Implementing Lindenmayer Systems

Simon Scorer
BSc Computer Science & Mathematics
(International)
2004/2005

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) ______________________________

Summary

The objectives of this project were to research the theory of Lindenmayer systems (L-systems), and then to develop a program with a user friendly interface that was capable of producing graphical representations of them. L-systems are, in short, systems of rules which can be used to model some aspects of plant growth and also some types of geometrical patterns in 2-dimensions. The main purpose of this program is for use as an educational or research tool for investigating L-systems.

These objectives were achieved with the design and implementation of a fully functioning program that can draw and edit various forms of L-system. The program was then tested and evaluated to determine how well it met its specified requirements.

Table of Contents

Chapter 1 – Introduction **1**

1.1 Aim 1

1.2 Objectives 1

1.3 Minimum Requirements 1

1.4 Project Schedules 1

Chapter 2 – Background Research **4**

2.1 L-system Theory 4

2.1.1 Background 4

2.1.2 Formal Definition of DOL-systems 5

2.1.3 Geometric Interpretation 6

2.1.3.1 2-Dimensional Turtle Interpretation 6

2.1.3.2 3-Dimensional Turtle Interpretation 8

2.1.4 Stochastic L-systems 8

2.2 Research Into Currently Available Software 8

2.2.1 Web-based Software 8

2.2.2 Fractint 10

2.3 Quality of Resources 10

Chapter 3 – Design **11**

3.1 Overview 11

3.2 System Requirements 11

3.3 The Graphical User Interface 11

3.4 Proposed Method 12

3.5 Programming Language 13

3.5.1 The Java Swing API 13

3.6 UML Outline of Proposed System 13

Chapter 4 – Implementation **15**

4.1 Overview 15

4.2 L-system Alphabet 15

4.3 Input Processing 15

4.3.1 LSystem.java 15

4.3.2 Turtle.java 16

4.4 Drawing the Image – LSystemDiagram.java 18

4.4.1 Scaling and Positioning the Image 20
4.5 The Graphical User Interface – Display.java 21
4.5.1 Layout Arrangement 22
4.5.2 Adding Actions 23
4.5.2.1 Drawing the Diagram 23
4.5.2.2 Loading Pre-set L-systems 24
4.5.3 The File Menu 24
4.5.3.1 Setting Image Size 24
4.5.3.2 Saving Images to File 25
4.6 Error Handling 25
4.7 Reliability 26
4.8 Problems Encountered 26

Chapter 5 – Evaluation 27
5.1 Evaluation Criteria 27
5.2 Evaluating the Program 27
5.2.1 Image Output 27
5.2.2 Handling Unreliable Input 27
5.2.3 Dealing with Large Computations 28
5.2.4 Comparison to Available Software 29
5.3 Possible Extensions 30
5.3.1 3D L-systems 30
5.3.2 Stochastic L-systems 30
5.3.3 Improving the Interface 30

Bibliography 31

Appendices
A Project Experience 32
B UML Diagram 33
C Screen Shots 34
D Produced Images 36

1 Introduction

1.1 Aim

This project aims to research the theory of Lindenmayer systems (L-systems) and ultimately to develop a computer program that is capable of producing graphical representations of them. The main purpose of such a program will be for use as an educational tool. The program will aid people in understanding and investigating L-systems. There are likely to be limited practical or commercial opportunities for such a program, but it could be used by people to explore an interesting subject area.

This project will draw on programming experience gained during my time at the University and also possibly on elements of algorithm analysis and computer graphics.

1.2 Objectives

The objectives of this project are to:

- Research the theory of L-systems and produce a summary of important ideas and definitions.
- Research currently available software that performs similar tasks and, with these in mind, decide upon the specific requirements of my program.
- Design and implement a program that is capable of producing 2-dimensional graphical representations of L-systems.

1.3 Minimum Requirements

The agreed minimum requirements of this project are:

- To produce an account of the basic theory of L-systems including a classification of the basic types.
- To produce a system in Java that can be given an expression and will output a graphical representation of this expression.
- That the user will be able to change the initial expression (axiom) and view the effect that this has on the graphical representation.
- That the user of the system will be able to change various other parameters of the L-system and view the effect that this has on the graphical representation.

Some possible extensions are:

- To produce a user manual that will accompany the system.
- To extend the graphical output of the system from 2 dimensions to 3.

1.4 Project Schedules

My original project schedule, as appeared in the mid-project report can be seen in table 1.1.

Task	Completion Date
Research L-system theory & definitions	19/11/04
Research similar systems currently available	26/11/04
Write up a draft Research chapter	26/11/04
Plan the basic functions of my system	03/12/04
Design the basic layout of my GUI	03/12/04
*** submit mid-project report ***	**09/12/04**
Design the outline of the Java program using UML	10/12/04
Implement the GUI	10/12/04
(SEMESTER 1 EXAM PERIOD : 10/01/05 - 21/01/05)	
Write up a draft Design chapter	28/01/05
Implement a basic working system	04/02/05
Implement a final working system	04/03/05
Write up a draft implementation chapter	11/03/05
*** submit table of contents and a draft chapter ***	**11/03/05**
Test & evaluate system	18/03/05
Write up a draft Evaluation chapter	01/04/05
Put Project Report together	15/04/05
*** submit project report ***	**27/04/05**

Table 1.1 : Project Plan

You will notice that the scheduled date for completion of the implementation of the program is almost 2 months before the deadline for submission of the project report. This was to allow sufficient leeway in the event of any delays in the implementation process. It also allowed enough time to perform testing and evaluation as well as the considerable task of compiling the project report. You will also notice that there is a large gap between December 10th and January 28th, this is to allow revision for and completion of the first semester exams in January.

The implementation of the program, and in particular the graphical user interface, proved to be quite challenging and time consuming tasks. Partly due to the fact that I had limited previous experience of producing complex graphical programs. The knock-on effect of this was to push back many of the subsequent deadlines. Enough time had been allowed for delays of this nature, and the project was still completed on schedule. The revised project schedule can be seen in table 1.2.

Task	Completion Date
Research L-system theory & definitions	19/11/04
Research similar systems currently available	26/11/04
Write up a draft Research chapter	26/11/04
Plan the basic functions of my system	03/12/04
Design the basic layout of my GUI	03/12/04
*** submit mid-project report ***	**09/12/04**
Design the outline of the Java program using UML	10/12/04
(SEMESTER 1 EXAM PERIOD : 10/01/05 - 21/01/05)	
Implement a prototype GUI	28/01/05
Write up a draft Design chapter	11/02/05
Implement a basic working system	18/02/05
*** submit table of contents and a draft chapter ***	**11/03/05**
Implement a final working system	18/03/05
Write up a draft implementation chapter	25/03/05
Test & evaluate system	08/04/05
Write up a draft Evaluation chapter	15/04/05
Put Project Report together	22/04/05
*** submit project report ***	**27/04/05**

Table 1.2 : Revised Project Plan

2 Background Research

2.1 L-system Theory

2.1.1 Background

Lindenmayer Systems (L-systems) are named after the man who developed them – a biologist called Aristid Lindenmayer. They are, in short, systems of rules which can be used to model some aspects of plant growth and also some types of geometrical patterns in 2-dimensions.

Many of the ideas discussed in this chapter are from a book by Przemyslaw Prusinkiewicz and Aristid Lindenmayer (1990) called 'The Algorithmic Beauty of Plants'. Rozenberg and Salomaa (1980) also discuss the theory of L-systems but go into more mathematical detail than is required for this project.

L-systems were initially designed as a way of modelling the development of plants and other cellular organisms. It is obvious that many plants possess a quality known as 'self-similarity', discussed by Mandelbrot (1983), which is where small sections of an object strongly resemble the object as a whole. This can be seen, for example, in many trees, where if you look at individual branch sections or leaves they can often look geometrically very similar to the whole tree.

L-systems use the idea of rewriting to model this effect of self-similarity. Prusinkiewicz and Lindenmayer (1990) describe rewriting as the process of successively replacing parts of an initial object using a set of rewriting rules or productions. They illustrate this idea with the example of the Koch Snowflake (figure 2.1). The principle of its construction is described by Mandelbrot (1983) as follows:

> One begins with two shapes, an initiator and a generator. The latter is an oriented broken line made up of N equal sides of length r. Thus each stage of the construction begins with a broken line and consists in replacing each straight interval with a copy of the generator, reduced and displaced so as to have the same end points as those of the interval being replaced.

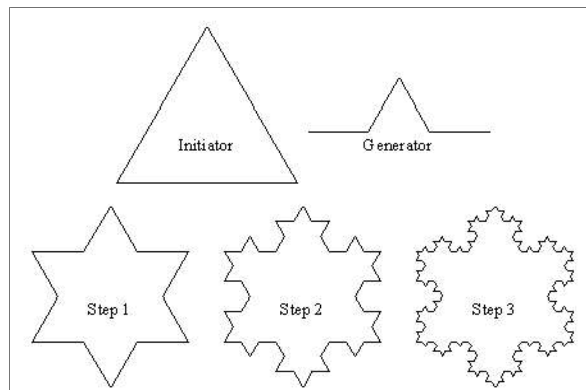


Figure 2.1 : The Koch Snowflake

Prusinkiewicz and Lindenmayer (1990) discuss how Noam Chomsky (1957) included a lot of work on formal grammars in his book, 'Syntactic Structures'. Chomsky's work generated wide interest in rewriting systems and it was as a new type of rewriting system that Lindenmayer introduced L-systems. A major difference between L-systems and Chomsky grammars is that L-systems apply productions in parallel whereas Chomsky grammars apply productions sequentially. This means that in L-systems all the letters of a given word are replaced simultaneously. The biological motivation behind L-systems is the reason for this as the productions are intended to model cell divisions.

2.1.2 Formal Definition of DOL-systems

The simplest class of L-system are deterministic and context free L-systems, called DOL-systems. Below is the formal definition for DOL-Systems given by Prusinkiewicz and Lindenmayer (1990):

> Let V denote an alphabet, let V^* be the set of all words over V, and V^+ be the set of all non-empty words over V. A *string OL-system* is an ordered triple: $G = (V, \omega, P)$
>
> where V is the *alphabet* of the system,
>
> $\omega \in V^+$ is a non-empty word called the *axiom*.
>
> and $P \subset V \times V^*$ is a finite *set of productions*.
>
> A production $(a, \chi) \in P$ is written as $a \rightarrow \chi$
>
> where a is called the *predecessor* of the production,
>
> and χ is called the *successor* of the production.
>
> It is assumed that for any letter $a \in V$, there is at least one word $\chi \in V^*$ such that $a \rightarrow \chi$.
>
> If no production is explicitly specified for a given predecessor $a \in V$, the *identity production,* $a \rightarrow a$ is assumed to belong to the set of productions P.
>
> An OL-system is *deterministic* (denoted *DOL-system*) if and only if for each $a \in V$ there is exactly one $\chi \in V^*$ such that $a \rightarrow \chi$.

The following example will illustrate the idea of an L-system. Let the alphabet be $V = \{x, y\}$, so all strings are built using only the letters x and y. Let the set of productions P, that operate on these letters, be $x \rightarrow y$ and $y \rightarrow xy$, this means that an occurrence of the letter x is replaced by the single letter y and an occurrence of the letter y is replaced by the string xy. Now let the axiom be $\omega = x$, this is the string with which we begin our rewriting process. In the first iteration the letter x is replaced by the letter y. In the second iteration the letter y is replaced by the string xy. In the third iteration the letter x of the string xy will be replaced by the letter y and the letter y will be replaced by the string xy giving the string yxy. This process continues as shown below:

x
y
xy
yxy
xyyxy
yxyxyyxy
xyyxyyxyxyyxy
.
.

