The Complexity of L-systems with Cryptography

Vural Erdogan
School of Computer Science,
University of Hertfordshire, College Lane,
Hatfield, Hertfordshire, AL10 9AB,
The United Kingdom
ve17aae@herts.ac.uk

***Abstract*—We created an L-system engine which includes rewriting string method with Logo Turtle graphics to achieve the complexity of L-systems. Also, the program possesses prototype of an encryption-decryption algorithm that enables users to create a private-encrypted message to send the second party by using strings or decipher the encrypted message which comes from second parties. In order to do this, we used reverse engineering method for the cryptographic part that runs opposite way of L-systems. This paper tried to demonstrate that L-systems might form the new fractal structures and contribute other fields such as cryptography.**

Keywords—L-system, Lindenmayer System, Cryptography, Morse Alphabet, Complexity, Plant Modelling, Reverse Engineering.

I. INTRODUCTION

Figure 1: An example of Lindenmayer System.

Lindenmayer systems (L-systems) are a rewriting and visualisation method developed by Aristide Lindenmayer in 1968. The system creates new generations from an axiom, seed, by applying grammar rules for each iteration (shown in Fig. 1). The last generation or iteration is rendered graphically by reading each character in the string. With this method, A. Lindenmayer contributed to simulate the growth of multicellular organisms. Particularly, modelling plants and fractals were stated in these studies.

The popularity of L-systems initiated many botanists, biologists and computer theorists to study in this field [1]. One of the famous computer scientist in this field is P. Prusinkiewicz who attempted to develop L-systems in his entire studies [2], [3], [4]. Besides these, L-systems does not only consist of strings or visual graphics. Some computer scientists proved that L-system might be used in a different field out of modelling plants or fractals such as music rendering that creates variable sounds depend on the features of the branches while growing [5], [6]. This improvement also illustrates that L-systems might contribute to the other areas.

In this project, first, we observe previous studies about plant modelling and fractal structures using our L-system engine. Second, we develop new rules in order to achieve the complexity of fractal structures; some structures are similar to plant growth. Therefore, we may clarify that the complexity of fractal forms might be achieved a variety of different rules. Third, we indicate to code a new algorithm that reversing the L-system in order to reach the rules and axiom from the last generations. The reverse engineering may provide us to find the rules from the least number of known values such as generations or axioms which are given by L-system. Last, with this algorithm, we plan to encrypt a message to send the second parties. Also, if an encrypted message receives from the second persons, we try to solve the cypher by using the reversing algorithm.

The second chapter explains how and which methods are implemented to achieve the complexity of previous L-system studies by the program, and we propose new fractal structures. In chapter three, we simulate the plant-like structures that proposed by other scientists and propose new plant-like patterns which are developed from the previous studies. The fourth chapter illustrates how we develop a reversing algorithm and which features are included. At the last chapters, we indicate how we can create an encrypted message and how we decipher it (All the codes and program link are included in appendices).

II. METHOD

A. *L-systems*

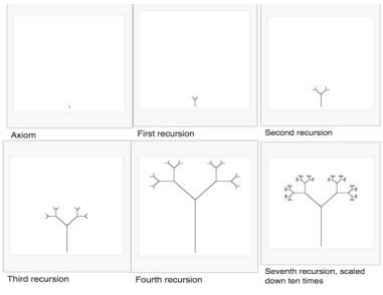


Figure 2: The visual generations of a rule [7].

The L-system implements a recursive method for each generation that contributes leading to self-similarity.

Therefore, fractal forms might be portrayed with an L-system conveniently. Lindenmayer indicated the resemblance of fractal structures between generations of the rule and natural plants growth in a visual environment in his book. An example of the fractal forms and its similarity of natural plant growth are indicated in Fig. 2 ([7]) that reiterating generations. Each step of the rule applied the recursive method to produce new branches.

Beginning of the programming, we decided to code a method to reiterate the strings by implementing the rules. After each rule implemented, the method should append them to the new string. Therefore, we may obtain a new generation. Fig. 1 is a clear example of a string L-system. The initial state of string, axiom, produces the new generations. To develop the string L-system to graphical L-system, turtle graphics, introduced by Logo programming, were implemented which is a popular method for drawing shapes and figures [8].

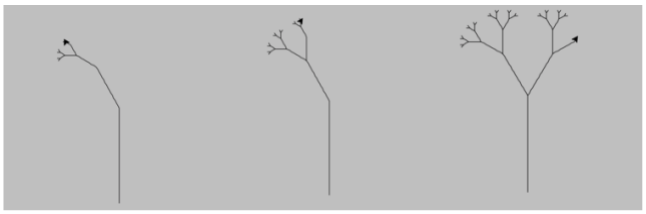


Figure 3: The Turtle Graphics.

