Program for driving a gear with numerical commands

Murariu Ioana-Livia

Group 30433

# Introduction

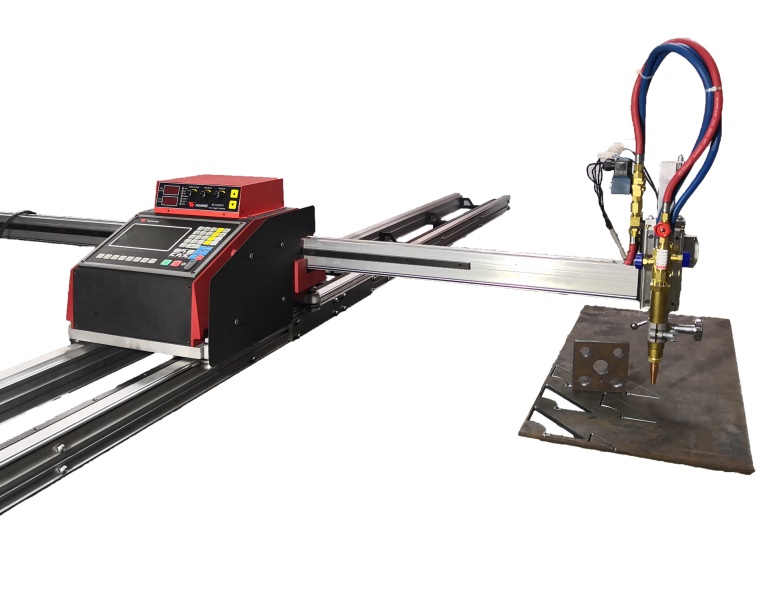
Flame cutting, also known as, Oxy Acetylene Cutting, Oxy Fuel Gas Cutting, or Oxygen Burning, is a process that use fuel gasses(or liquid fuels, such as gasoline) in order to cut or weld metals. The process is a Thermo-Chemical process, which implies the usage of source of intense heat and pure oxygen. The pure oxygen purity should be at least 99.5% purity, which is the minimum purity typically provided in cylinders and bulk systems. A reduction in purity of 0.1% will reduce cutting speed about 10%, so one can see the necessity for the purity. [1]

The process can cut from sheet metal thickness to 2800 mm material. Metal sheets thinner than 25mm are typically only cut by plasma or laser, as thin materials are really difficult to cut by flame.

The Oxy Acetylene Cutting was discovered in 1899 in USA, by accident by John Harris. Flame cutting is still used today, but for more precision it was further developed. A machine named U arm machine was created in order to enhance the accuracy and the reliability of the torch. [1]

In the 60’s and 70’s a light incorporating a cruciform was positioned over a black line. The cruciform was handled by an operator that would follow the black line and cutting heads produced the profile. The Optical Head transformed the accurate guidance of multi-headed machines.[2]

Computer guided systems were introduced in the mid 80’s. Shapes were drawn typically on a BBC computer and downloaded to a “Burny” controller on the machine. Nowadays, a range of user friendly software is available and can be operated without any special knowledge of computer programming. Most cutting machines are equipped with a marking device, which is used for making reference lines for the subsequent operations. Centre points for drilling are also marked on the machine itself. Text marking is used for part identification. The purpose of marking on the cutting machine is to reduce cycle time and improve relative dimensional accuracy of the different operations. [2]



# Objectives

The objective of this project is to simulate the Oxy Acetylene Cutting trajectory on a metal sheet.

The simulation will be developed in Java programming language, using the Eclipse IDE as support software. The trajectory of the “cut” will be determined by a set of parameters, written in a separate file. The parameters will be sent in streams of five variables, the first variable being whether the segment cut will be a straight line or a circle segment, the following four, depending on the type of segment, will be:

* If the segment is a straight line, the first two parameters are the xy coordinates of the starting point and the other two are the xy coordinates of the ending point
* If the segment is a circle segment, the first parameter is the x coordinate of the center of the circle, the second parameter is the octant which will be drawn, the third parameter is the y coordinate of the center of the circle, and the last parameter is the radius

The heat of the flame and the thickness of the metal sheet will be negligible, the purpose of this exercise being only to simulate the trajectory of the machine’s arm.

One constraint that should be applied to the parameters is that the last end point coordinates should not be larger than the size of the window, thus the trace will not continue to draw outside the window.

The cutting process will be visible in real time in the Eclipse design window. The application runtime will vary depending on the number of input streams of parameters.

# Theoretical fundaments

## 3.1 Subject applicability and domain of application

With the evolution of technology, new methods of testing the created systems and their means of working were needed. A safe way of testing the functionality of a system and the result of their execution is the computer simulation.

Computer simulation is the reproduction of behavior of the system using a computer to simulate the outcome of a mathematical model associated with that said system. The computer simulation is a safe and cheap way of verifying the model functionality and accuracy. The data for simulation can vary widely, both in size and type. For example, some simulations could only require some numbers, while others could require more parameters and conditions for testing the system. Finally, the data output visualization is set in order to represent the desired information by the user. Their appearances can be simple, like matrices or numbers, or more complex, like 2D diagrams or CGI animations. [3]

## 3.2 Application characteristics

The main applicability of the Oxy Acetylene Cutting trajectory simulator is to provide an easy way to verify the data input and output of the machine, in order to assert the desired procedure’s accuracy.

The application described in this paper focuses mainly on getting parameters from an adjacent file and giving a graphical representation of the said parameters. The simplicity of this application will ensure that it can be used by inexperienced and experienced users alike.

## 3.3 Similar applications and features

Even though they are not simulation applications, but real-life applications, the complexity of the oxyfuel burn softwares is worth mentioning.

For mechanized cutting applications, oxyfuel burn software is used to program the cutting machine. For some cases, the Computer Numeric Control (CNC) software is used to program individual parts, but most fabricators and manufacturers rely on software commonly referred to as CAD/CAM nesting software which offers far greater features and capabilities.