Notice that during this process all characters of each string are simultaneously replaced at each step.

2.1.3 Geometric Interpretation

Initially L-systems were only conceived as a theoretical framework and weren't capable of describing the geometrical aspects of plant development. Consequently several ideas for geometric interpretations of L-systems were developed, including one based on turtle interpretation which is explained in the following couple of sections.

2.1.3.1 2-Dimensional Turtle Interpretation

Prusinkiewicz and Lindenmayer (1990) define the basic idea of turtle interpretation as follows:

> A *state* of the *turtle* is defined as a triple (x, y, α)
> where (x, y) are Cartesian coordinates representing the turtle's *position*.
> and α is an angle, called the *heading*, and is interpreted as the direction in which the turtle is facing.
> Given the *step size* d and the *angle increment* δ , the turtle can respond to commands represented by the following symbols:
>
> F Move forward a step length d.
> The state of the turtle changes to (x', y', α)
> where $x' = x + d \cos \alpha$
> $y' = y + d \sin \alpha$
> A line is drawn between (x, y) and (x', y').
>
> \+ Turn left by angle δ.
> The next state of the turtle is $(x, y, \alpha + \delta)$. The positive orientation of angles is counter-clockwise.
>
> \- Turn right by an angle δ.
> The next state of the turtle is $(x, y, \alpha - \delta)$.

The idea of turtle interpretation can be demonstrated by reconsidering the Koch Snowflake from figure 2.1. This can be generated using turtle interpretation. The axiom would be ω = F- -F- -F, and

the angle increment $\delta = 60°$. This axiom describes a shape equivalent to the initiator of the earlier example. The production in this case would be F → F-F++F-F, which is equivalent to the generator. The results of the first two iterations of this L-system are shown below in figure 2.2.

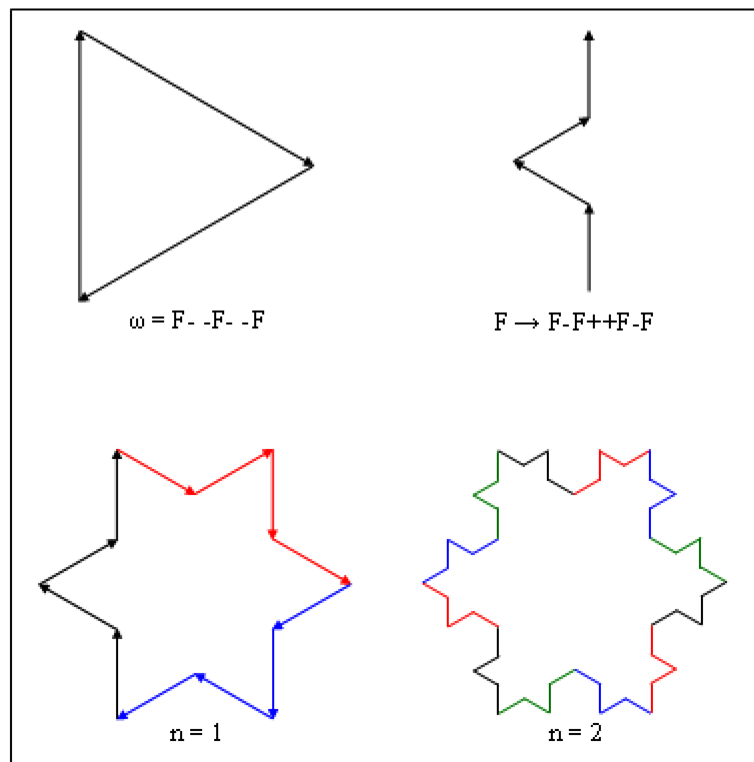


Figure 2.2 : Turtle Interpretation

The turtle interpretation seen so far is capable only of producing a single continuous line. This line could double back on itself any number of times and be as complicated as you wanted, but it would always just be a single line. However, in nature plants and other cellular organisms are frequently branching structures and so there is an extension to turtle interpretation to allow for the representation of these. Prusinkiewicz and Lindenmayer (1990) introduce two new symbols that allow for branching:

[	Pushes the turtle's current state onto a stack.
]	Pops a state from the stack and sets this as the current state for the turtle.

You can think of the '[' operation as asking the turtle to remember its state at that given point and then the ']' operation is used to recall the most recently stored state.

The use of brackets can be demonstrated with an example. Consider the string F[+F]F[-F]F, this can be interpreted as follows: draw a forward line then remember this position, then rotate anti-clockwise and draw a line, now go back to the remembered state and draw another forward line now remember this state and then rotate clockwise and draw a forward line then go back to the remembered state and

finally draw another forward line. This is illustrated in figure 2.3.

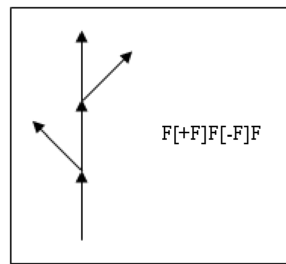


Figure 2.3 : Branching Example

2.1.3.2 3-Dimensional Turtle Interpretation

Prusinkiewicz and Lindenmayer (1990) also define a 3-dimensional turtle interpretation that uses three vectors to represent the turtle's current orientation in space. These three vectors indicate the turtle's heading, its direction to the left and its direction up. Rotations about these vectors are then represented by three rotation matrices.

The program created in this project could be extended to cater for 3-dimensional as well as 2-dimensional L-systems.

2.1.4 Stochastic L-systems

Prusinkiewicz and Lindenmayer (1990) also discuss stochastic L-systems which introduce variations between images of the same L-system. This is achieved by assigning probabilities to a set of productions that all operate on the same character. The effect of this when generating images of plants is designed "to introduce specimen-to-specimen variations that will preserve the general aspects of a plant but will modify its details."

Again the program created in this project could be extended to deal with stochastic L-systems.

2.2 Research Into Currently Available Software

This section will investigate what is currently available in the way of L-system image generators and evaluate some of those found.

2.2.1 Web-based Software

There are several systems that are available over the over the World Wide Web. Two particular Java applets that performed some of the tasks that this project shall address are *Milan's L-System Generator* by Milan Verma (2004), from the Department of Computer Science at Queen Mary, University of London, and one from a website called *JavaView* (2004). Screen shots of these two applets can be found in Appendix C.

The two systems both had a similar general layout style, with the system area split vertically into two sections, one for the displayed image, and one for the user to input and modify the parameters of the L-system image to be generated. The parameters that could be changed were largely the same in both systems, these were: the axiom, the replacement rules, the number of iterations and the turning angle.

Milan's L-System Generator also had the following nice features:

- a drop down menu so the user could load several pre-set example L-systems.
- a drop down menu so the user could select the line colour to be used in the image.
- an input field for the user to specify the starting angle for the L-system.

The system from the *JavaView* website also had several nice features including:

- a check button to choose whether the image was automatically re-sized to fit the display area.
- an area displaying all of the characters of the alphabet being used.
- sliders for the user to alter the number of iterations and the angle.
- a check button which enables the displaying of the 'Current State' of the L-system string.

Both of the systems also had some faults or areas that could be improved. *Milan's L-System Generator* just had one minor issue to mention:

- although the user was able to load pre-set L-systems, once the user had done so, the string representing that L-system remained displayed in the top drop down list, even if the user themselves had changed any or all of the details of this L-system.

The system on the *JavaView* website also had a few problems:

- it didn't have a button for the user to update the displayed image, although pressing 'return' on the keyboard performed this task. However there was a 'Reset' button that replaced whatever information the user had changed with the systems default settings - which could be frustrating if pressed in error.
- it offered the user the ability to define replacement rules for all the characters of the alphabet including the operations + , - ,] and [. Which in most cases will be unnecessary and could lead to confusion.
- the images produced by this system had small red circles at end of each straight segment which can obscure the desired image if there are too many of them.

There was also one issue which affected both of these systems; if the number of iterations was raised too high the system appeared to freeze. This is because the computations involved become enormous as the complexity of the process of rewriting the strings is exponential. This causes problems if the user raises the number of iterations too high either by accident or intentionally. A possible solution to this problem would be for the system to issue a warning to the user if the size of the calculation

was likely to be above a certain size.

2.2.2 Fractint

Another piece of available software is a system called Fractint. Fractint is a free ware fractal generator produced by *The Stone Soup Group* and it is available from the *Fractint Homepage* (2004). It is a DOS based piece of software used to produce Fractal images, and can also be used to produce images of L-systems.

This system is not very visually 'attractive' and it is not a particularly user friendly or intuitive piece of software. This is mainly because the system runs under DOS which may be unfamiliar to many users, who are more likely to be comfortable with a window based graphical user interface. It is, however, a very powerful and versatile system. It can produce images of L-systems with a large number of iterations much faster than the two Java applets I looked at. Also, when the system is drawing an L-system that takes some computational time to produce it displays the following message to the user, so that they know that the system has not frozen : *L-system thinking (higher orders take longer)*.

To produce images of L-systems you needed to create a file containing the relevant L-system information (such as the angle, axiom and productions) and then use this file to produce the image. Fractint displays its images using the full screen. Once the image is displayed you cannot change the parameters of the L-system while the image remains on the screen because you need to go back to the file and update the information and then ask the program to redraw the L-system. If the program is to be used as an educational or research tool it would be desirable for the image and parameters to be in view at the same time so the user can clearly see the effects of any changes made.

2.3 Quality of Resources

The definitions seen in section 2.1 came from a book produced by Prusinkiewicz and Lindenmayer (1990), this is a very reliable source of information about L-systems seeing as it was Aristid Lindenmayer himself who developed the theory of L-systems.

The two Java applets seen in section 2.2 are not professional commercial pieces of software, they are projects undertaken by interested amateur programmers. As such, they are bound to have some faults and limitations. The program created in this project is intended to improve upon these faults and limitations. Again, Fractint is not a commercial piece of software as it is distributed free of charge. It did, however, have the feel of a much more professional and powerful piece of software than the two applets.

3 Design

3.1 Overview

Having taken into consideration the software reviewed in the previous chapter the program developed in this project will follow the same basic layout as the Java applets, where the display is split roughly into two sections, one for the image and another for the input of parameters. This is to enable the user to view the effects of any changes to the L-system's parameters as they are made. The good and bad features of the systems discussed in the previous chapter were taken into consideration.

3.2 System Requirements

The primary function of the program is clearly to produce an image of an L-system given the relevant parameters that describe it. The parameters required to define an L-system are an axiom, a set of production rules, the number of iterations and, to define the turtle interpretation, the angle increment by which to turn as well as the starting heading of the turtle. The program will therefore need to have features that allow the user to input and adjust all of this data, and then draw the required image at the user's request – by clicking a 'draw' button.

These are the minimum features that the program would need, in addition to these there are also several other attributes, some of which were available in the systems reviewed in section 2.2, that would also be useful. These are described as follows:

The program should:

- have the ability to change both the line and background colours that are used in the image.
- be able to load the parameters for various 'pre-set' L-systems and view the images produced.
- be able to save the produced diagram as an image file (for example JPG or PNG formats), so the image could be used in various applications.
- be able to change the size of the image created to enable use in a variety of situations.
- be able to size the image to fill the available space in the window.
- have a 'clear' button that removes all input from the input fields and clears the image from the screen, so the user can begin inputting new information easily.
- have a 'reset' button that reverts all input fields to the values of whichever pre-set L-system was most recently selected.

