", "
                + "length = "+saving[loop][1]+" ", "
                + "width = "+saving[loop][2]);
        }
        break;
    }
}

```

Figure 11 : get lower waste area

when the lower waste area found then save it in *minsave*



```

    }

    for (int counter = 0; counter < matrixLength.length; counter++)
    {
        if (matrixLength[counter] < minsave)
        {
            //System.out.println("min : " + minsave/2 + " > " + matrixLength[counter]/2);
            minsave = matrixLength[counter];
            getProcessNumber(minsave);
            counting=counter;
        }
    }

    System.out.println("The minimum waste area is: " + minsave/2 );
    mainGlass = fieldp*fieldl;
    setMainSheetArea(mainGlass);
    waste = minsave/2;
    setWasteArea(waste);
    System.out.println("Percentage : "+getPercentage()+"%");
    System.out.println();
    //displayIdSaver();

    MainSheetwithDisplay new1 =new MainSheetwithDisplay(MainLength,MainWidth);
    i = 0;
    comparememory();

```

Figure 12 : get lowest waste area

lowest waste area found and save it in *minsave*.

Waste area generating is in the OpenFileGenerate.java

5.1.4 Guillotine Cut Algorithm Implementation

Guillotine cut concept in this project must follows this conditions,

The valid rectangle entered :

logic 1 ; If the rectangular width \leq free area main sheet width

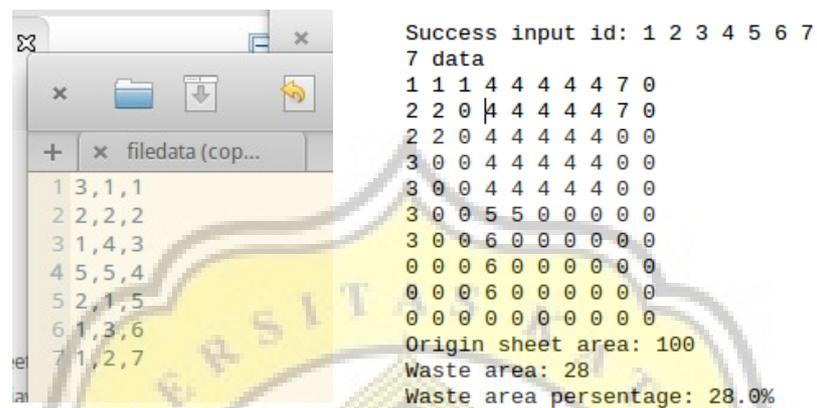
logic 2; If the rectangular length \leq free area main sheet length

The valid rectangle arrangement condition based on guillotine cut:

- Fill the main sheet (matrix data) from left-above
- Then fill under after check the space if it is still fit, if not then create a vertical line , start placed again from left-above.

- The guillotine cut will pass the rectangle which not fit enter free area in main sheet

Matrix example with data file .txt



```

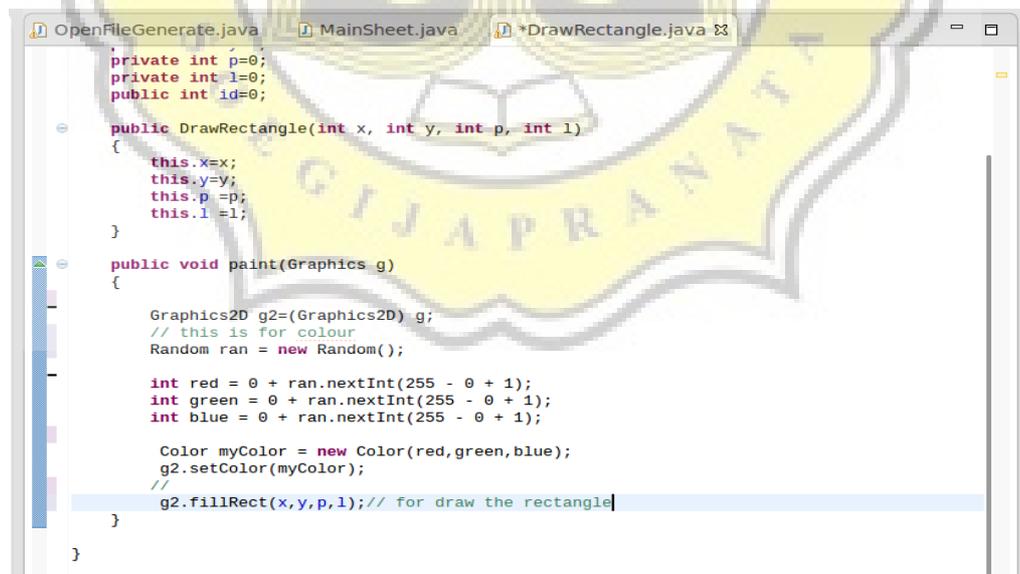
Success input id: 1 2 3 4 5 6 7
7 data
1 1 1 4 4 4 4 7 0
2 2 0 4 4 4 4 7 0
2 2 0 4 4 4 4 4 0 0
3 0 0 4 4 4 4 4 0 0
3 0 0 4 4 4 4 4 0 0
3 0 0 5 5 0 0 0 0 0
3 0 0 6 0 0 0 0 0 0
0 0 0 6 0 0 0 0 0 0
0 0 0 6 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
Origin sheet area: 100
Waste area: 28
Waste area percentage: 28.0%

```

Figure 13 & 14 : .txt file & example matrix data output with guillotine cut condition

Guillotine cut process is in the MainSheet.java (`void processGuillotineCut(int lRect, int wRect, int id)`)

5.1.5. Drawing Rectangles