CAM software helps drive the manufacturing process. While there are many different types of CAM software used in industry today, CAD/CAM software is typically used for programming mechanized flat plate cutting, offering fabricators and manufacturers a single software solution for all of their cutting needs. This combination solution provides all of the necessary functionality to complete the job, for example the CAD concept, part preparation, nesting, numerical code output. [4]

Technologies used

The application is written in Java using the following components:

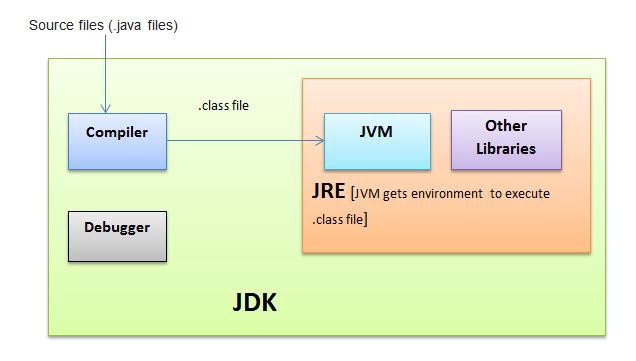
* Eclipse IDE
* JDK 1.8
* Java Swing and Java Awt

**Eclipse IDE**

Eclipse is an Integrated Development Environment used in computer programming, mostly for developing applications in Java, but can also be used for C, C#, ABAP etc. It is an open-source software that uses plug-ins to provide all the necessary functionalities. Its run-time system is based on Equinox, an implementation of the OSGi core framework specification.With the exception of a small run-time kernel, everything in Eclipse is a plug-in. Eclipse provides a large variety of plug-ins for different kinds of features, some of which are from third parties using both free and commercial models. It also provides development for servers, like Tomcat or Glassfish, and is capable of installing the required server for development directly on the IDE. [5]

**JDK 1.8**

Java Development Kit is a key platform component for building Java applications. It is one of the cores of program developing, alongside Java Virtual Machine(JVM) and Java Runtime Environment(JRE). JDK is a package of tools for developing Java-based software, whereas the JRE is a package of tools for running Java code. The JVM is the Java platform component that executes programs. [6]

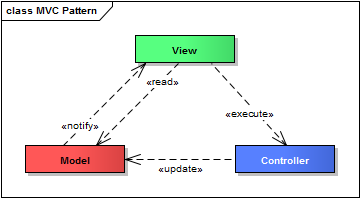


**Java Swing**

The GUI will be built using the javax.swing and the java.awt packages. The java.awt package contains all of the classes for creating user interfaces and for painting graphics and images. The java.swing package provides a set of "lightweight" (all-Java language) components that, to the maximum degree possible, work the same on all platforms. [7]

## 3.4 Design Patterns

**Model-View-Controller**

The application will be designed by using the MVC design pattern(Model-View-Controller). MVC is used to separate the application’s concerns in three parts:

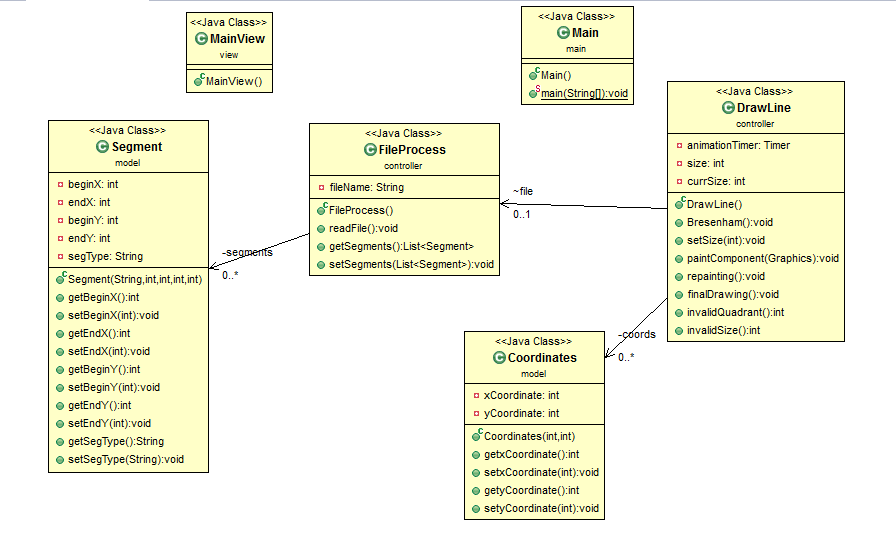
**Model** - represents the object, which carries the data which will be processed

**View** - represents the visualization of the data that the model carries

**Controller** - is the “master” of the application; it controls the data flow of the models and updates the views whenever the model changes

This design pattern was chosen in order to maintain the clarity of the software’s structure, and to enhance the application development process. [8]

## 3.5 Implementation structure



# Detailed Design and Implementation

## 4.1 Detailed Design

The application structure is based on the MVC design structure. The packages which make the structure is:

* **Model** – contains all the data that will be processed
* **Controller** – contains the methods that directs and transforms the data
* **View** – makes the visualization of data
* **Main** – contains the main function of the application

### 4.1.1 Model

The package model contains the classes **Segment** and **Coordinates**.

**Class Segment**

The class Segment represents a classification of the data that is read from the data.txt. The parameters are:

* **Private String segType**, which represents the type of the read segment: circle or straight
* **Private Int beginX**, which represents for the straight line the x coordinate of the start point, and for the circle line the x coordinate for the center of the circle
* **Private Int endX**, which represents for the straight line the x coordinate of the end point, and for the circle line the quadrant in which we want the circle line to be drawn
* **Private int beginY** , which represents for the straight line the y coordinate of the start point, and for the circle line the y coordinate for the center of the circle
* **Private Int endY**, which represents for the straight line the y coordinate of the end point, and for the circle line the radius of the circle

The class additionally contains methods for getters, setters, and constructors.

**Class Coordinates**

The class Coordinates represents the coordinates of the point that will be drawn. These data will create a “pool” of information, from which the drawing algorithm will extract directly the position info of each pixel. The parameters are:

* **private int xCoordinate**, which represents the x coordinate of the point
* **private int xCoordinate**, which represents the y coordinate of the point

### 4.1.2 View

Package view contains the class **MainView**.

**Class MainView**

The class **MainView** displays the central panel that will show the representation of the trace let by the cutter.

The application is set to close by default when the window is closed. Then, the size and position of the window are set.

### 4.1.3 Package controller

The package contains the classes **FileProcess** and **DrawLine**.

**Class FileProcess**

The purpose of this class is to take the data entry from the file data.txt and input it in a list of type *Segment*, **segments**.

The list will be later on run through, in order to display all the segments whose parameters were written in the data.