3.3 The Graphical User Interface

The next problem was to decide on the general look of the graphical user interface, or GUI. As stated previously the program's GUI will be split into two sections, one that contains all of the areas for entering and altering data such as the axiom, production rules etc. and another that displays the

generated image. The display will be split vertically with the data area on the left side and the displayed image on the right hand side, as illustrated in figure 3.1.

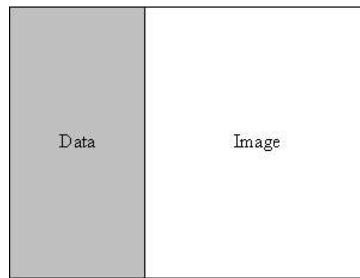


Figure 3.1 : GUI Layout

The exact layout of the various data fields now needed to be decided. The most important fields are the input fields for the axiom, production rules, turn & starting angles and the number of iterations. These should be positioned towards the top of the data fields area. These are made up of a label (e.g. Axiom:) followed by an area for the user to input their desired value. Beneath these will be a selection of buttons. The draw button should be the most prominent of these as it is likely to be the most frequently used. Finally the data fields area will also have some drop down lists; one for selecting a pre-set L-system, and one each for selecting the background colour and line colour of the image. These features will combine to produce a GUI with a layout similar to that seen in figure 3.2.

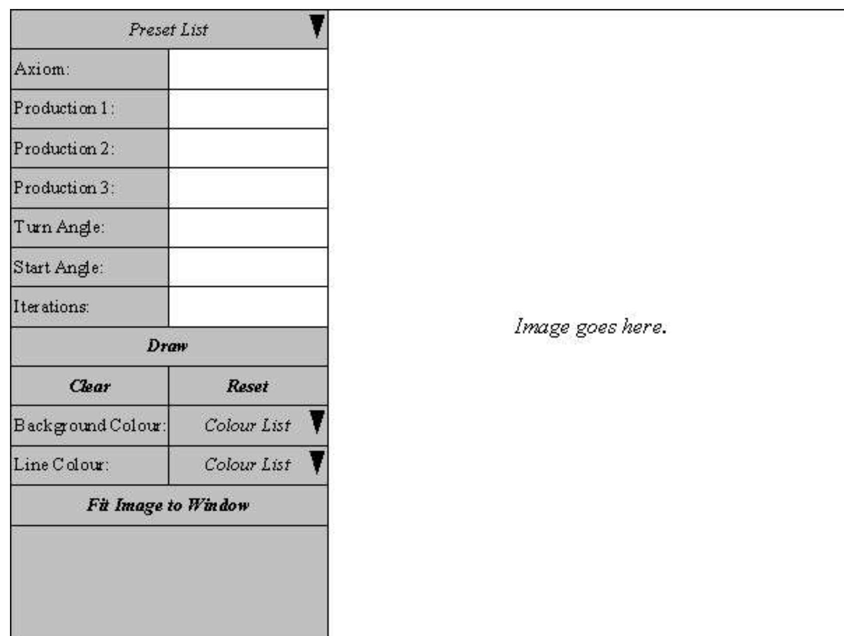


Figure 3.2 : GUI Layout

3.4 Proposed Method

The idea behind the program is to first collect the information from the GUI, then apply the production rules to the axiom to produce an updated string. It will then use an implementation of turtle interpretation to turn this string into a set of coordinates, which will then be connected as

necessary to produce an image representing the L-system.

3.5 Programming Language

Choosing the programming language to use in this project was an important decision. During my first two years of study I completed modules covering object oriented programming, specifically using C++ and Java, so it was sensible for me to take an object oriented approach using either of these languages if they were suitable to the task. Textbooks on programming in both C++, Deitel and Deitel (2001), and Java, Barker (2002), were consulted.

Java and C++ have many similarities and I felt that either would be suitable for the requirements of this project, so particular features of both need to be considered. Barker (2002) explains how Java has an extensive selection of application programming interfaces, or API's , which provide a platform independent way of accessing operating system functions including, and of particular benefit to this project, GUI rendering. One such API is the Java Swing API that provides a comprehensive set of GUI components that, to a large degree, work the same way on all platforms. Java code itself is also platform independent, whereas compiled C++ code will only run on the platform on which it was compiled. Platform independence is a desirable feature in any program so that it may be run on Windows machines, Unix machines, Macintosh's etc. and Barker (2002) also describes how if these Java API's are used in developing code then “the resultant Java code is truly portable”. Because of its platform independence and GUI packages Java was used throughout this project.

3.5.1 The Java Swing API

The Java Swing API contains GUI components that have a Java-specific look no matter what platform they are being used on. The documentation for this package (as well as for all Java packages) can be found on the Java API Specification website from Sun Microsystems (2005). The components in this API have names of the form J*ComponentName* (for example JTextField, JButton, JFrame etc.) and these are used throughout the development of the GUI in this project.

3.6 UML Outline of Proposed System

As an object oriented approach was to be taken to the problem, the program would be made up of several classes that perform various tasks. There needed to be a class that deals with all of the GUI aspects of the program, this class is called Display and it extends the JFrame class from the Java Swing API. The Display class handles all of the components of the GUI, for example the buttons and input fields, as well as all of the user interaction with it. There is a separate class that actually deals with drawing the diagram, this is called LSystemDiagram and it extends JPanel, also from the Java Swing API. The Display class then contains an instance of the LSystemDiagram class as one of its components to display the image.

There will be two classes that enable the processing of input by turning the information entered by the user into a set of coordinates, these are LSystem and Turtle. The LSystem class has all the attributes of an L-system, these being strings representing the axiom and production rules, doubles for the start and turn angles and an integer representing the number of iterations to be calculated. The LSystem class takes all of this information and first performs the string rewriting and then, using the Turtle class, turns the produced string into a list of coordinates representing the image to be drawn. The Turtle class has all of the attributes of a turtle, these being integers representing its current x and y position, doubles to represent its current angle (or heading) and turn angle (angle increment) and a stack to store states (a state being the turtles x and y position and heading). It also performs all the actions seen in section 2.1.3.1: move forward, rotate, and push and pop states onto the stack. The LSystem class has a Turtle as an attribute that it uses to turn the rewritten string into a list of coordinates, which are then passed to the LSystemDiagram class to actually draw the image. Finally, there is a class called LSViewer which acts as the main program.

More details about how these classes are implemented follows in chapter 4. An outline UML diagram of the proposed system is shown in figure 3.3. A full UML diagram for the finished system can be found in Appendix B.

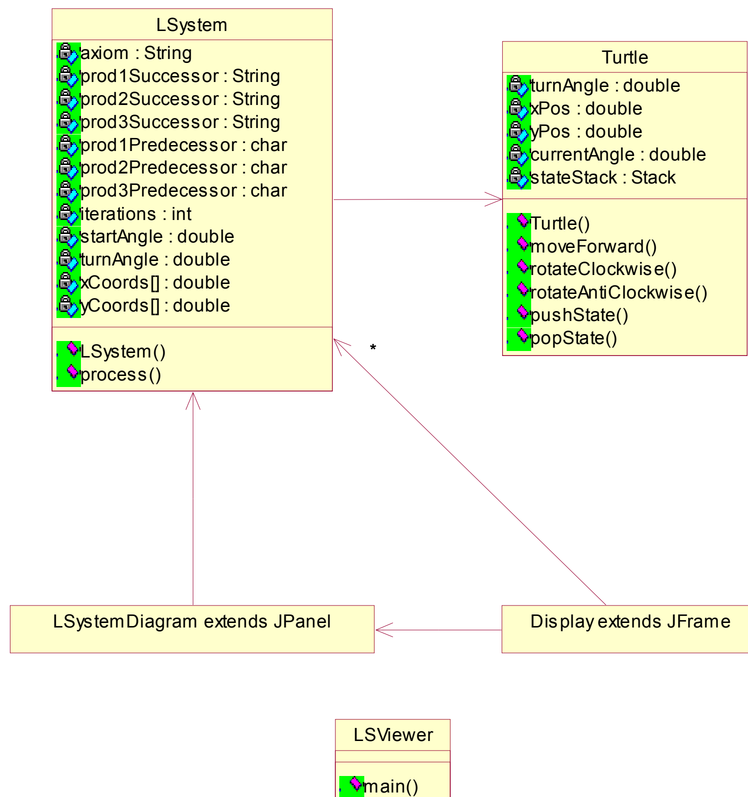


Figure 3.3 : UML Outline

4 Implementation

4.1 Overview

This chapter looks at how the program performs each task in more detail. It doesn't go into the detail of how every single process of the program works, but it looks at some of the more important ideas and implementation details. It was discussed generally in chapter 3 how the program would go about turning the user input into a diagram, by first applying the production rules to the axiom, then using turtle interpretation to turn this into a set of coordinates and then finally draw the diagram by connecting the relevant points with straight lines. This would be implemented in Java with classes called Display, LSystem, Turtle, LSystemDiagram and LSViewer.

The implementation process can be broken down into several smaller tasks. One of which is producing the GUI, and the others are to firstly process the input to produce a set of coordinates for the image and secondly to actually draw the diagram given this set of coordinates.

4.2 L-system Alphabet

The L-system alphabet used should include all of the 2-dimensional turtle commands seen in section 2.1.3.1. It is also helpful to have more than one character that defines the 'move forward' command so that different types of line can be drawn by defining different production rules for each one. For example many of the examples in Prusinkiewicz and Lindenmayer (1990) have 'left' and 'right' edges in them. As well as 'F', the alphabet will also contain 'L', 'R', 'X' and 'Y' all representing the 'move forward' command. The complete alphabet will be : F, L, R, X, Y, +, -,] and [.

4.3 Input Processing

The first problem considered was to process the input, regardless of how this input was gathered and without trying to draw any diagrams. This was so that the accuracy of the input processing could be checked before attempting the more complicated tasks of implementing a fully functioning GUI and drawing the diagram itself.

4.3.1 LSystem.java

Given the relevant information needed to describe an L-system (an axiom, a set of production rules, a turn angle and a number of iterations) the first step taken in processing this input is to apply the production rules to the axiom.

This is implemented in LSystem.java by the operation called process( ) which stores the updated string to a variable called outputString at the end of each iteration. It starts by copying the axiom to outputString, if the number of iterations is zero then it is finished, however if the number of iterations

is greater than zero then it must apply the production rules. It examines each character of outputString in turn – if a character matches one of the production predecessors then the corresponding production successor is added to a temporary variable called tempOutput, if the character examined is not matched by a production predecessor (but this character is still in the L-system alphabet) then this character itself is added to tempOutput (this is the identity production seen in section 2.1.2). After every character of outputString has been examined the content of tempOutput is copied to outputString and tempOutput is cleared, then if there are more iterations to perform the process is repeated.