For this project, we planned to program it with Python programming language. After several comparisons for IDE, we decided to do it via "PyCharm CE" that includes turtle library, allow us to plot our codes, and random library, which assist us creating an array and adjustable random distributions for encryption. The turtle in the turtle library enables the user plot codes in a sequence. With Pop and Push functions that allow us to return starting point of producing branch and create a different fractal or a branch again, we are able to create a Lindenmayer system on turtle graphics (Fig. 3).

B. Fractals

Figure 4: Dragon Curve

In the literature, we encountered several fractal forms which are named as Koch Snowflake [9], Dragon Curve (Fig. 4) [10], Sierpinski Triangle (Fig. 5) [11], and we tried to simulate them on the engine. In the codes, there are some character features for the turtle to draw it.

Table I: Properties of Dragon Curve

Axiom	FV
Rule 1	V = V+YF
Rule 2	Y = FV-Y
Generation	11
Length	1.5
Angle	90°

In Table II, 'F' symbolises a function, which will be used for 'go forward' entire paper but in this example, we preferred V and Y for 'go forward', '+' symbolises turning turtle head with a positive angle which is given in the table, '-' is same as '+' but it turns the turtle head negative way. 'Generation' determines how many times the axiom will be iterated. 'Length' scales the whole structure, if the user prefers to observe patterns with a close distance, the user should choose length bigger.

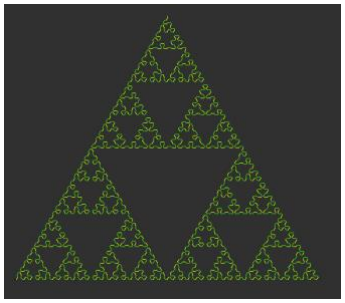


Figure 4: Sierpinski Triangle

First, we tried to draw the previous fractal structures in the literature. Therefore, we started with a Dragon Curve which is a famous example of fractal forms. Another fractal form is Sierpinski Triangle in Fig. 5. The triangle is also a popular form of math area. In Fig. 5, the triangles have a different form when we compare Sierpinski triangles. All the triangles' edges illustrate fluctuations. However, it still seems as Sierpinski Triangle when we look whole structure.

Table III: Properties of Sierpinski Triangle

Axiom	F
Rule 1	F=Y-F-Y
Rule 2	F=F+Y+F
Generation	5
Length	1.5
Angle	60°

In Table III, 'F' and 'Y' symbolises 'turtle go forward' as we mentioned before. '+' and '-' will symbolise angles' direction through this paper. This usage is a general usage when we compare symbols in literature. Another structure is Koch Snowflake which is one of the earliest fractal

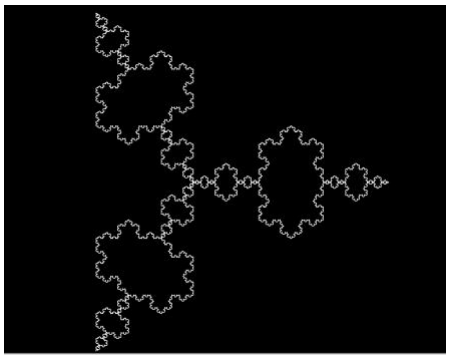


Figure 6: Anti-Snowflake

structures (1904). In Fig. 6, we attempted to print another version of Koch Snowflake by changing rules [11].

Table IV: Anti-Snowflake

Axiom	+F--F--F
Rule	F = F-F++F-F
Generation	5
Length	1
Angle	60˚

We accidentally discover Anti-Snowflake structure while experimenting Koch Snowflake, which has 'F+F--F+F' rule; on the contrary Anti-Koch Snowflake has 'F-F++F-F' rule. After observing several fractal structures, we decided to develop some patterns to achieve more complex structures. The new rules should have included similar fractal forms which mean that minor parts of the structure must resemble whole major part of the structure. Therefore, after numerous experiments, we reached a new fractal form in Fig. 7 by adding some new functions to rules. The red lines and green lines show which part observed by reiterating.