The data will be extracted by using streams and lambda expressions. Using the Stream *stream*, the data is processed, and for each line in the stream, a new **Segment** element is created. The data is processed by splitting it and giving it to their respective attributes.

**Lambda expressions** are introduced in Java 8 and are touted to be the biggest feature of Java 8.

**Lambda expression** facilitates functional programming, and simplifies the development a lot. In computer programming, an anonymous function (function literal, lambda abstraction, or lambda expression) is a function definition that is not bound to an identifier. Anonymous functions are often arguments being passed to higher-order functions, or used for constructing the result of a higher-order function that needs to return a function. If the function is only used once, or a limited number of times, an anonymous function may be syntactically lighter than using a named function. Anonymous functions are ubiquitous in functional programming languages and other languages with first-class functions, where they fulfill the same role for the function type as literals do for other data types. [9]

A **stream** provides an interface to a sequenced set of values of a specific element type. However, streams don’t actually store elements; they are computed on demand. Streams consume from a data providing source such as collections, arrays, or I/O resources. [10]

The class also has as parameter the path to the file from where we get the data, **fileName**.

**Class DrawLine**

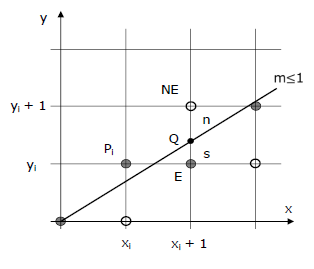
The **DrawLine** class is the most complex class. It generates the visual representation of the segments and the animation. The class extends the class **JComponent**, which is the base class for all Swing components except top-level containers, and provides a "pluggable look and feel" (L&F) that can be specified by the programmer or selected by the user at runtime. [11]

The idea of making the real-time simulation of the movement of the flame head is to draw each segment pixel by pixel. For that, the **Bresenham** algorithm for line and circle was used. The full implementation of both **Bresenham** Algorithms can be seen on the **Appendix**, **section 1**.

#### Bersenham Algorithm for line

The Bersenham Algorithm is a line drawing algorithm that determines the points of a raster that should be selected in order to represent a close approximation to the line between two points.

The basic idea is to find the closest integer coordinates to the actual line path, and compute iteratively the next point. We shall compute a decision variable d, and depending on this value, we will choose the next position E or NE, where E is the position under the line and NE is the position above the line.



#### Bresenham Algorithm for circle

The main difference between this algorithm and the previous one is that we will chose the pixel in the east (x+1, y)  or the west or in the south east (x+1, y-1). This can be decided using the decision parameter d as If d > 0, then (x+1, y-1) is to be chosen as the next pixel as it will be closer to the arc, else (x+1, y) is to be chosen as next pixel.

The algorithm here presented is for drawing a full circle, but in our application it was modified in order to draw a (chosen) octet of a circle.

The class Overrides the method **paintComponent()**. In this method, all the painting takes place, and takes Graphic objects as parameters. Inside this method, the Bresenham algorithms are called, from the method Bresenham().[12] These algorithms are accessed as follows:

* We iterate through the list of Segments
* If **segTyp** is equal with *str*, we call the Bresenham line algorithm
* If **segTyp** is equal with *cir*, we call the Bresenham circle algorithm

Before calling each of the algorithms above, we make sure that the input data is correct by using asserts.

**assert** is a Java keyword used to define an assert statement. An assert statement is used to declare an expected boolean condition in a program. If the program is running with assertions enabled, then the condition is checked at runtime. If the condition is false, the Java runtime system throws an AssertionError. [12]

In our case, if the assertions return false, a **JOptionPane** will be shown.

The real-time animation is implemented using the **AnimationTimer** class. The class **AnimationTimer** allows to create a timer, that is called in each frame while it is active. An extending class has to override the method handle (long) which will be called in every frame. The methods **start()** and **stop()** allow to start and stop the timer. [13] The implementation using **AnimationTimer** can be seen on **Appendix, section 3**.

In this project, the line is not properly drawn pixel by pixel, but each iteration of the function **repainting()** will add a new pixel to the line that is to be drawn, thus creating the illusion of the drawing being created pixel by pixel. The pixels’ coordinates are being stored in the List cords, for a better access in the drawing algorithm.

Additionally, a grid is drawn on the panel, so it is easier to give the input data. The grid is drawn using the **g.drawLine** method, which takes as parameters the coordinates of the beginning point and the end point.

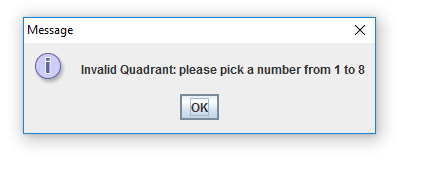
# Experimental Results

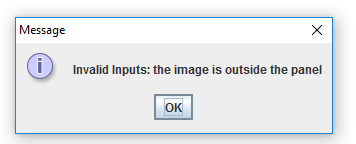
The user enters the data in the *data.txt* file. The data consists in lines of 5 parameters, where:

* the first parameter is the type of line: cir(circle) or str(straight)
* if the type of line is *cir*
  + the second parameter is the x coordinate of the circle
  + the third parameter is the octant of the circle we want to draw
  + the fourth is the y coordinate of the center of the circle
  + the last parameter is the length of the radius of the circle
* If the type of line is *str*
  + the second parameter is the x coordinate of the start of the line
  + the second parameter is the y coordinate of the start of the line
  + the second parameter is the x coordinate of the end of the line
  + the second parameter is the y coordinate of the end of the line

If the parameters do not correspond to the requests (The octant does not exist, the line is outside the window), the application will return a pop-up with a message.