This is implemented using a pair of nested for loops – the outer for loop repeats for the number of iterations, and the inner for loop is used to examine each character of outputString in turn. This is shown in the code in figure 4.1.

```
String outputString = axiom;
for (int j=0; j < iterations; j++) {
    for (int i=0; i < outputString.length(); i++) {
        if (outputString.charAt(i) == prod1Predecessor) {
            tempOutput = tempOutput + prod1Successor;
        }
        else if(outputString.charAt(i) == prod2Predecessor) {
            tempOutput = tempOutput + prod2Successor;
        }
        else if(outputString.charAt(i) == prod3Predecessor) {
            tempOutput = tempOutput + prod3Successor;
        }
        else if (outputString.charAt(i) == 'F', 'L', 'R', 'X', 'Y',
                                          '+', '-', '[' or ']') {
            tempOutput = tempOutput + outputString.charAt(i);
        }
        else {
            error with input...
        }
    }
    outputString = tempOutput;
    tempOutput = "";
}
```

Figure 4.1 : Code Excerpt from LSystem.java

4.3.2 Turtle.java

The next step after generating the new string, is to interpret its meaning using turtle interpretation. The Turtle class implements all of the turtle operations seen in section 2.1.3.1, these operations are called moveForward( ), rotateClockwise( ), rotateAnticlockwise( ), pushState( ) and popState( ). The state of a turtle is represented by an array of 3 doubles – one each for its x and y position and one for its heading. The implementation of these operations can be seen in figure 4.2.

The LSystem class uses this Turtle class to interpret outputString and produce a set of coordinates. It does this by examining each character of outputString in turn and calling the relevant turtle commands for each character found.

```
public void moveForward() {
    xPos = xPos + Math.cos(Math.toRadians(currentAngle));
    yPos = yPos + Math.sin(Math.toRadians(currentAngle));
}

public void rotateClockwise() {
    currentAngle = currentAngle - turnAngle;
}

public void rotateAntiClockwise() {
    currentAngle = currentAngle + turnAngle;
}

public void pushState() {
    double[] state = new double[3];
    state[0] = xPos;
    state[1] = yPos;
    state[2] = currentAngle;
    stateStack.push(state);
}

public void popState() {
    double[] state = new double[3];
    state = (double[]) stateStack.pop();
    xPos = state[0];
    yPos = state[1];
    currentAngle = state[2];
}
```

Figure 4.2 : Code Excerpt from Turtle.java

Two of these commands, moveForward( ) and popState( ), change the turtle's position, so after calling either of these methods the new position of the turtle needs to be stored. LSystem has two arrays of doubles, called xCoords and yCoords, which are used to store the x and y coordinates of the L-system. After calling either moveForward( ) or popState( ) the position of the turtle is stored in these arrays. The popState( ) command indicates the end of a branch segment, which is marked in the coordinate arrays by placing infinity in each one, then calling popState( ) and storing the new positions. This will be explained in section 4.4. After each character of outputString has been examined there will be a full set of coordinates stored in the two arrays.

This process is implemented using a for loop to examine each character of outputString and if/else statements to perform the relevant task. As the coordinates are being produced, LSystem keeps a record of the largest and smallest values for both the x and y coordinates, this information will be used for scaling and positioning the image, which is discussed later in section 4.4.1. This implementation is shown in figure 4.3.

```
turtle = new Turtle(0.0, 0.0, turnAngle, startAngle);
xCoords[0] = yCoords[0] = 0.0;
xMax = xMin = yMax = yMin = 0.0;
numCoords = 1;

for(int i = 0; i < outputString.length(); i++) {
    if (outputString.charAt(i) == 'F', 'L', 'R', 'X' or 'Y') {
        turtle.moveForward();
        xCoords[numCoords] = turtle.getXPos();
        yCoords[numCoords] = turtle.getYPos();
        numCoords++;

        // update max/min info
        if (turtle.getXPos() > xMax) xMax = turtle.getXPos();
        else if (turtle.getXPos() < xMin) xMin = turtle.getXPos();
        if (turtle.getYPos() > yMax) yMax = turtle.getYPos();
        else if (turtle.getYPos() < yMin) yMin = turtle.getYPos();
    }

    else if (outputString.charAt(i) == '-') {
        turtle.rotateClockwise();
    }

    else if (outputString.charAt(i) == '+') {
        turtle.rotateAntiClockwise();
    }

    else if (outputString.charAt(i) == '[') {
        turtle.pushState();
    }

    else if (outputString.charAt(i) == ']') {
        // to note the end of the branch
        xCoords[numCoords] = java.lang.Double.POSITIVE_INFINITY;
        yCoords[numCoords] = java.lang.Double.POSITIVE_INFINITY;
        numCoords++;

        turtle.popState();
        xCoords[numCoords] = turtle.getXPos();
        yCoords[numCoords] = turtle.getYPos();
        numCoords++;
    }
    else {
        invalid character...
    }
}
```

Figure 4.3 : Code Excerpt from LSystem.java

4.4 Drawing the Image – LSystemDiagram.java

Once there is a full set of coordinates these are used to draw a 2-dimensional diagram. As discussed earlier the program has a class called LSystemDiagram which extends JPanel, which is the component of the GUI that displays the diagram. This class has an instance of LSystem as one of its attributes which contains the set of coordinates needed to draw the image. The constructor for LSystemDiagram is also given a dimension and two colour objects, one for the background colour and one for the line colour.

The idea behind drawing the diagram is fairly simple – given the list of coordinates a straight line is drawn from each point to the next. However, this only works if the L-system is simple (not branching), if it is a branching L-system then you don't always want a line from one point to its neighbour. When the end of a branch is reached you need to go back to the last point before the branch and then draw a new line from there.

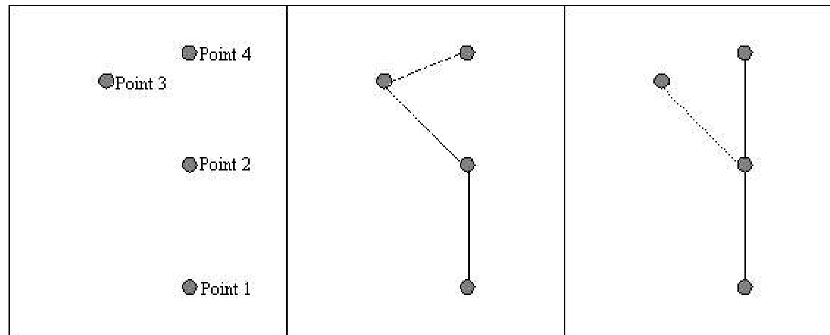


Figure 4.4 (a) (b) (c) : Branching Example

Consider the example of drawing the L-system defined by the string F[+F]F, this should look something like the image in figure 4.4(c). If the coordinate arrays for this L-system were a list of the four points in the order p1, p2, p3, p4 and these were simply joined by lines we would end up with something looking like figure 4.4(b). This problem can be solved by placing some sort of marker in the array of coordinates to label the end of each branching segment.

The coordinates are stored in an array of doubles so the marker must also be some kind of double object, but one that wouldn't normally occur as a coordinate. Java has a double constant that represents infinity which can be used because infinity would never be generated as a normal coordinate. When the arrays of coordinates are being produced by LSystem, infinity is placed in the arrays every time before popState( ) is called and then the new coordinates are placed in the arrays. The implementation of this was seen in section 4.3.2, in figure 4.3.

When processing the above example of F[+F]F, instead of just producing a list of coordinates p1, p2, p3, p4 and connecting these as in figure 4.4(b), the program will produce the following arrays:

xCoords = [$p1_x$, $p2_x$, $p3_x$, ∞, $p2_x$, $p4_x$]   yCoords = [$p1_y$, $p2_y$, $p3_y$, ∞, $p2_y$, $p4_y$]

```
for (int i = 0; i < (numCoords - 1); i++) {

    double inf = java.lang.Double.POSITIVE_INFINITY;

    while (xCoords[i + 1] == inf | xCoords[i] == inf) {
        i++;
    }

    double x1 = xCoords[i];
    double x2 = xCoords[i + 1];
    double y1 = yCoords[i];
    double y2 = yCoords[i + 1];

    ...

    LineSegment thisLine = new LineSegment(x1, y1, x2, y2, lineColor);
    thisLine.draw(g);
}
```

Figure 4.5 : Code Excerpt from LSystemDiagram.java

Instead of simply connecting each point to its neighbour, the program skips over each occurrence of infinity and draws a line between the next pair of points. The arrays above will produce a line from

p1 to p2, then p2 to p3, then skip over infinity and draw a line from p2 to p4 as shown in figure 4.4 (c). The implementation of this is shown in figure 4.5. LineSegment in this code is a simple class that is given 4 doubles representing the start and end point of the line as well as a colour. Calling LineSegment's draw method will then draw a straight line in the specified colour between the two points.

4.4.1 Scaling and Positioning the Image

The positions of the coordinates produced will be correct relative to each other but not relative to the display space. The only definite positioning of any of these coordinates is that the first one is at the point (0.0, 0.0). The diagram needs to be centred and also scaled to best fill this area – the size of the area can then be adjusted to resize the diagram.

To centre the diagram we need to know the location of the four extremes of the diagram: the largest and smallest x and y coordinates. These are stored during the production of the coordinates. This was seen in the code section 4.3.2, figure 4.3.

The centre of the diagram needs to be placed at the centre of the display area. The centre of the diagram is calculated using its maximum and minimum x and y coordinates, the difference between this and the centre of the display area is then calculated and each coordinate is adjusted by this value.

Once the diagram is centred, it is scaled. The largest dimension of the diagram needs to be calculated first, as this is the one that will affect the scaling. Then each coordinate is translated to the origin, scaled and then translated back to the centre of the display area.

This process is shown in figure 4.6 which shows the additions made to the code in figure 4.5. Figure 4.7 is used to illustrate what each dimension relates to. The longEdge value in the code is simply the largest value out of imageWidth and imageHeight and the value 10 is subtracted to provide a small border (of 10 pixels) around the image.

```
for (int i = 0; i < (numCoords - 1); i++) {

    double x1 = xCoords[i] + xAdjust;
    double x2 = xCoords[i + 1] + xAdjust;
    double y1 = yCoords[i] + yAdjust;
    double y2 = yCoords[i + 1] + yAdjust;

    // shift to origin, then scale, then shift back
    x1 = ((x1-halfDWidth) * ((dWidth-10) / longEdge)) + halfDWidth;
    x2 = ((x2-halfDWidth) * ((dWidth-10) / longEdge)) + halfDWidth;
    y1 = ((y1-halfDHeight) * ((dHeight-10) / longEdge)) + halfDHeight;
    y2 = ((y2-halfDHeight) * ((dHeight-10) / longEdge)) + halfDHeight;

    LineSegment thisLine = new LineSegment(x1, y1, x2, y2, lineColor);
    thisLine.draw(g);
}
```