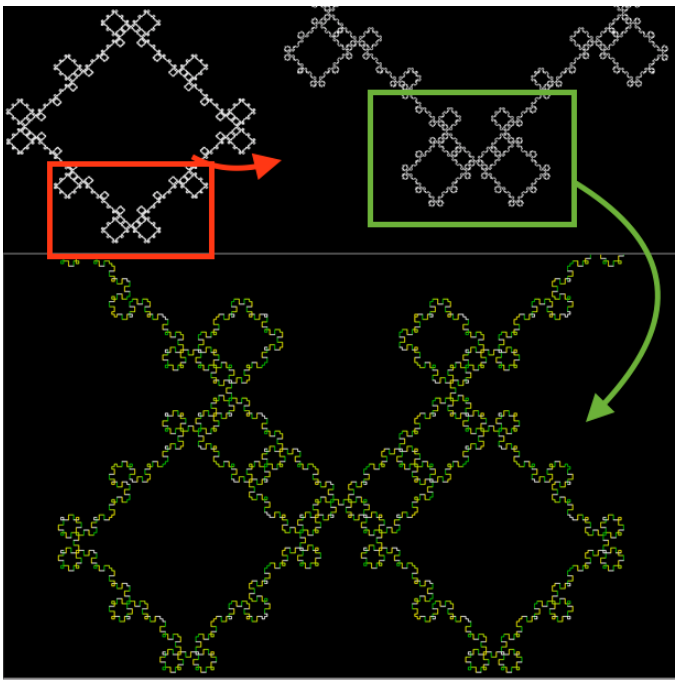


Figure 7: The New Fractal Pattern 1. (Length: Top; 0.5, 1 Bottom; 2.5.)

Table V: Properties of the Pattern 1

Axiom	F-F-F-F
Rule	F = X+F-F+F+F-F+X
Generation	5
Length	2
Angle	90˚

Fig. 7 indicated a square fractal structure due to choosing 90˚ degree. The axiom has been determined 'F-F-F-F'. As a result, we may obtain a closed curve figure since four 'F' iteration leads to turn 360˚ degree. If the user prefers three F (F-F-F) for the axiom, it will display a 270˚ figure which is 90˚ open. Another point is 'X' has been used for the new rule. The 'X' function runs the turtle opposite of 'F'. In other words, when the program detects 'X', it orders 'go back' to the turtle a length distance. 'X' function contributed us to develop new more pattern such as Fig. 8.

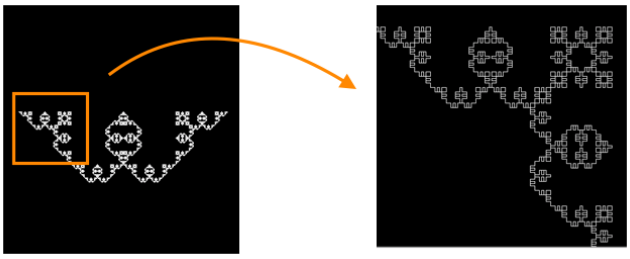


Figure 8: The New Fractal Pattern 2 and its zoomed part.

Table VI: Properties of the Pattern 2

Axiom	F
Rule	F = F-F+F+F-X-F+F+F-F
Generation	4
Length	4
Angle	90˚

The Pattern 2 consists of several fractal forms. In Fig.8, we may observe various quadratic fractals. Also, the pattern includes some structures which might resemblance humanoid-head-like (Fig. 9). Due to this similarity, we named it 'humanoid'. The pattern only shows one edge since the axiom is determined 'F'. If the user would choose to observe a curved figure, the axiom should be integrated to 360˚ degree. Therefore, choosing the axiom as 'F-F-F-F' will contribute to observing a closed figure. The other point, 'X' function is utilised for this pattern in the middle of the rule. Therefore, the turtle implements 'go forward opposite of turtle's head direction' when it reads 'X'.

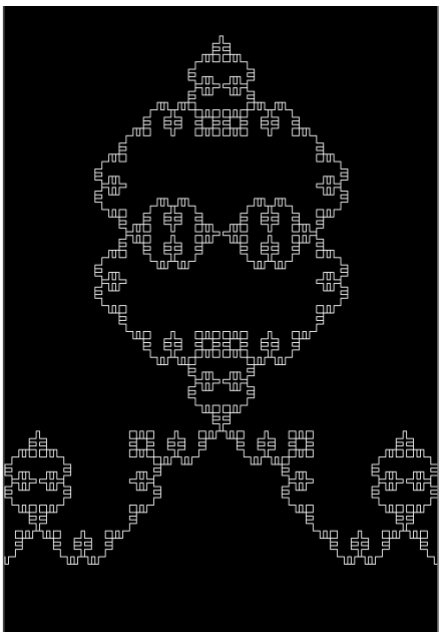


Figure 9: The Humanoid

C. Plants Modelling

In this section, after observed the mathematical fractal forms in the literature and proposed new structures, we attempted to draw some fractal tree structures proposed by other scientists. While researching the articles on literature, we encountered with several variations of L-systems such as stochastic L-systems, Context-Sensitive L-systems and Bracket L-systems [12]. As a result, the second method has been determined as developing Bracket L-systems to achieve the complexity of L-systems. In order to do that, we changed angle, axiom, constant and rule of the proposed L-systems, while simulating new samples.