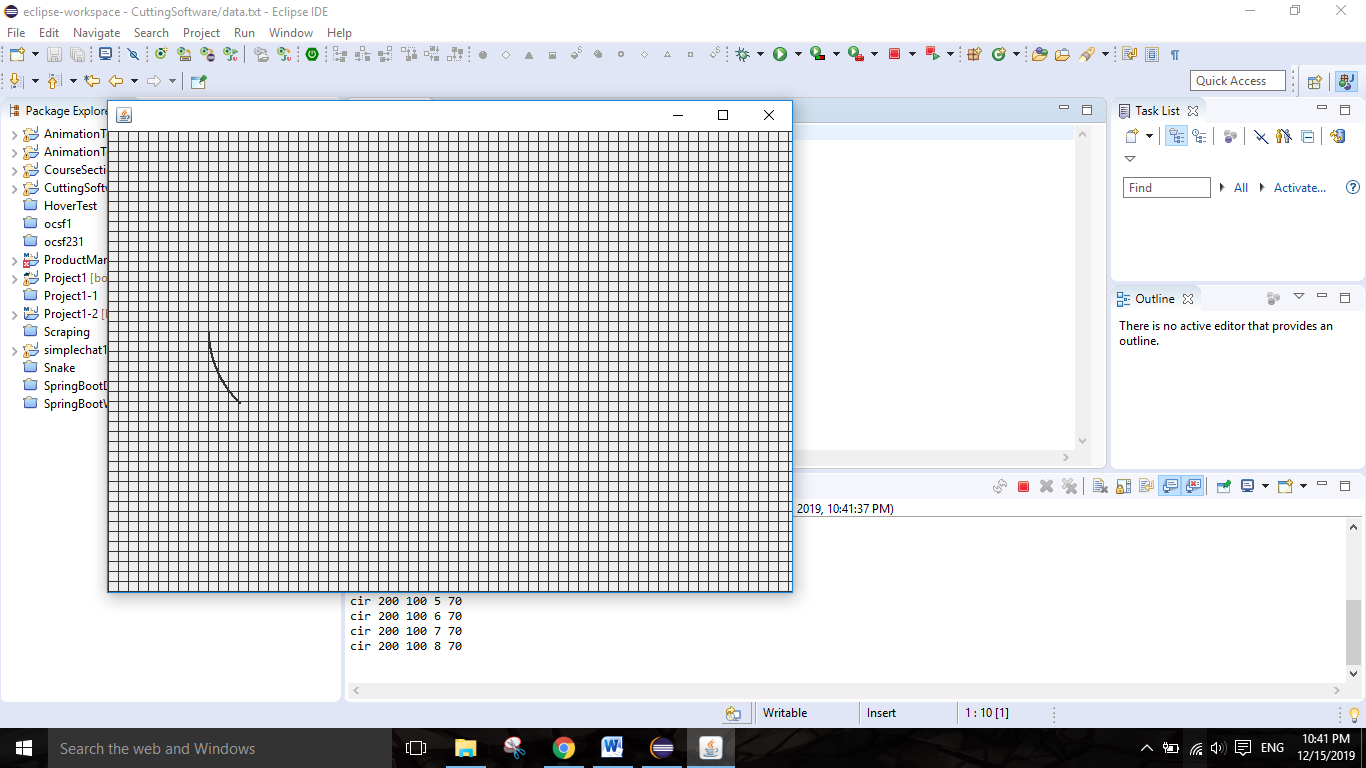




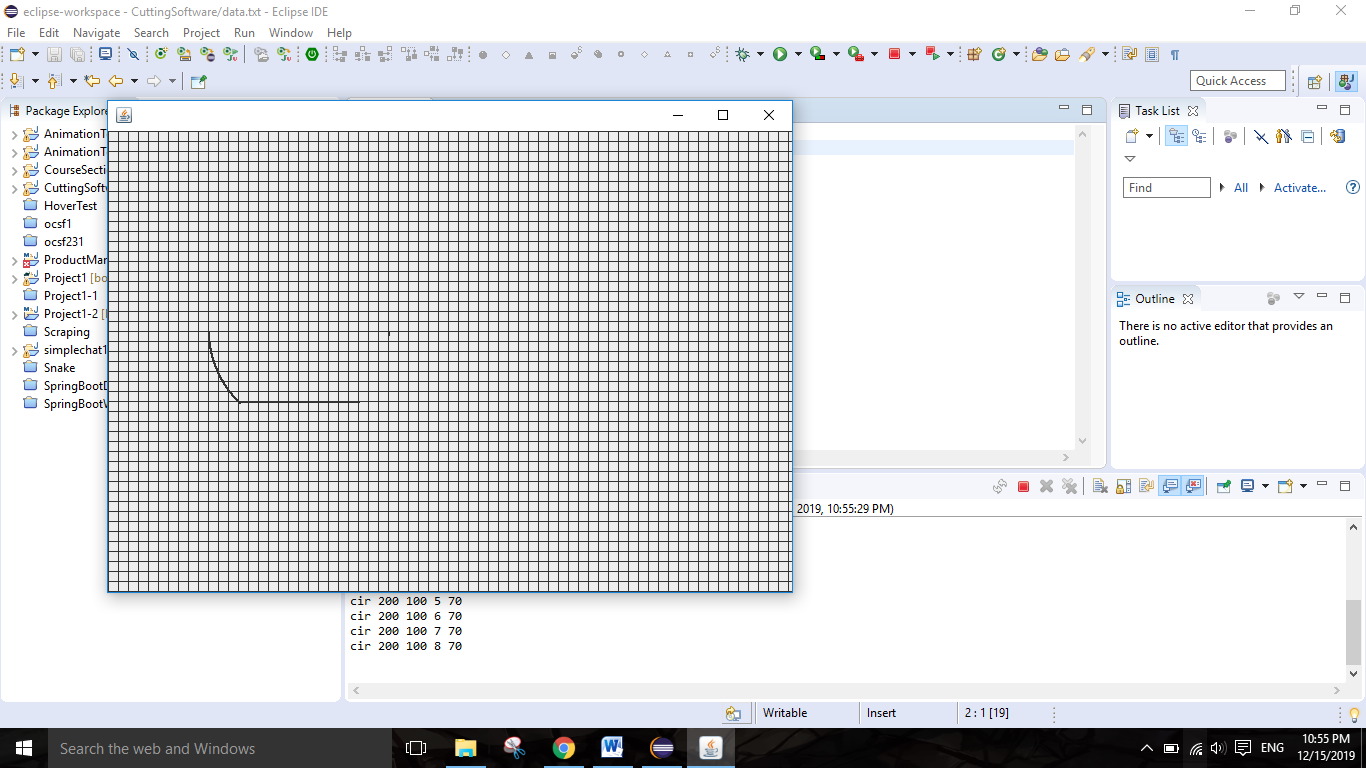
If the coordinates are correct, the application will start and the animation will take place. Each line is drawn in the order of the file.

We want to draw a flower in a pot.