Figure 4.6 : Code Excerpt from LSystemDiagram.java

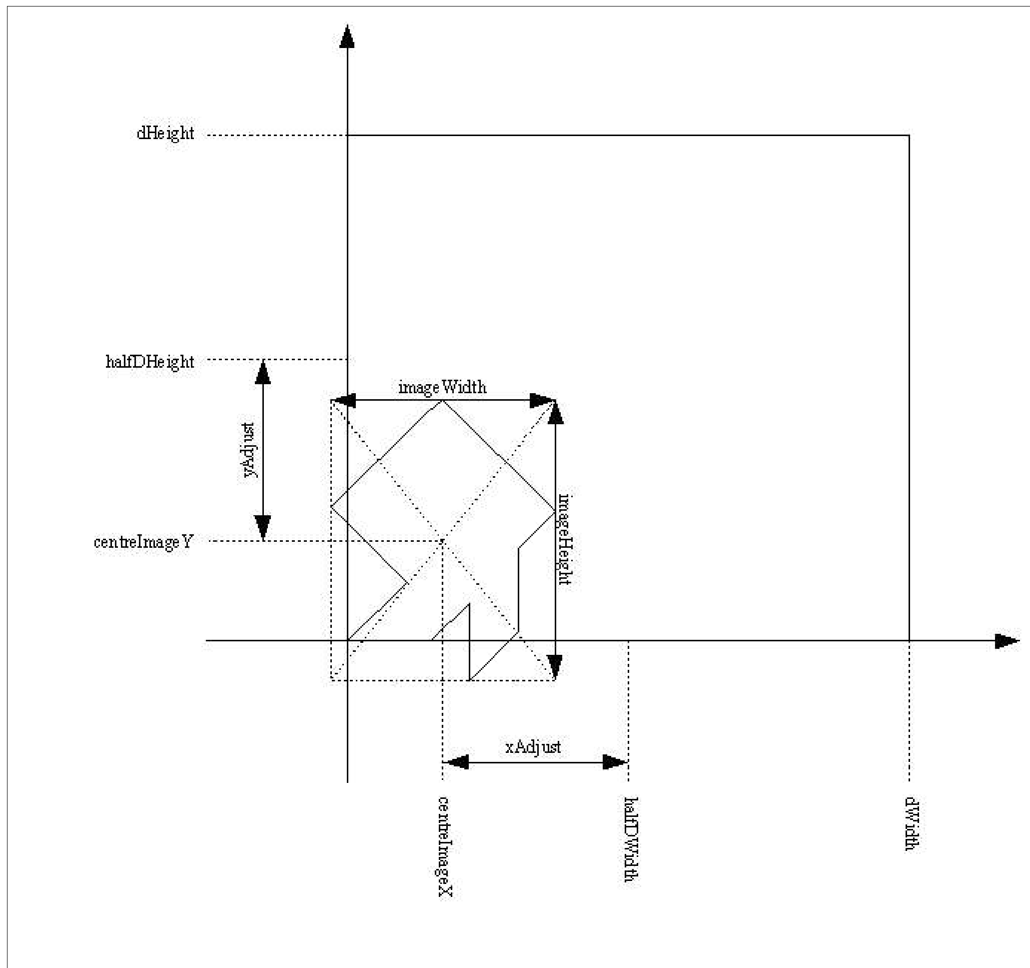


Figure 4.7 : Diagram Positioning

4.5 The Graphical User Interface – Display.java

The graphical user interface (GUI) is implemented in Display.java. This not only displays the produced image, but it is responsible for handling all user interaction with the program.

Barker (2002) states that the fundamental approach to programming GUIs is to assemble graphical building blocks called components in specific ways to get the desired 'look' and then program their logic to behave in a way that we want. We have already seen much of the behind the scenes logic that enables the production of an image from a given input, the GUI is then responsible for gathering this input and using this to draw the image.

As discussed in section 3.5.1, components from the Java Swing API are used to create the majority of the GUI. A container is a type of component that can be used to contain and organise other components. In this program contentPane is the outermost container and has two more containers added to it. These two containers represent the splitting of the display into the data input and diagram display area. A JPanel called infoPanel will contain all of the components for inputting and

amending the data. The other component is a JScrollPane called drawScrollPane. JScrollPanes are containers that can have scroll bars along their edges when their contents are larger than the JScrollPane itself. This allows the user to scroll around the image if its larger than the display area.

The drawScrollPane then has a JPanel called drawPanel added to it. The drawPanel is the container that will actually hold the instance of LSystemDiagram that displays the image itself. This arrangement is shown in figure 4.8.

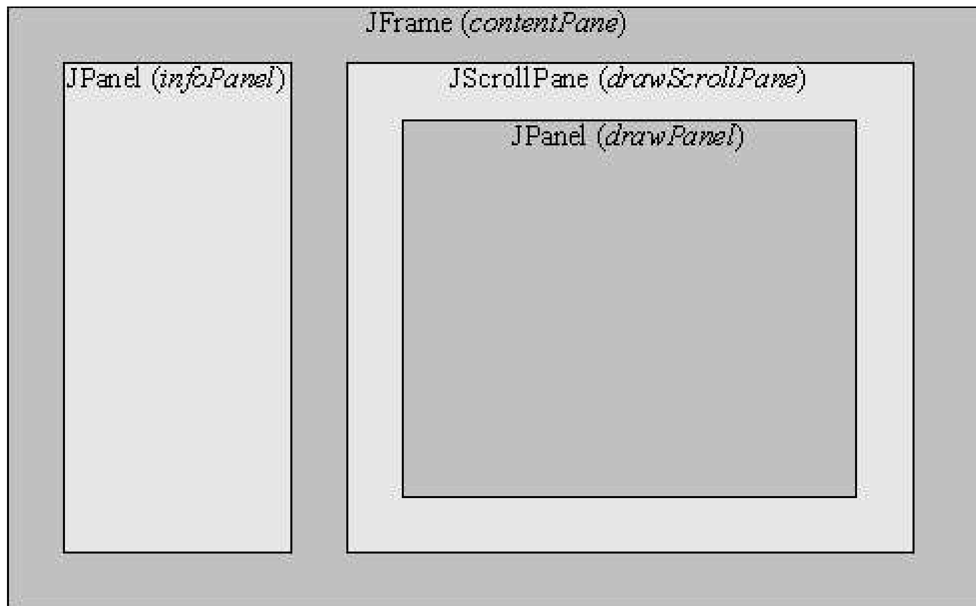


Figure 4.8 : GUI Structure

4.5.1 Layout Arrangement

Containers have layout managers that arrange the component objects contained within them. The contentPane uses a BorderLayout manager which divides the space into 5 regions: north, south, east, west and center. The infoPanel is added to the west region of contentPane and the drawScrollPane is added to the center region. The drawScrollPane is added to the center and not the east region because the center region will take up any remaining space in the contentPane.

Contained within infoPanel are all of the components that allow the user to input and amend details of the L-system. These include JTextFields for inputting text such as the axiom and production rules, JLabels to label the JTextFields, JButtons for the draw, clear and reset buttons, JComboBoxes for choosing the colours and pre-set L-systems, JSpinners for the angles and number of iterations and a JTextArea that displays the contents of outputString. In order to arrange this in a grid like formation infoPanel uses a layout manager called GridBagLayout. The result of this layout can be seen in the screen shot in figure 4.9.

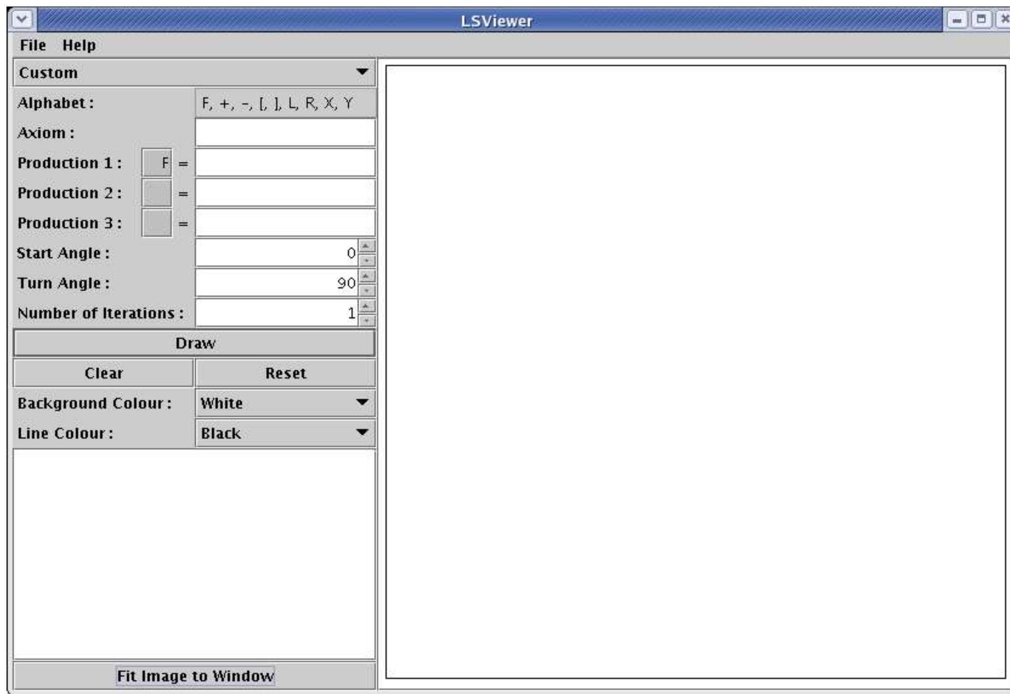


Figure 4.9 : GUI Screen shot

4.5.2 Adding Actions

Components such as the buttons, colour lists and pre-set list, that are required to 'do' things have actionListeners attached to them. ActionListeners are used to control what happens when a particular component is used, for example, figure 4.10 shows an actionListener added to the drawButton, which calls the drawDiagram( ) operation of Display whenever the draw button is clicked.

```
drawButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        drawDiagram();
    }
});
```

Figure 4.10 : Code Excerpt from Display.java

4.5.2.1 Drawing the Diagram

The Display class has an instance of the LSystem class called currentLSystem as an attribute, which stores all the information about the L-system to be drawn. The drawDiagram( ) operation of Display then gathers all of the data that is contained in the various text fields and spinners and updates the currentLSystem with these attributes. It then calls the process( ) operation of LSystem to generate coordinates and creates an instance of LSystemDiagram with this currentLSystem as a variable. LSystemDiagram also has a dimension, and two colours as variables which have also been acquired by the GUI. This instance of LSystemDiagram, which is called diagramPanel, is then added to the drawPanel which is contained within the drawScrollPane (see figure 4.8).

The drawDiagram( ) method is reused whenever the system needs to redraw the L-system image. For instance whenever the draw button is pressed, the 'return' key on the keyboard is pressed, the reset button is pressed, the clear button is pressed, a colour (line or background) is selected, a pre-set L-system is selected or the image is resized.

4.5.2.2 Loading Pre-set L-systems

As well as currentLSystem, Display also has several more instances of LSystem as attributes which have names of the form preset*LSystemName*, for example presetTree1 and presetKochSF. These instances of LSystem are created with the relevant L-system parameters for various pre-set L-systems that can be 'loaded'. The names of all of these L-systems appear in the pre-set list which has an attached actionListener that changes the contents of currentLSystem to whichever one of these L-systems was chosen before calling the drawDiagram( ) method to redraw the diagram of the selected L-system.

The 'clear' button also uses a pre-set L-system, called presetCustom, to clear all the data and diagram. This is an instance of LSystem with an empty string as the axiom, 1 as the number of iterations, 0.0 as the start angle, 90.0 as the turn angle and no production rules. When the 'clear' button is pressed the program changes the currentLSystem to presetCustom and redraws the image. Similarly the 'reset' button reloads whichever pre-set is currently selected and redraws the image.

4.5.3 The File Menu

The GUI has a file menu which is an instance of JMenu attached to a JMenuBar. The file menu contains a quit option, which exits the program, as well as an option to save the image and and an option to specify the size of the image.

4.5.3.1 Setting Image Size

When the user selects the option to set the image size, an input dialog box (figure 4.11) pops up asking the user to enter a size for the image in pixels, the image is then created as a square of this size. This value is given to diagramPanel and the image is redrawn. There is also a button on the GUI for resizing the image to best fit in the available space in the scroll pane. This determines the current smallest dimension of the scroll pane and sets the dimensions of the image to be slightly less than this before redrawing the image.