```

private int p=0;
private int l=0;
public int id=0;

public DrawRectangle(int x, int y, int p, int l)
{
    this.x=x;
    this.y=y;
    this.p =p;
    this.l =l;
}

public void paint(Graphics g)
{
    Graphics2D g2=(Graphics2D) g;
    // this is for colour
    Random ran = new Random();

    int red = 0 + ran.nextInt(255 - 0 + 1);
    int green = 0 + ran.nextInt(255 - 0 + 1);
    int blue = 0 + ran.nextInt(255 - 0 + 1);

    Color myColor = new Color(red,green,blue);
    g2.setColor(myColor);
    //
    g2.fillRect(x,y,p,l);// for draw the rectangle
}
}

```

Figure 15 : Drawing rectangle implementation

Drawing rectangle is one of the feature from java

```
import java.awt.geom.Rectangle2D;
import javax.swing.*;
```

To draw the rectangle must have x coordinat, y coordinat, rectangle width, and rectangle length.

fillRect it used for draw Rectangle with colour inside and this project use the random color for each rectangles.

X and Y coordinat get from the formula of last rectangle placed and guillotine cut condition.

Drawing rectangle is in the DrawRectangle.java

5.1.6. Guillotine Cut Layout Algorithm Implementation

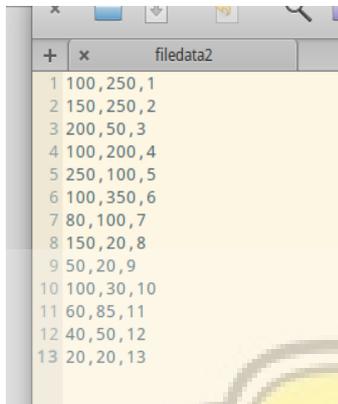
Guillotine Cut Layout Algorithm implement by add

```
DrawRectangle rectangle = new
DrawRectangle(0, 0, lRect, wRect);
trial.add(rectangle);
trial.setVisible(true);
```

into guillotine-cut function, but for implement easier this project create class guillotine cut with display in MainSheetwithDisplay.java

5.2. Testing

This project run in eclipse Kepler, tested with selected filedata.txt with following format and filled main sheet glass length x width 500 x 500 pixel



| ID | Value 1 | Value 2 | Value 3 |
|----|---------|---------|---------|
| 1 | 100 | 250 | 1 |
| 2 | 150 | 250 | 2 |
| 3 | 200 | 50 | 3 |
| 4 | 100 | 200 | 4 |
| 5 | 250 | 100 | 5 |
| 6 | 100 | 350 | 6 |
| 7 | 80 | 100 | 7 |
| 8 | 150 | 20 | 8 |
| 9 | 50 | 20 | 9 |
| 10 | 100 | 30 | 10 |
| 11 | 60 | 85 | 11 |
| 12 | 40 | 50 | 12 |
| 13 | 20 | 20 | 13 |

Figure 16 : .txt file for testing

First testing by choose button “Get File” this is the result



Figure 17 : Guillotine cut layout result

Result above only shows the guillotine cut and not all rectangles entered to the sheet and still in origin sequence of ID number, only 9 data entered.

Compare after implementing both algorithm this is the result if user choose button "Optimum Area!"(Left picture)

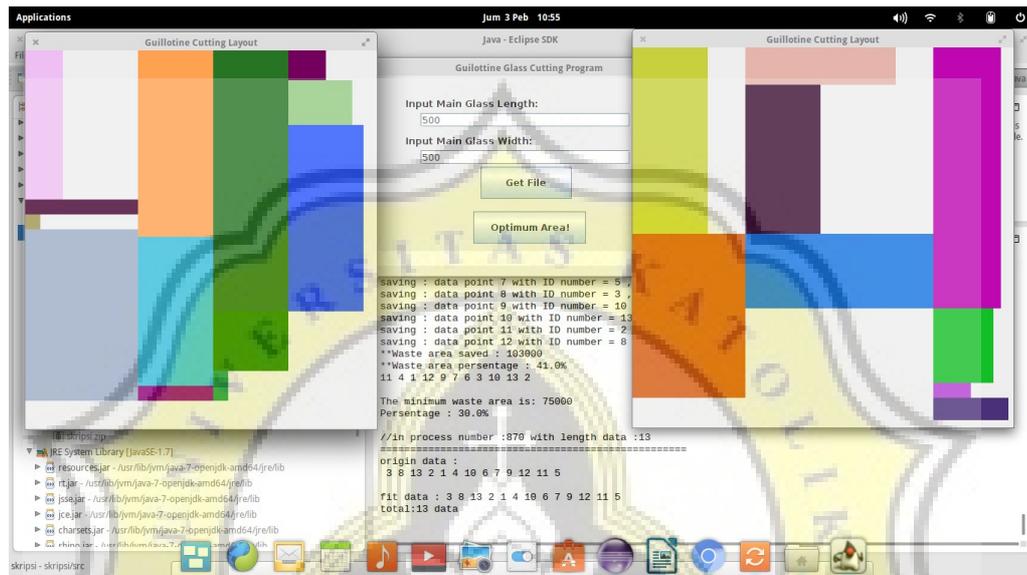


Figure 18:Generated guillotine cut layout result compare

All the data entered to the main glass sheet in the same sheet size.