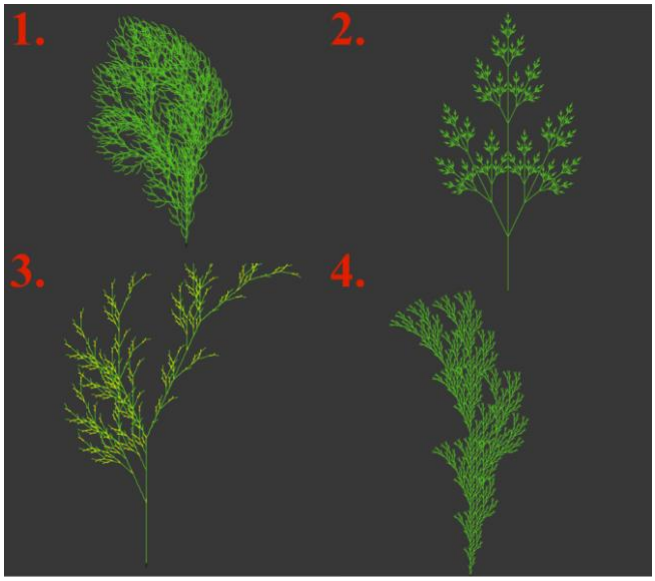


Figure 10: The plant-like structures created by the L-system engine

Table VII: Properties of the plant-like structures in Fig. 10

Abv*	*1. Plant*	*2. Plant*	*3. Plant*	4. Plant
Ax	F	X	Y	F
R1	F = FF-[-F+F+F]+[+F-F-F]	X = F[+X][-X]FX	Y = F-[[Y]+Y]+F[+FY]-Y	F = F[+F]F[-F][F]
R2	-	-	F = FF	-
G	4	7	5	5
L	3	1	3	4
A	22.5	25.7	22.5	20

In Fig. 10, we obtained the forms with the program which were proposed by A. Lindenmayer [1]. All the properties of the structures are explained in Table VII. We illustrated the rules that belong the plants and emphasised lack of the rule by symbolising with a hyphen "-". The Bracket L-system is implemented for plant modelling due to its branching function that draws a line, goes back the start point, and afterwards, draws another line. Therefore, it might create a branching form with this method. For creating a new string generation, the axiom should be processed by the string creator function which gets characters from the axiom and generates a new string according to rules. However, the program iterates the axiom's characters one by one. For example, if the program detects 'AB' as an axiom, first, it creates a new string iterating 'A' rule, afterwards program saves this in the memory, then it generates for 'B' this time. The program runs it in a short time. As a result, this idea pushed us to code an encryption-decryption algorithm.

* Abbreviations: Axiom, Rule 1, Rule2, Generation, Length, Angle respectively.

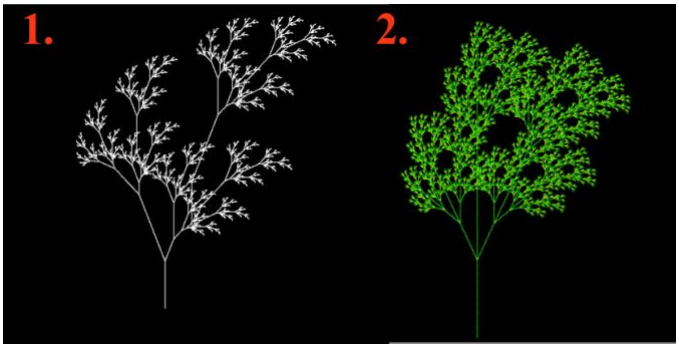


Figure 11: The new plant-like forms.

Table VIII: Properties of the new proposed forms in Fig. 11

	1. Plant	*2. Plant*
Axiom	Y	Y
Rule 1	Y= F[-FY]+Y-[F+FY]	Y = F[-Y][+Y]-[[F]+FY]+[+FY]-Y
Rule 2	F=FF	F=FF
Generation	7	6
Length	0.5	1.7
Angle	22	25

Regarding the previous studies, we attempted to reach new plant-like forms. The forms in Fig. 11 illustrates the new findings, and both plant features have been explained in Table VIII. The first plant structure may evoke the plant 2 in Fig. 10; however, we developed both of them from the plant 3 (Fig. 10). Both plant-like structures possess two rules. All these plant-like and fractal forms have been generated by the proposed L-system engine.