Figure 4.11 : Set Size Dialog

4.5.3.2 Saving Images to File

When the user selects the option to save the image a JFileChooser (another Java Swing component) pops up from which the user specifies a file location. The program then saves the image as either a JPEG or PNG image by creating a buffered image of the same dimensions as the displayed image and then 'painting' a copy of the L-system diagram onto this buffered image, and finally writing the image to the file specified by the user. The images found in Appendix D were created in this way.

4.6 Error Handling

In this program there are many opportunities for mistakes in the input to be made. The most obvious are with the input of the axiom and production successor strings, but also with the turn and start angles, number of iterations and production predecessors. Any faulty input needs to be clearly and graciously dealt with to avoid nonsense output or the program crashing.

The turn and start angles and iterations are easy to deal with since we know that both of these angles should be doubles in the range [-360.0, 360.0] and that the number of iterations should be an integer in the range [0, n], where n is some suitable upper bound to prevent unrealistically large computations (19 is used in this program). It is easy to impose restrictions on the input of these three values as they all use JSpinners to gather the data and JSpinners have the ability to restrict input to a particular data type and range.

The axiom, production successor and production predecessor strings should only contain characters from the specified alphabet (section 4.2). If they contain characters that are not in this alphabet then the program will issue an error message to the user via a pop-up window informing them that there is an invalid character (and which character this is) and will not then try to process this input. An additional check with the production predecessor is that it only consists of one single character and not a string, although it ignores leading and trailing spaces. Two examples of the error messages produced are shown in figures 4.12 and 4.13.

Figure 4.12

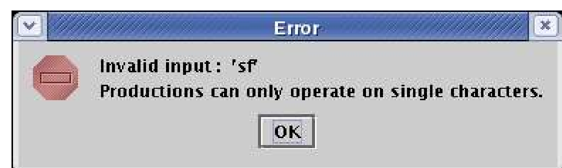


Figure 4.13

The program also catches various exceptions that may be thrown and displays a relevant error message. For example, an emptyStackException will be thrown if you attempt to pop an item from an empty stack, this may occur if the brackets used for branching don't match up correctly.

4.7 Reliability

It was seen in section 2.2.1 that a common problem with the two Java applets was that if the computation was too large then the programs would appear to freeze or crash. This was due to the number of calculations involved in L-systems being exponential. This program warns the user if the computation for a requested L-system is going to be large.

You can estimate the number of computations in a particular L-system by predicting the number of line segments involved in the image. In the case where there is just one production rule, let a = number of draw line commands in the axiom, p = number of draw line commands in the production successor, and i = number of iterations. An estimate for the number of line segments, e, is then given by the following formula:

$$e = a \cdot p^{i}$$

An upper bound on this estimate, for cases when there is more than one production rule, can be achieved with the following formula:

$$e = a \cdot p_{max}^{\ i}$$

Where p_{max} is the greatest number of 'move forward' commands in any of the production predecessors.

Before processing the L-system, the program calculates the value of e, using the above formula, and issues a warning (figure 4.14) to the user if this is above a certain value. The warning asks the user if they wish to continue drawing the image, if they choose to continue there is still the possibility that the program will fail to complete the computation, but the user gets the choice about whether to risk this or not. This also gives the user the chance to check that the input is correct before deciding whether to continue. The value for which it will offer a warning was calculated by trying to draw various L-systems, printing out the value of e each time, and then noting for which values of e the image was successfully drawn in a reasonable time. The figure decided upon was 30,000.

Figure 4.14 : Large Computation Warning Dialog

4.8 Problems Encountered

The implementation process on the whole was fairly problem free. However, the process was more time consuming than first anticipated. In particular creating a fully functioning and 'bug' free GUI took quite some time as well as the task of actually producing the 2-dimensional diagrams. Solving the problem of drawing branching L-systems also proved quite troublesome.

5 Evaluation

5.1 Evaluation Criteria

The program now needs to be evaluated to see how well it performs the tasks required of it. The program was intended as a tool for investigating L-systems and the following evaluation criteria are intended to assess its suitability as such. The criteria for evaluating the program are:

- to test if diagrams created by the program are accurate.
- to test how well the program copes with unreliable input.
- to see how good the program is at warning against large computations.
- to compare this program to the software evaluated in section 2.2.

5.2 Evaluating the Program

5.2.1 Image Output

To test the image output of the program the idea was to find examples of L-system images from a trusted source and compare the images produced by this program with those found. Prusinkiewicz and Lindenmayer (1990) contains many suitable such diagrams with the associated information: the number of iterations, the angle increment, the axiom and the production rules.

The drawing of a wide variety of L-systems was attempted so as to test all aspects of the program. There was a combination of branching and non-branching L-systems and some with just a single production rule and some with several. Specifically the images that were drawn came from Prusinkiewicz and Lindenmayer (1990) and were figures 1.7 (a) & (b), 1.9 (a), (b), (c), (d), (e) & (f), 1.10 (a) & (b), 1.11 (a) & (b) and 1.24 (a), (b), (c), (d), (e) & (f). The images produced by this program for these L-systems can be found in Appendix D. All of the images created were accurate recreations of the images found in Prusinkiewicz and Lindenmayer.

5.2.2 Handling Unreliable Input

The first aspect of unreliable input that was tested was the use of characters not in the L-system's alphabet. This was done by choosing a character not in the alphabet, 'a' for this test, and placing it once at a time in each of the following fields: the axiom, the production predecessors and the production successors. The rest of the input was left as a correct L-system with the following input:

Axiom: F
Production 1: F → FF-[-F+F+F]+[+F-F-F]
Production 2: BLANK
Production 3: BLANK

The result of clicking draw was noted, and the results are shown in table 5.1 below

Input Field	*Input*	*Result*
Axiom	Fa	Error message
Axiom	a	Error message
Production Predecessor 1	Fa	Error message
Production Predecessor 1	a	Error message
Production Predecessor 2	a	Error message
Production Predecessor 3	a	Error message
Production Successor 1	FF-[-F+F+aF]+[+F-F-F]	Error message
Production Successor 1	a	Error message
Production Successor 2	a	Image drawn correctly
Production Successor 3	a	Image drawn correctly

Table 5.1 : Input Error Results

The system noticed the error, cancelled drawing the image and issued an error message to the user in all but two of these cases. In the two cases where production successors 2 and 3 were given the input of 'a' no error message was issued and the image was drawn correctly. This was because these two productions had no predecessor defined so the contents of their successor was irrelevant and would have no impact on the image produced. Perhaps it would have been better if the program had warned the user that the input in these fields was faulty.

Another possibility for error with the input is with the use of brackets. For every 'pop state' command, indicated by a ']', there needs to be a corresponding 'push state' command, indicated by a '['. To test this, some examples of input that shouldn't work were tried in the axiom and production successor fields, with correct input everywhere else. The results are shown in table 5.2 below.

Input Field	*Input*	*Result*
Axiom	F[-F]F[+F]][F]	Error message
Production Successor 1	R[+L]]R[-L]+L	Error message
Production Successor 2	F[+X]-X]FX	Error message
Production Successor 3	FF--F+F+F]+[+F-F-F]	Error message

Table 5.2 : Bracketing Error Results

The program noticed all of these errors and issued the appropriate error message to the user.

As discussed earlier in section 4.6, the JSpinners that receive the input for the start angle, turn angle and the number of iterations prevent any erroneous input from being entered.

5.2.3 Dealing with Large Computations

To evaluate the programs ability to warn against large computations, several L-systems were chosen

and the number of iterations was raised until the program failed to draw the image in an allowed time limit – chosen to be 30 seconds. The L-systems chosen are detailed in table 5.3 below.

Name	***Axiom***	***Production Rule(s)***
Tree 1	F	F → FF-[-F+F+F]+[+F-F-F]
Koch Snowflake	F++F++F	F → F-F++F-F
Dragon Curve	L	L → L+R+ R → -L-R
Hexagonal Gosper Curve	L	L → L+R++R-L—LL-R+ R → -L+RR++R+L--L-R

Table 5.3 : L-system details

The results of raising the number of iterations are shown in table 5.4 below.

L-system	***Iterations***	***Result***
Tree 1	4	Image drawn
	5	Warning Issued – image then drawn after approx. 5 seconds
	6	Warning Issued – image failed to draw after 30 seconds
Koch Snowflake	6	Image drawn
	7	Warning Issued – image then drawn after approx. 20 seconds
	8	Warning Issued – image failed to draw after 30 seconds
Dragon Curve	13	Image drawn
	14	Image drawn after approx. 10 seconds
	15	Warning Issued – image then drawn after approx. 30 seconds
	16	Warning Issued – image failed to draw after 30 seconds
Hexagonal Gosper Curve	5	Image drawn
	6	Warning Issued – image failed to draw after 30 seconds

Table 5.4 : Large Computation Warnings

It can be seen from table 5.4 that every time the image failed to be drawn within 30 seconds the program had previously issued a warning to the user that it may fail. The image was successfully drawn after issuing a warning in some cases, this is acceptable since the program is only warning the user that the program *could* fail, not that it definitely will.

5.2.4 Comparison to Available Software

The program produced in this project has improved upon the software seen in section 2.2. Compared to the two Java applets this program has many more features, including the ability to save the images created, specify the size of the images, specify both the line and background colours of the image and issue error messages and warnings to the user. In addition to these features, I believe that this

program has a better overall ease of use than the two Java applets.

This program is certainly much more user-friendly than Fractint when it comes to drawing L-systems. This is partly is because Fractint's main purpose is not to just draw L-systems, they are just one of the many types of fractal it can produce. Also, the graphical user interface in this program gives it a much better ability for the user to experiment with these L-systems and view the effects of altering parameters.

5.3 Possible Extensions

There are several areas where this project could be developed further. Two of these areas are based around extending the types of L-systems that the program can deal with, in particular to allow for 3D and not just 2D L-systems and also to deal with stochastic L-systems. There are also many more possible extensions to the L-system theory. There is also scope for improving the user interface and its features.

5.3.1 3D L-systems

As discussed briefly in section 2.1.3.2 there exists a turtle interpretation for L-systems that is capable of producing 3-dimensional images. Much of the input processing could be dealt with in a similar way as in this project and the Turtle class could possibly be extended to cope with 3D L-systems. The most challenging part of this extension is likely to be implementing the actual drawing of 3D images. If a suitable method for displaying 3D images were found then it would certainly be possible to extend this implementation to 3-dimensions.

5.3.2 Stochastic L-systems

Again, as discussed in section 2.1.4 the theory can be extended to cover stochastic L-systems. This project could be extended to deal with these. The GUI would have to be adapted to allow for the input of the probabilities assigned to each production and the turtle implementation could be adapted to apply productions based on these probabilities.

5.3.3 Improving the Interface

There are also many areas for improving or changing the user interface. The user could be given more control over the appearance of the diagram – they could be able to change the line thickness or style for example. A method for zooming in on particular areas of the diagram is another possibility. Some way of making the diagrams 'animate' through several iterations to give the appearance of growth is quite a large potential extension. These are just a few of the many possibilities for extending the interface created in this project.

Bibliography

Barker, J, (2002), *Beginning Java Objects.* Birmingham: Wrox, pp.13-24, Chapter 16.