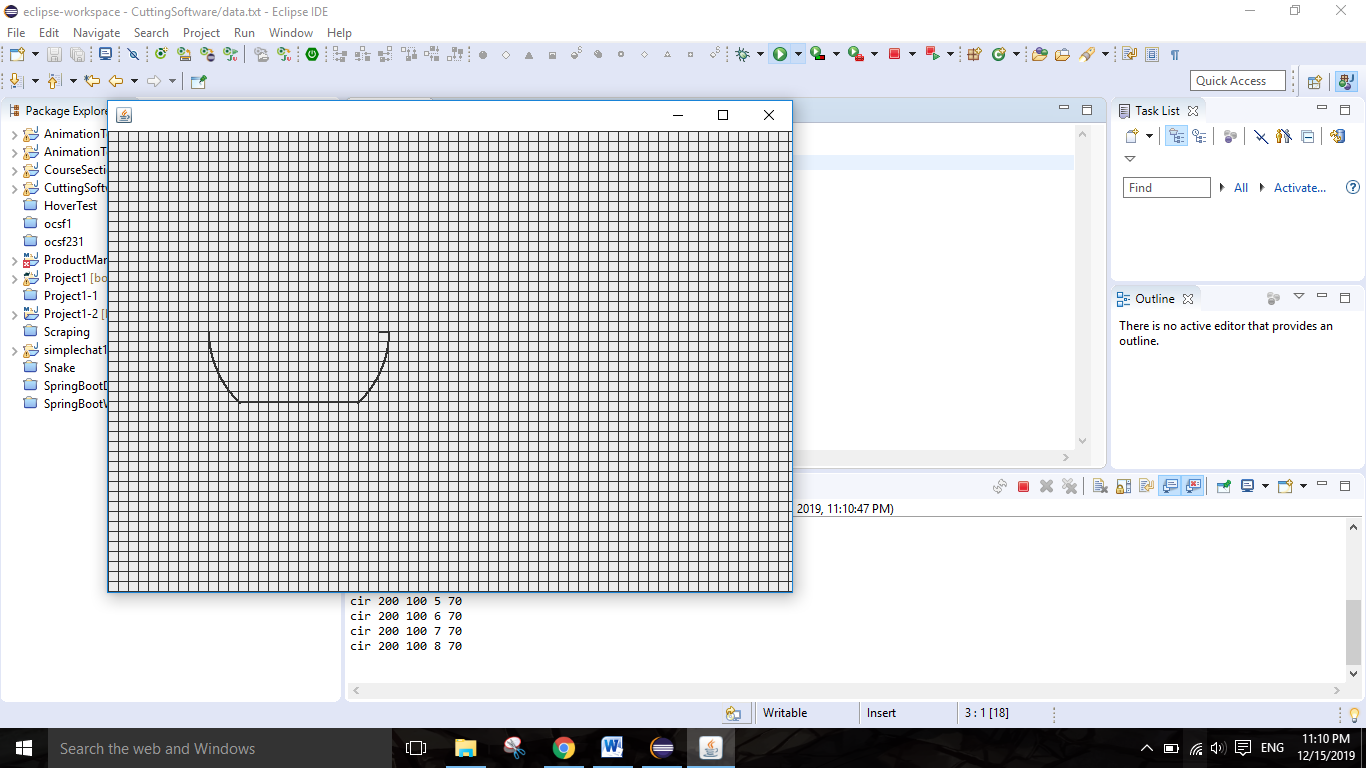
1. The first set of parameters is *cir 200 6 200 100*, so a segment of a circle in the sixth octant with the center on coordinates *(200,200)* and the radius of *100* will be drawn. The coordinates can be verified on the grid, each unit of the grid being equal with 10 points of the given coordinates.



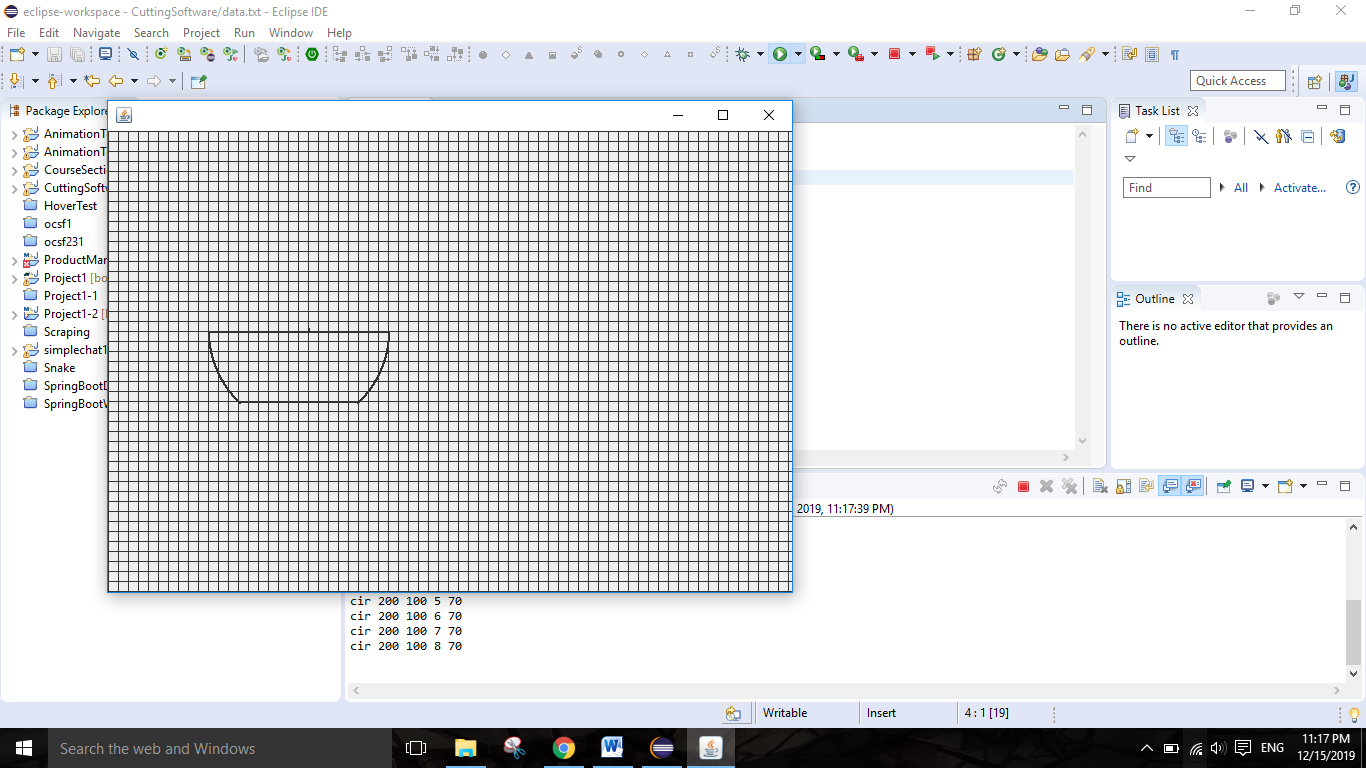
1. *str 130 250 270 270*, a straight line with the initial coordinates *(130,250)* and the final coordinates *(270,270).* The Bresenham algorithm was modified in that way that is possible to make an horizontal or vertical line, by giving the delta x/delta y an implicit value of *100* whenever it is required.