D. The Reversing Algorithm

After obtaining several bracket OL systems, we decided to program a reverse algorithm that works opposite of L-Systems (Fig. 12). To do this, we applied the last generations which are '[a, b, a, a, b]' and '[a, b, a, a, b, a, b, a]' respectively. We named all the generations according to their orders by inspiring from Fig. 12. As a result, it is fair to name last generations as 'G4' and 'G5'. The generations contribute us to run the experiment the least number of known parameters. Therefore, we determined steps of the reversing algorithm first:

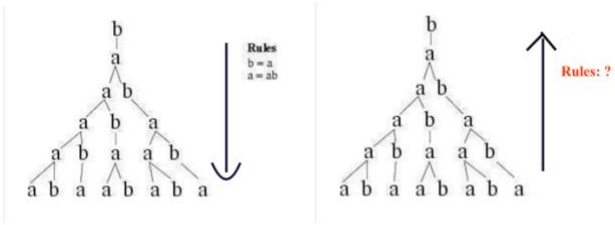


Figure 12: L-system and The Reversing Algorithm.

• Finding last generations from the iterated system or given system (G4 and G5).
• Calculating unique characters of the generation which comes from before the last generation (E.g. G5: 'A', 'B').
• Generating all the combinations and permutations of these unique strings ('A', 'B, 'AA', 'AB', 'BA'...).
• Iterating each permutation with L-system and compare them whether the created generation equals to last generation (G5) till the program verifies.
• After confirmed the rules, the found rules (not the unique characters) should be detected in the last generation (G5), which contributes us to explore the unique character in the one previous string (G4), and the program tries this method swapping generations with previous ones after confirming every later generation till the determines the axiom.

The steps lead us to select the iterated rules' length less than five in order to find the rules rapidly. Otherwise, the permutations and longer rule length cause the program to delay the process time because of confirming the generation.

E. The Encryption Algorithm

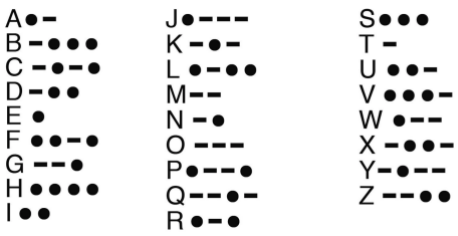


Figure 13: Morse Alphabet

The all the above steps in the reversing algorithm assisted us to define encryption logic. With reversing method, we decided to encrypt an L-system which consists of Morse alphabet strings that will enable us to send a message to second parties. The Morse alphabet (Fig. 13) is often used for emergency situations; however, due to the 2-letter structure, we favoured to use it (Fig. 14). Consequently, we were going to be able to import and send a meaningful message. The encryption algorithm was more complex than the reversing algorithm because of using the random library. We preferred to alter characters from the iterated strings in the L-system to encrypt a message. For instance, if we have a string in Generation 3 (G3) and Generation 4 (G4) which are '•-•' and '•-••-' respectively, we alter '-' character in the middle of G4

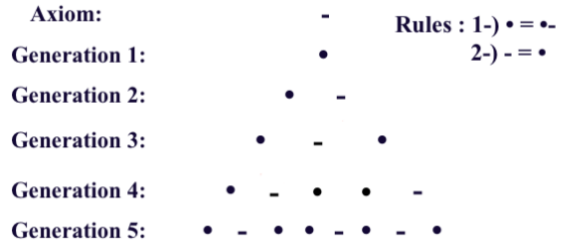


Figure 14: The Morse Alphabet with L-system

to '•••' and '•' character in the middle of G5 to '•--•-' in order to create our first message 'A' which is the equals in our Morse Alphabet (Fig. 15). While coding on the program, we may convert '•' equals 'A' and '-' equals 'B' to create a message conveniently such as Fig. 12. In this way, we may

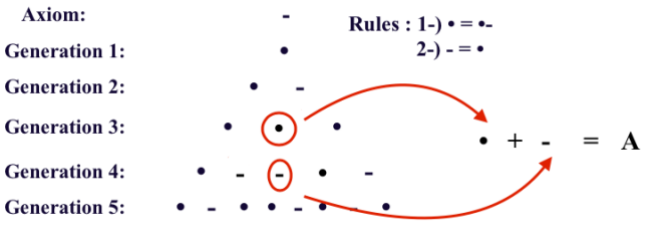


Figure 15: 'A' message in L-system

picture that how the algorithm will be implemented.

In our codes, we determined the rules number as three; nevertheless, two rules have been determined for the experiments which are optional and might be any two characters. Also, the axiom can be chosen more than one character or any character by depending on rules. Even though the rules are optional to choose, we generally stated 'A' and 'B' characters which are ideal for the program due to the program's code structure.