Chomsky, N, (1957), *Syntactic Structures*. The Hague: Mouton.

Deitel, H M & Deitel, P J, (2001), *C++ How to Program, Third Edition.* New Jersey: Prentice Hall.

Dyer, M E, (2002), *Lecture notes - CO22 Theory of Computation*, School of Computing, University of Leeds.

Efford, N, (2002), *SO21 Handbook, 2002-03 Session,* School of Computing, University of Leeds.

Fractint, (2004), *Fractint Development Team Homepage.* URL:http://www.fractint.org [2nd December 2004]

JavaView, (2004), *L-System Tutorial.* URL:http://www.javaview.de/vgp/tutor/lsystem/PaLSystem.html [1st December 2004]

Mandelbrot, B B, (1983), *The fractal geometry of nature*. Oxford: Freeman. Chapter 6.

Prusinkiewicz, P, & Lindenmayer, A, (1990), *The Algorithmic Beauty of Plants*. New York ; London: Springer-Verlag, pp.1-28.

Rozenberg, G & Salomaa, A, (1980), *The Mathematical Theory of L Systems*. New York ; London: Academic Press.

Sun Microsystems, (2005), *Java 2 Platform, Standard Edition, v 1.4.2, API Specification.* URL:http://java.sun.com/j2se/1.4.2/docs/api/index.html [22nd April 2005]

Verma, M, (2004), *Milan's L-System Generator.* URL:http://www.dcs.qmul.ac.uk/~milan/LSystemApplet.html [1st December 2004]

(2004), *Fractint Homepage.* URL:http://spanky.triumf.ca/www/fractint/fractint.html [2nd December 2004]

Appendix A: Project Experience

I had never undertaken a piece of work of this size and importance prior to this project, and initially it seemed a very challenging prospect. However, it gave me the opportunity to develop my research, project management and programming skills that have been gained during my degree.

I was very pleased with the end product in terms of the program produced and I found the programming to be a rewarding and satisfying experience, although it was also an extremely frustrating experience at times! It would have been nice to have had more time to develop the software, as there are several enhancements that would improve the program. This wasn't possible because the programming that was completed was fairly time consuming. In fact the programming side of things took up the majority of time spent on this project, possibly at the expense of writing up the work. I would urge other students not to underestimate the amount of time involved in writing working code, especially if many of the methods involved are unfamiliar, and to ensure that work is started on the implementation as soon as possible.

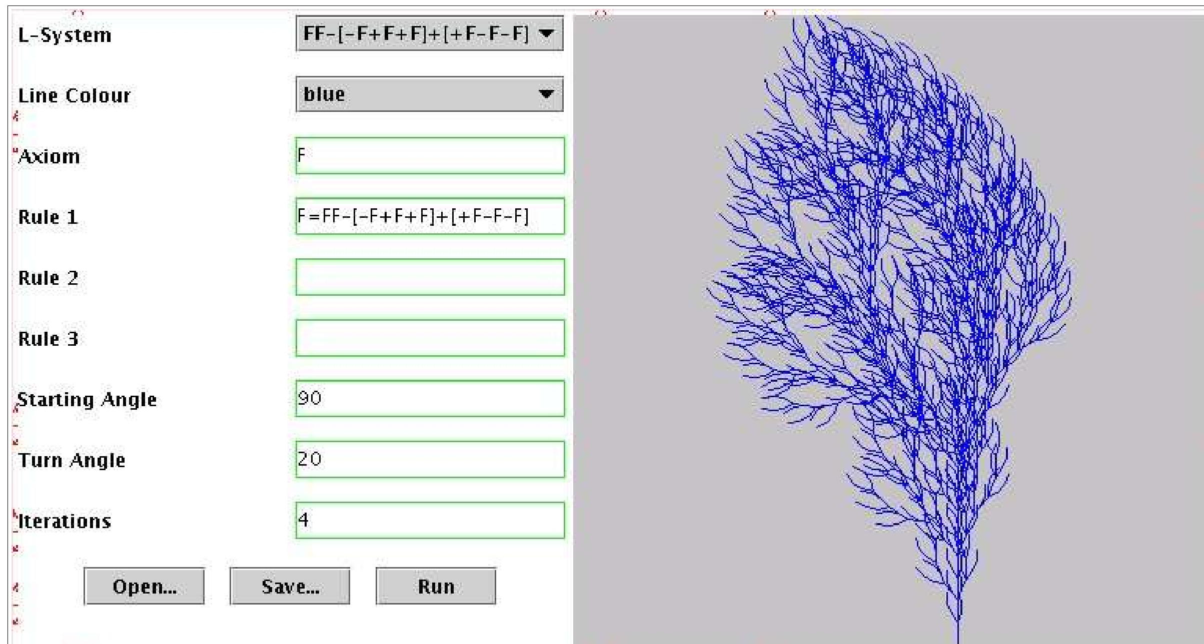
Although I did complete some of the written work during the course of the project, I left the majority of producing the report until the end. I found that spending a good amount of time on writing the mid-project report to be invaluable as this meant that I already had a good background chapter written up before Christmas. In hindsight, it would have been sensible to have compiled the whole report as the project progressed instead of leaving the bulk of it to the end. This is an especially important consideration since it is only the report that gets marked, so its important not to spend all of the available time on the implementation.

I found it difficult to balance the time spent working on this project with all of the other coursework and study that my other courses entailed. This was especially difficult because this was only a 20 credit (instead of 40 credit) project, which meant that in both semesters I had five other courses to concentrate on. I also found that this being a 20 credit project made it difficult to judge exactly how much work was required of me, as the majority of projects were worth twice this amount.

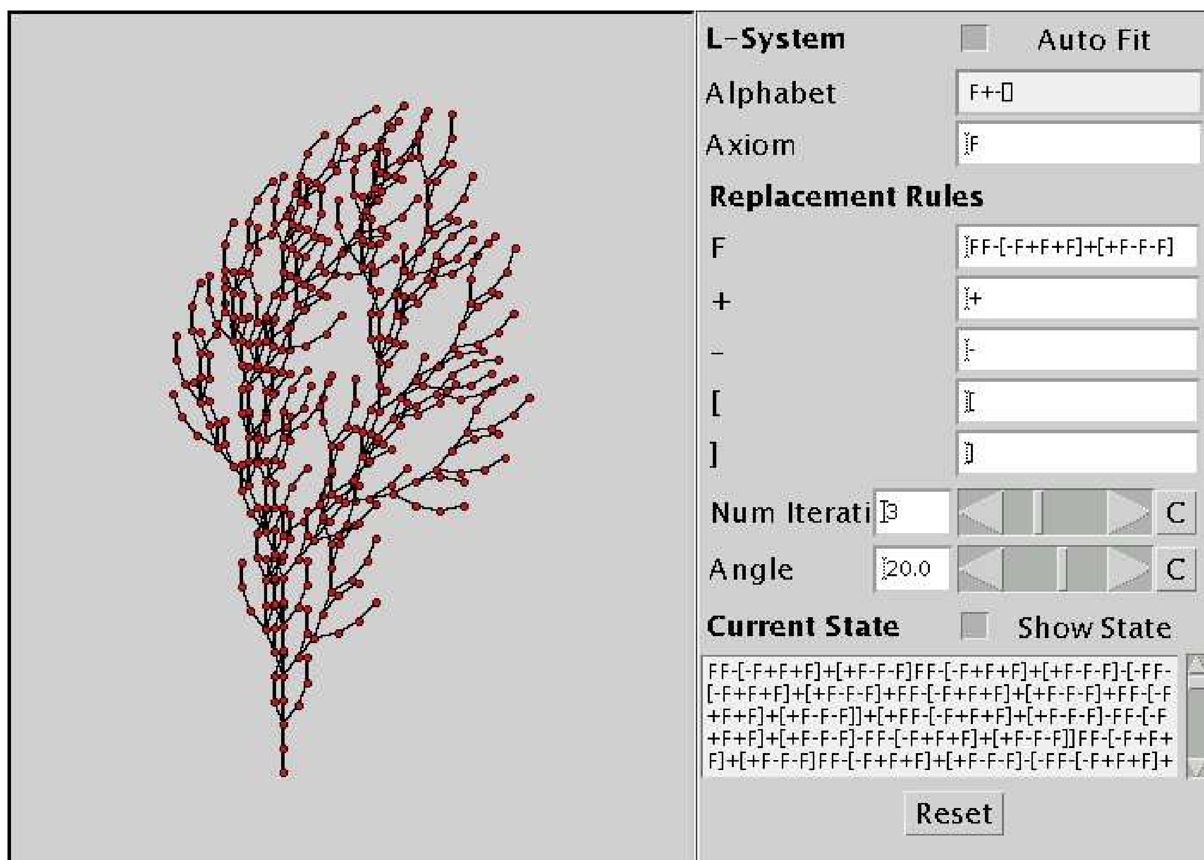
I would advise future students to ensure that they make a start on the project as early as possible in the first semester because the deadline may initially seem very far off but the time disappears extremely quickly, especially with all the other work involved in final year. I would also recommend getting started on the implementation process as soon as possible and producing good quality written work throughout the duration of the project to save lots of time writing up the report at the end of the project.