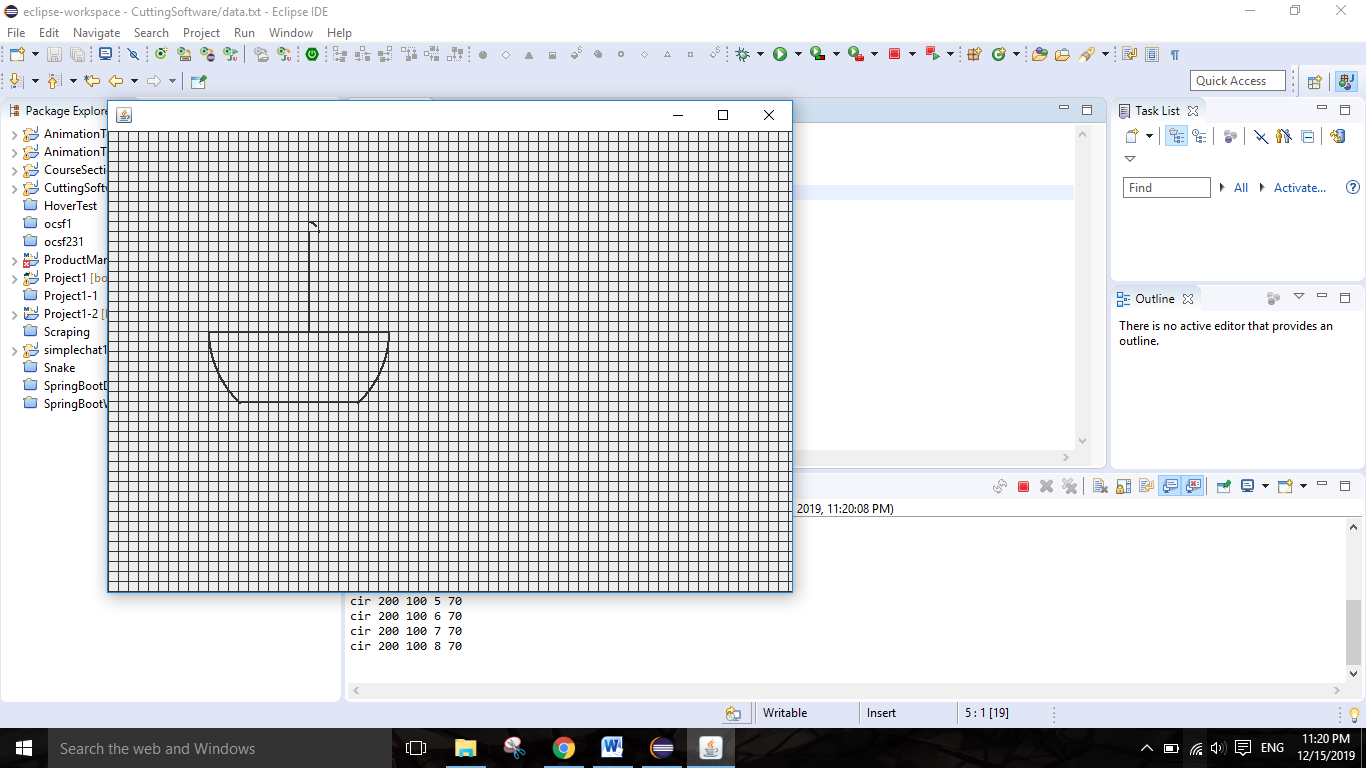
1. *cir 180 4 200 100*so a segment of a circle in the fourth octant with the center on coordinates *(180,200)* and the radius of *100* will be drawn.



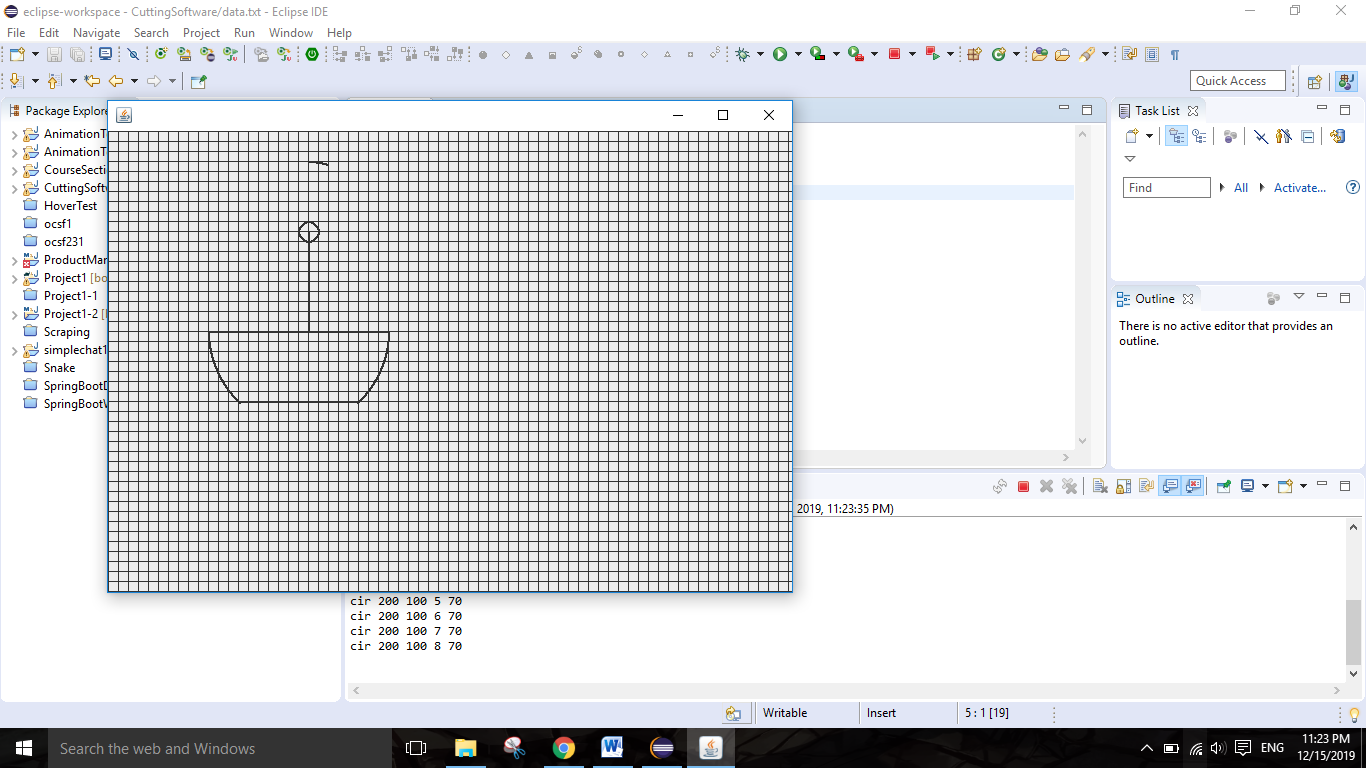
1. *str 280 100 200 200*creates a straight line from *(280,100)* to *(200,200),* so backwards. With this, the pot is complete