Instead of creating a message with a regular order, we chose to generate it with a random library that generates numbers in the array, and this array's numbers define the orders of the characters which will be modified. For example, assuming that the random library generated an array which is '['4', '6', '10']' that defines fourth, sixth, and tenth characters will be altered for the encryption. Therefore, the message could be generated in a random order each time. The random library only creates numbers as long as all the generations' total character number except last two generations. Consequently, we were not going to modify last two generations' characters that enable us to merge two algorithms, the reversing algorithm and the encryption algorithm.

The crucial point is that sometimes random library could generate same numbers or the numbers which are higher than later ones. In order to fix 'higher number' problem, we added some codes which contribute the array is being sorted from

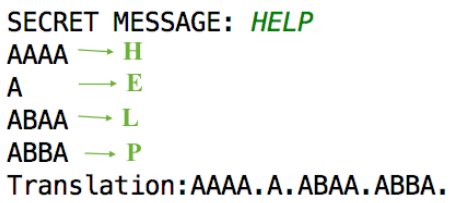


Figure 16: Translation of 'HELP'.

small numbers to large numbers automatically. Also, an increasing number of the chosen order with 'sorted' feature prevented the same number problem.

F. The Decryption Algorithm

The Decryption Algorithm is the simplest one among others. The logic of this algorithm is that if an encrypted message receives, the algorithm finds the rules from last generations, explodes other generations till the axiom just as the reversing algorithm, and compare all the generations with the encrypted message. The program starts reading from the axiom to the last generation and left to right while comparing, and when the program detects a modification, the program checks it whether it is 'A' or 'B'. If it changed from 'A' to 'B', the program saves it as 'B'. After, the program checks other secret characters and appends them to new string bound to order.

While running the program, we realised that some issues appeared in the results. The program could not define a message which possesses many permutations due to the adjacent structure. For example, the message 'HELP' equals to 'AAAAABAAABBA' in our Morse alphabet. At the same time, its equality also coincided with other words such as 'SVSG'. For preventing this, the dots have been used between each character (Fig. 16). Therefore, the program could realise the characters separately (Fig. 17). As a result, we may observe that 'SVSG' represents 'AAA.AAAB.AAA.BBA' with a 'dot' translation.

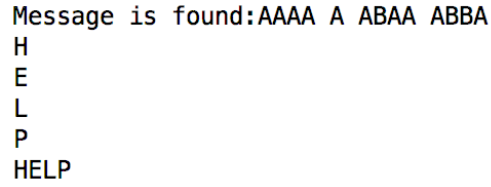


Figure 17: Translation of the encrypted message.

III. Conclusion

In chapter B, we achieved to draw the previous fractal studies which are Anti-Snowflake, developed from Koch Snowflake, Dragon Curve and Sierpinski Triangles by the proposed program. Utilising Logo Turtle graphics enabled us to draw patterns the same. As a result, first experiments, re-drawing the fractals in the literature by the proposed L-system engine, was achieved.

The second simulations in chapter B demonstrated that the new fractal forms might be developed by the L-system program. The rules which entered by the users may be processed using our engine. Therefore, the users have an opportunity to draw their patterns. Also, the proposed new patterns in chapter B illustrated that the complex repetitive or symmetrical patterns could be reduced to the simple rules concerning complexity. In addition, the repetitive forms might be varied utilising more complex rules. Consequently, the second experiment, the new patterns can be developed by the engine and enhanced as the complexity, was achieved.

In chapter C, we examined the L-system plant modelling with the program's ability whether it can draw the plant-like structures or not. The third simulations illustrated that the engine is able to draw previous plant-like structures in the literature. On the other hand, the fourth simulations indicated that the new patterns could be generated by using the engine. Therefore, the third experiment, whether the engine is able to generate plant-like structures, was achieved.

We examined the reversing algorithm in chapter D whether the engine can run the system opposite way such as reversing engineering. The experiments showed that the program could reverse given L-system from last two generations to axiom. The fourth experiment, regarding whether reversing engineering is possible on L-systems, was confirmed.

The chapter E tested to encrypt a message. The results demonstrated that the L-system program is able to encrypt a message by using L-system algorithm and the encryption algorithm. On the other hand, the last experiments in chapter F showed that the encrypted message could be solved by the program using the reversing algorithm and the decryption algorithm. The last experiments regarding cryptography were confirmed.

Mainly, all these studies indicated that the complexity in nature could be reduced to the simple rules, and L-systems can be used in cryptography field.