Appendix B: UML Diagram

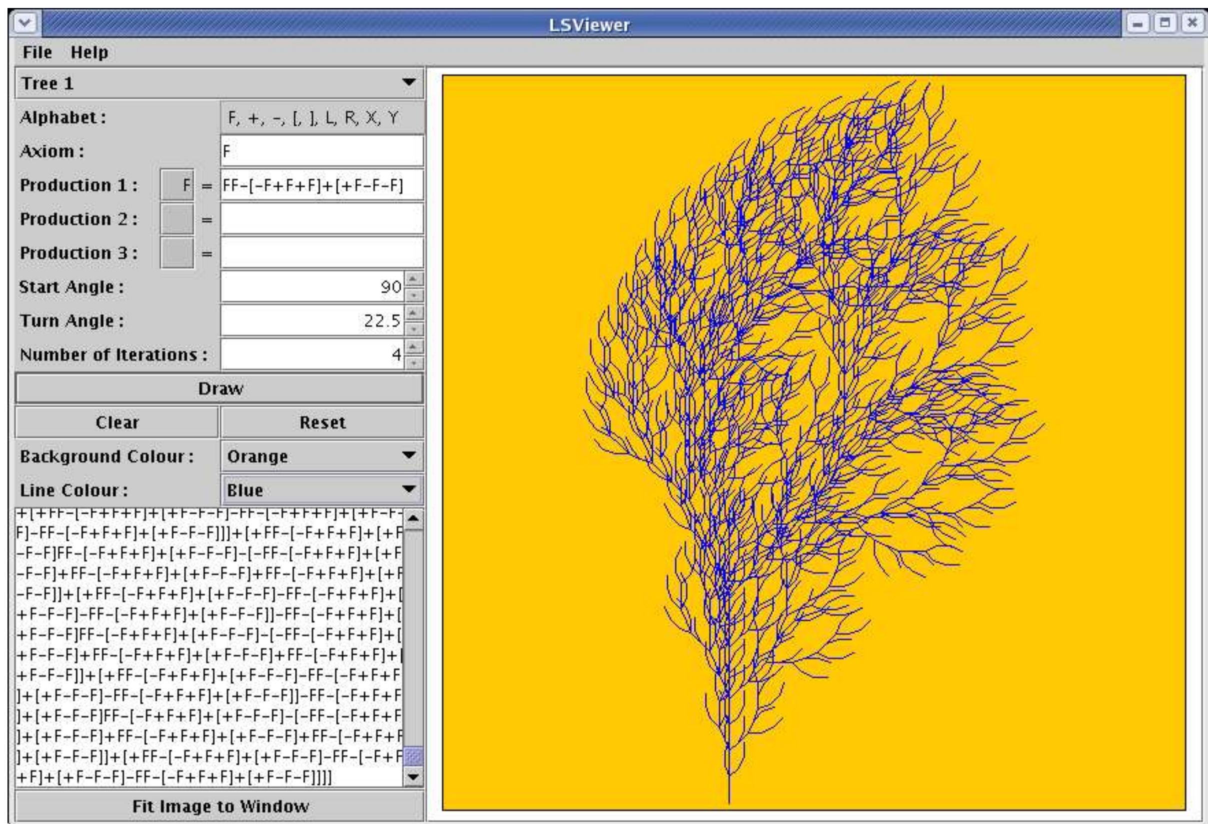
Appendix C: Screen Shots



Milan's L-System Generator. http://www.dcs.qmul.ac.uk/~milan/LSystemApplet.html



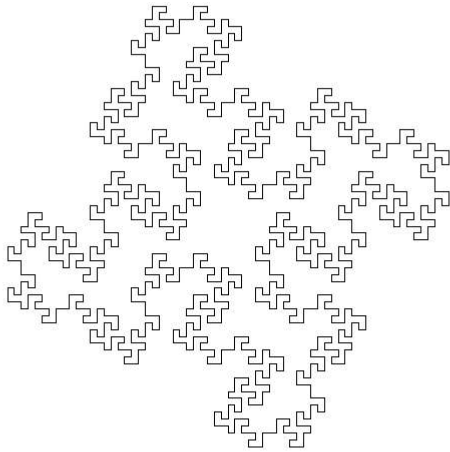
http://www.javaview.de/vgp/tutor/lsystem/PaLSystem.html

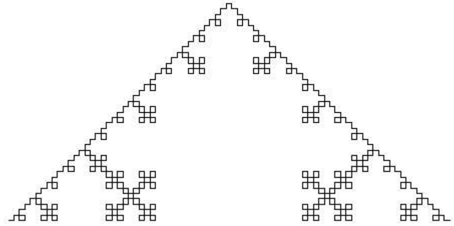


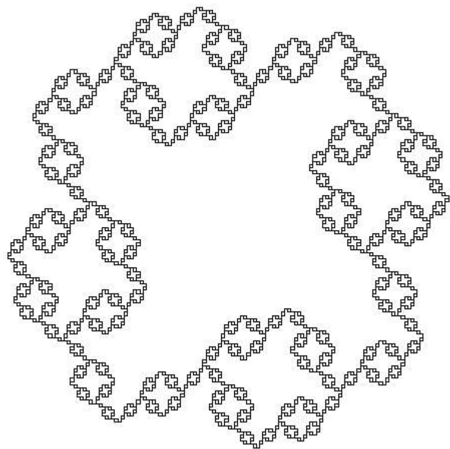
LSViewer created in this project.

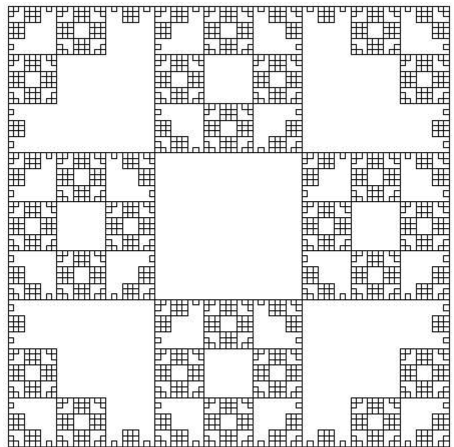
Appendix D: Produced Images

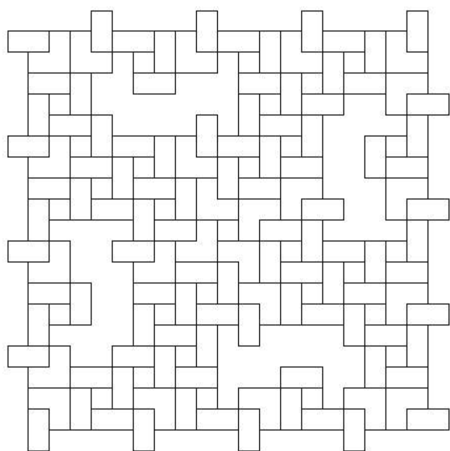
The black and white images on the following 3 pages are those discussed in section 5.2.1.

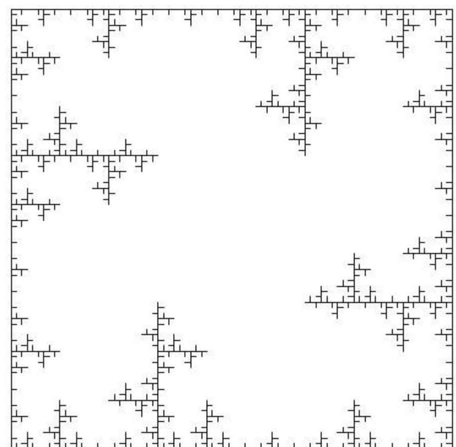


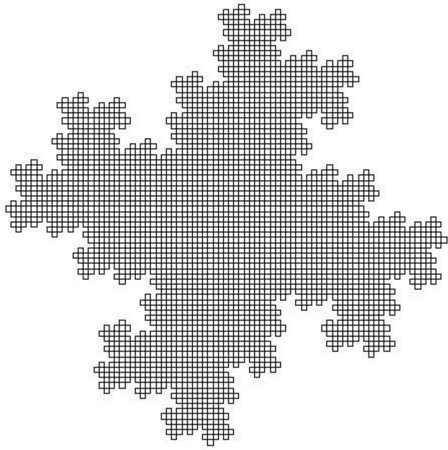
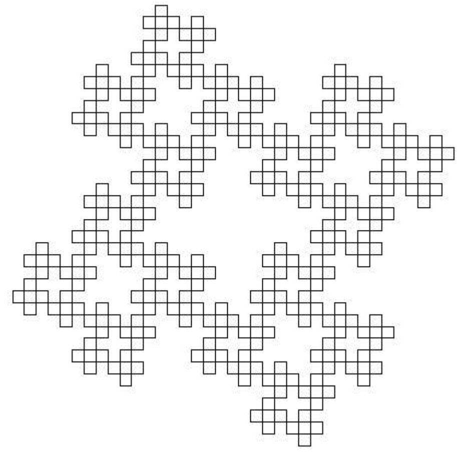
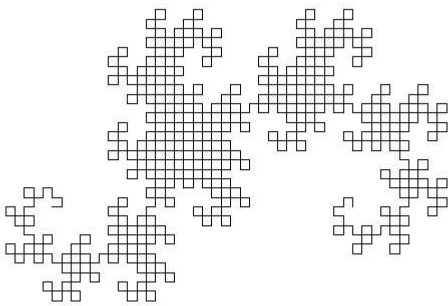
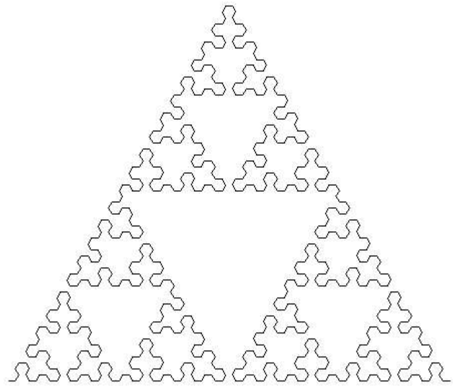
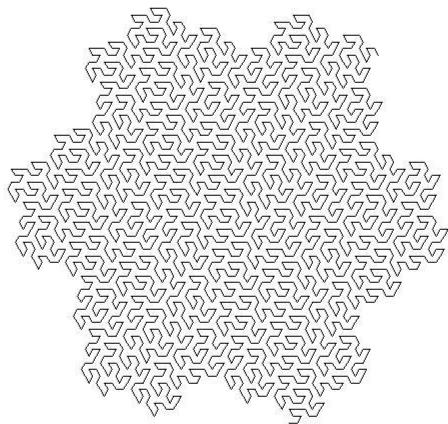
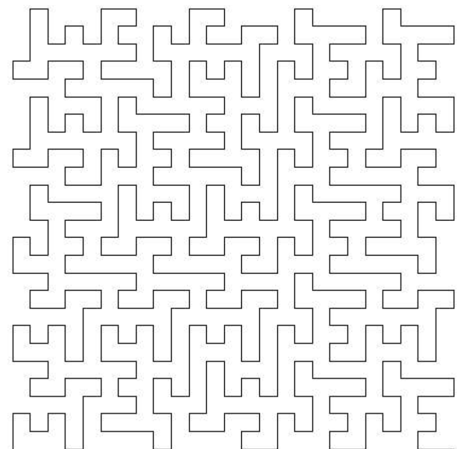


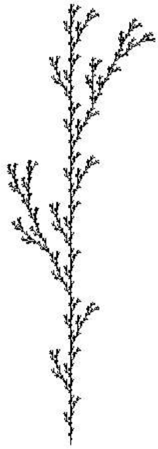
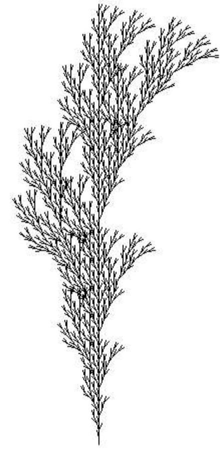
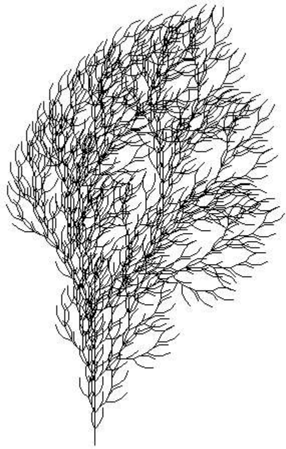
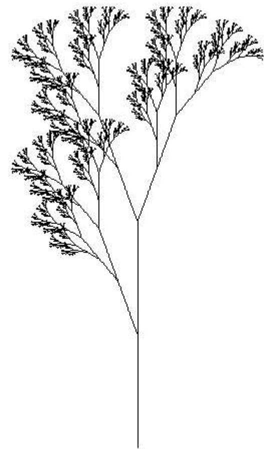
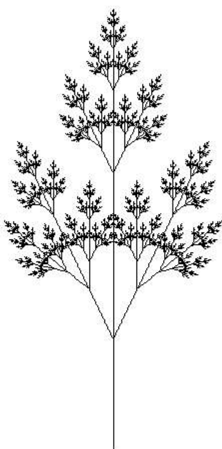
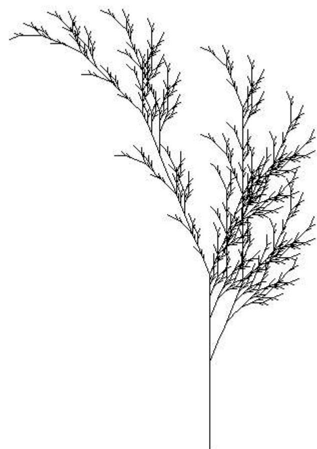








The following are a small selection of images demonstrating the ability of the program to generate pictures of varying colour and size.