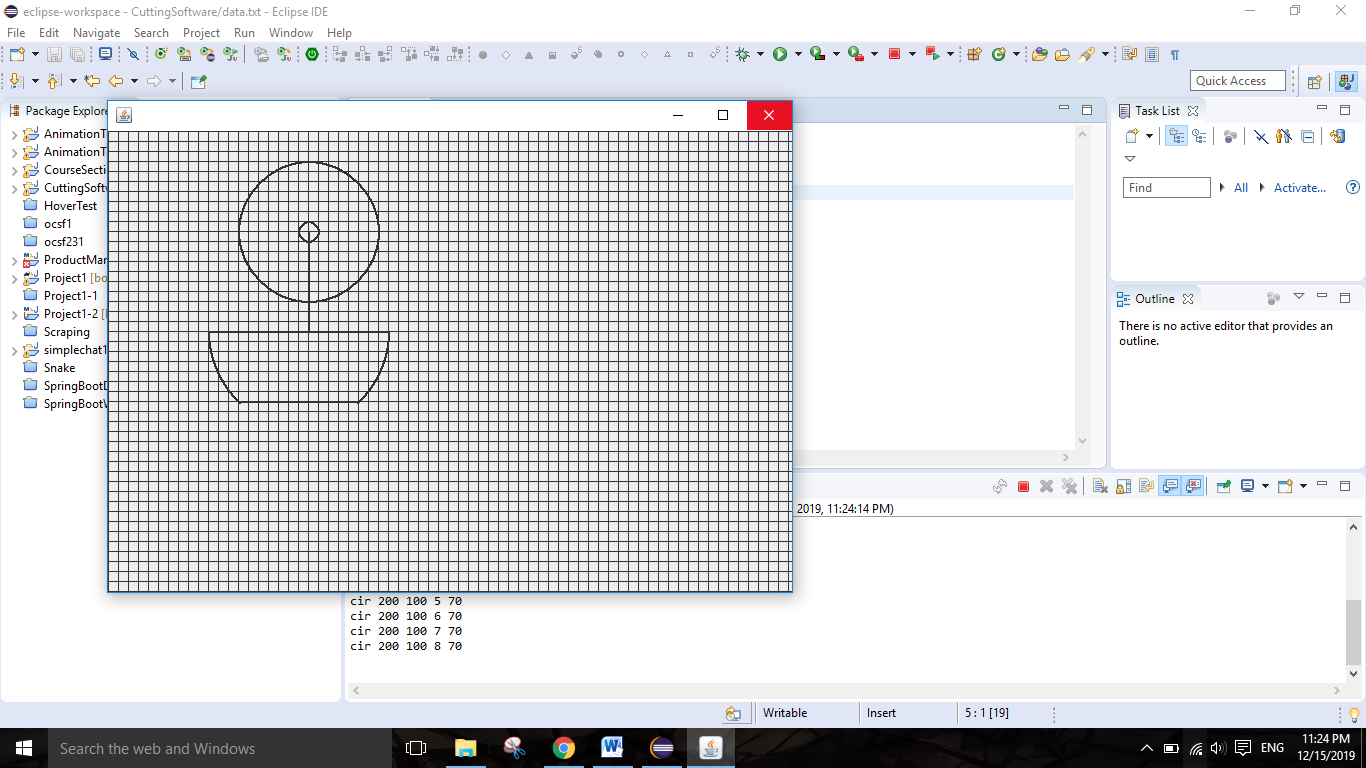
1. *str 200 200 200 100*creates a vertical line that will represent the stem of the flower.



1. the next 8 instructions will be used to create the ”receptacle” of the flower, thus a full circle with the origin in *(200,100)* and the radius of *10*



1. the final 8 instructions will create the “petals” of the flower, thus a full circle with the origin in *(200,100)* and the radius of *100*



# Conclusion

In conclusion, this application has helped me to enhance my Java developing skills and my planning techniques. Furthermore, the theme of the project, program for simulating a gear with numerical commands, gave me an example of the practical use of acquiring knowledge in the programming domain.

The mixture of techniques and algorithms used to create the simulation highlighted the interoperability between different kinds of knowledge acquired during the college time: the Bresenham algorithm studied at Elements of Computer Assisted Graphics course, the Java skills developed during the Programming Techniques course etc.