IV. Discussion

In literature, we could not encounter any example of cryptography studies related L-systems while searching. This paper is the first research that combines L-system and cryptography. Also, the reversing engineering on L-systems can be considered a new development for L-system studies. The new patterns which generated from the software indicate different complex fractal structures although the findings developed from the existing rules with a trial and error method instead of based on math formulas.

In this paper, we might infer that the complexity of structures in nature may be not as complicated as we thought. In addition, it is fair to say that the complexity of shapes can be modelled with software. Some unusual or symmetrical forms in nature always seem intriguing and attract scientists to research. With these studies, we tried to encourage other scientists to elaborate L-systems further and to create new programs which reverse or encrypt in a more complicated way. Therefore, this paper contributes modelling complexity for inspiring other scientists in different areas. Morphology, biology, botanic, computer and cryptography science can be related to these studies.

While coding the L-system engine, we encountered some issues and limitations related to the program. First of all, although the turtle library helps us to draw patterns, it has a time-consuming style. We had to wait for each pattern to finish its drawing. That caused fewer experiment numbers. Second, we attempted coding to generalise everything; however, algorithms and steps were being confused when many exceptions and parameters take into account. For this reason, we switched the method by focusing only one purpose. That enabled us to code a running software. Nevertheless, it is not efficient for a secure message system. The program accepts two rules which shouldn't be more than 7 or 8 characters as the confirmation might last hours. Also, axiom or rules should not include non-functional characters, which are called 'constant', for the reversing algorithm. The best options for the reversing algorithm has been determined 'A' and 'B' characters. The user can select alternative characters for the reversing algorithm. Yet, the encryption and decryption algorithms based on 'A' and 'B' letters. The other point is that the functional rules for drawing should be selected among these characters; 'F', 'Y', 'X', '+', '-', '[' and ']'. Otherwise, the program will show an error.

Before starting the project, we expected to find the rules only one known value, last generation. Yet, our experiments failed to prove it. Therefore, we had to use at least two known parameters. We chose the last two generations; however, it might be alterable such as axiom and generation. Another point is that we planned to code an algorithm that can reverse, encrypt and decrypt any rule number or characters. We realised that it was an enormously wide range. We limited ourselves as we effort to show the logic of the method. Even so, this study met the majority of our criteria.

For future studies, the encryption-decryption algorithm might be developed with communication systems by processing digital and analogue signals. The coding circuit generates the signal, and the receiver might encode signal by using electronic filters. Therefore, the third parties may not achieve the message until the third parties decipher the algorithm. Another project can be increasing the speed of the program. If the iterating speed increases, encryption and decryption algorithms may find the solutions quickly that would have led to increasing the security of crypto-message. The reason is that the encryption method or encrypted message should not be understood by the third parties and should be in high-level security. For this reason, it is better to use many characters, rules, and around 200 generations for the process that cause decreasing the speed. Besides these, the complexity of L-systems can be enhanced by applying image recognition algorithm that may be combined with the reversing algorithm. With this method, taking a photograph of trees may illustrate the codes of trees.

REFERENCES

[1] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer New York, 1990.

[2] P. Prusinkiewicz, "Modeling of spatial structure and development of plants: a review," *Sci. Hortic. (Amsterdam).*, vol. 74, pp. 113–149, 1998.

[3] P. Prusinkiewicz, "Graphical Applications of L-Systems," *Graph. Interface*, pp. 247--253, 1986.

[4] P. Prusinkiewicz, M. S. Hammel, and E. Mjolsness, "Animation of plant development," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH '93*, 1993, pp. 351–360.

[5] L. C. Kim and A. Z. Talib, "Improving L-System Music Rendering Using a Hybrid of Stochastic and Context-Sensitive Grammars in a Visual Language Framework," Springer, Berlin, Heidelberg, 2012, pp. 46–53.

[6] P. Worth and S. Stepney, "Growing Music: Musical Interpretations of L-Systems," Springer, Berlin, Heidelberg, 2005, pp. 545–550.

[7] "L-systems." [Online]. Available: https://en.wikipedia.org/wiki/L-system. [Accessed: 04-Jan-2018].

[8] "24.5. turtle — Turtle graphics for Tk — Python 2.7.14 documentation." [Online]. Available: https://docs.python.org/2/library/turtle.html. [Accessed: 30-Dec-2017].

[9] Helge von Koch., "On a continuous curve without tangents, constructible from elementary geometry," 1904.