# Bibliography

1. <https://www.esabna.com/us/en/education/blog/what-is-flame-cutting.cfm>
2. <http://www.malthouseengineering.com/what-is-flame-cutting/>
3. <https://en.wikipedia.org/wiki/Computer_simulation#History>
4. <https://www.hypertherm.com/en-US/learn/cutting-education/understanding-oxyfuel-technology/?region=NART>
5. <https://en.wikipedia.org/wiki/Eclipse_(software)>
6. <https://www.javaworld.com/article/3296360/what-is-the-jdk-introduction-to-the-java-development-kit.html>
7. <https://docs.oracle.com/%20javase/%207/%20docs/%20api/%20javax/%20swing/%20package-summary.html>
8. <https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm>
9. <https://www.tutorialspoint.com/java8/java8_lambda_expressions.htm>
10. <https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>
11. <https://docs.oracle.com/javase/7/docs/api/javax/swing/JComponent.html>
12. <https://docs.oracle.com/javase/tutorial/uiswing/painting/closer.html>
13. <https://docs.oracle.com/javafx/2/api/javafx/animation/AnimationTimer.html>
14. <http://csourcecodes.blogspot.com/2016/06/bresenhams-line-drawing-algorithm-generalized-c-program.html> ---for appendix 1
15. <https://www.geeksforgeeks.org/bresenhams-circle-drawing-algorithm/> ---for appendix 1

# Appendix

## [11] [12]

**public** **void** Bresenham() {

file.readFile();

**for** (Segment s : file.getSegments()) {

**if** (s.getSegType().equals("str")) {

**int** x, y, dx, dy, swap, temp, s1, s2, p, i;

x = s.getBeginX();

y = s.getBeginY();

**assert** s.getBeginX()>0 || s.getBeginX()<700 || s.getEndX()>0 || s.getEndX()<700 ||

s.getBeginY()>0 || s.getBeginY()<500 || s.getEndY()>0 || s.getEndY()<500 : invalidSize();

**if** (s.getEndX() == s.getBeginX())

dx = 100;

**else**

dx = Math.*abs*(s.getEndX() - s.getBeginX());

**if** (s.getEndY() == s.getBeginY())

dy = 100;

**else**

dy = Math.*abs*(s.getEndY() - s.getBeginY());

s1 = (**int**) Math.*signum*(s.getEndX() - s.getBeginX());

s2 = (**int**) Math.*signum*(s.getEndY() - s.getBeginY());

swap = 0;

coords.add(**new** Coordinates(s.getBeginX(), s.getBeginY()));

**if** (dx > dy) {

temp = dx;

dx = dy;

dy = temp;

swap = 1;

}

p = 2 \* dy - dx;

**for** (i = 0; i < dx; i++) {

coords.add(**new** Coordinates(x, y));

**while** (p >= 0) {

p = p - 2 \* dx;

**if** (swap != 0)

x += s1;

**else**

y += s2;

}

p = p + 2 \* dy;

**if** (swap != 0)

y += s2;

**else**

x += s1;

}

coords.add(**new** Coordinates(s.getEndX(), s.getEndY()));

}

**if** (s.getSegType().equals("cir")) {

**int** r = s.getEndY();

**int** x = 0, y = r;

**int** d = 3 - 2 \* r;

**assert** s.getEndX() == 1 || s.getEndX() == 2 || s.getEndX() == 3 || s.getEndX() == 4 || s.getEndX() == 5

|| s.getEndX() == 6 || s.getEndX() == 7 || s.getEndX() == 8 : invalidQuadrant();

**switch** (s.getEndX()) {

**case** 1:

coords.add(**new** Coordinates(s.getBeginX() + x, s.getBeginY() - y));

**break**;

**case** 2:

coords.add(**new** Coordinates(s.getBeginX() + y, s.getBeginY() - x));

**break**;

**case** 3:

coords.add(**new** Coordinates(s.getBeginX() + x, s.getBeginY() + y));

**break**;

**case** 4:

coords.add(**new** Coordinates(s.getBeginX() + y, s.getBeginY() + x));

**break**;

**case** 5:

coords.add(**new** Coordinates(s.getBeginX() - x, s.getBeginY() + y));

**break**;

**case** 6:

coords.add(**new** Coordinates(s.getBeginX() - y, s.getBeginY() + x));

**break**;

**case** 7:

coords.add(**new** Coordinates(s.getBeginX() - y, s.getBeginY() - x));

**break**;

**case** 8:

coords.add(**new** Coordinates(s.getBeginX() - x, s.getBeginY() - y));

**break**;

}

**while** (x <= y) {

// for each pixel we will

// draw all eight pixels

x++;

// check for decision parameter

// and correspondingly

// update d, x, y

**if** (d > 0) {

y--;

d = d + 4 \* (x - y) + 10;

} **else**

d = d + 4 \* x + 6;

**switch** (s.getEndX()) {

**case** 1:

coords.add(**new** Coordinates(s.getBeginX() + x, s.getBeginY() - y));

**break**;

**case** 2:

coords.add(**new** Coordinates(s.getBeginX() + y, s.getBeginY() - x));

**break**;

**case** 3:

coords.add(**new** Coordinates(s.getBeginX() + x, s.getBeginY() + y));

**break**;

**case** 4:

coords.add(**new** Coordinates(s.getBeginX() + y, s.getBeginY() + x));

**break**;

**case** 5:

coords.add(**new** Coordinates(s.getBeginX() - x, s.getBeginY() + y));

**break**;

**case** 6:

coords.add(**new** Coordinates(s.getBeginX() - y, s.getBeginY() + x));

**break**;

**case** 7:

coords.add(**new** Coordinates(s.getBeginX() - y, s.getBeginY() - x));

**break**;

**case** 8:

coords.add(**new** Coordinates(s.getBeginX() - x, s.getBeginY() - y));

**break**;

}

}

}

}

}

## 2.

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

**for** (**int** i = 0; i < 70; ++i) {

g.drawLine(0, i \* 10, 700, i \* 10);

}

**for** (**int** i = 0; i < 80; ++i) {

g.drawLine(i \* 10, 0, i \* 10, 500);

}

**for** (**int** i = 0; i < size; ++i)

g.drawRect(coords.get(i).getxCoordinate(), coords.get(i).getyCoordinate(), 1, 1);

}

## 3.

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

**for** (**int** i = 0; i < 50; ++i) {

g.drawLine(0, i \* 10, 700, i \* 10);

}

**for** (**int** i = 0; i < 70; ++i) {

g.drawLine(i \* 10, 0, i \* 10, 500);

}

**for** (**int** i = 0; i < size; ++i)

g.drawRect(coords.get(i).getxCoordinate(), coords.get(i).getyCoordinate(), 1, 1);

}