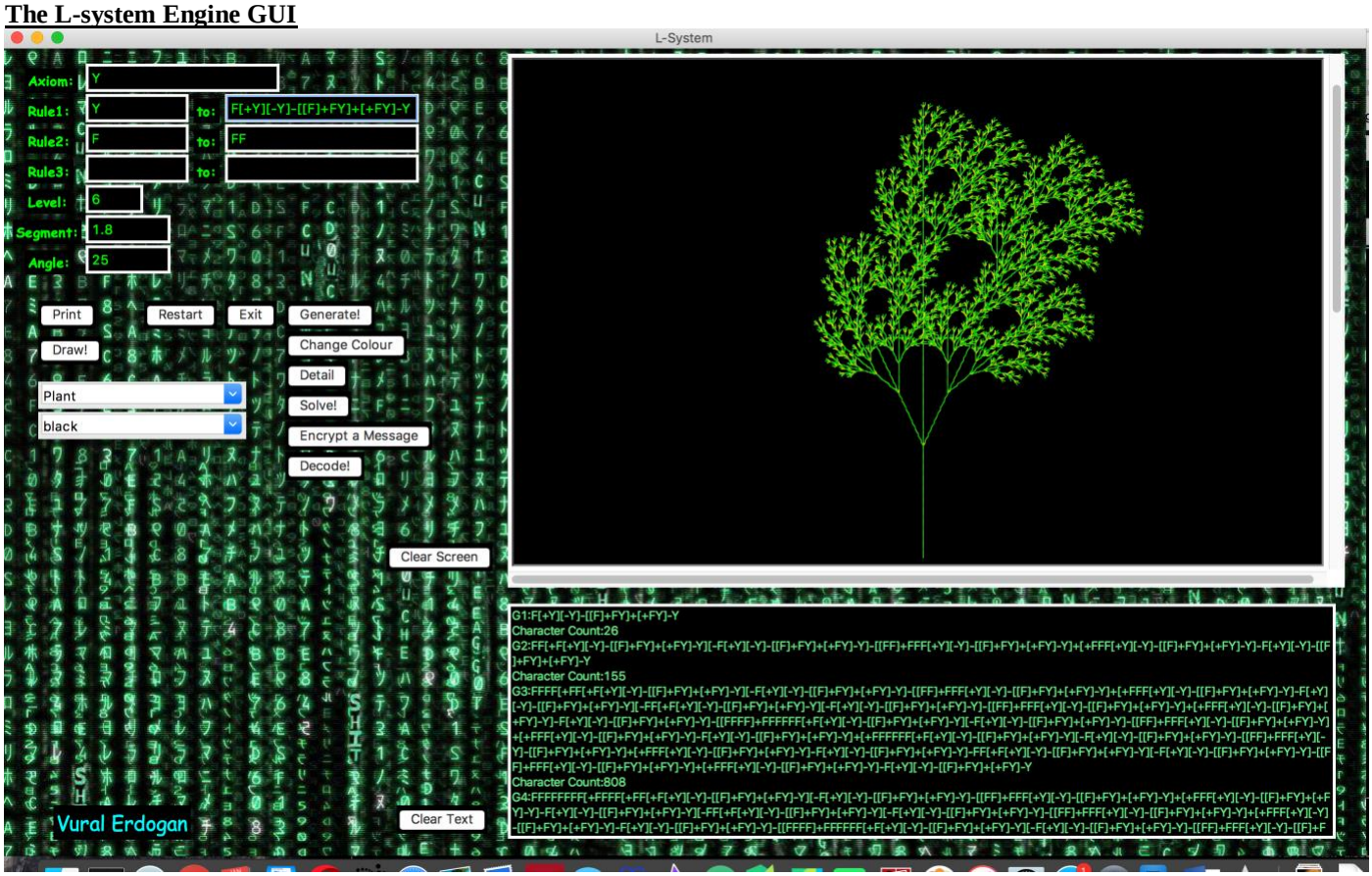
[10] "The Boundary of Dragon Curve." [Online]. Available: http://poignance.coiraweb.com/math/Fractals/Dragon/Bound.html. [Accessed: 30-Dec-2017].

[11] C. Elisa and T. LALLI Laura, "SIERPINSKY TRIANGLES IN STONE, ON MEDIEVAL FLOORS IN ROME."

[12] P. Prusinkiewicz and A. Lindenmayer, "Graphical modelling using L-systems," in *The Algorithmic Beauty of Plants*, 1990, pp. 1–50.

APPENDICES

The L-system Engine GUI



Program Link

https://github.com/vuralerdogan/L-systems

Codes

```
from tkinter import *
import tkinter as tki
from tkinter import Tk, Button, ttk
import turtle
import sys
import os
from PIL import ImageTk
import numpy as np
import random
import itertools

#-=B AND .=A
MORSE = {'A': 'AA',  'B': 'BAAA', 'C': 'BABA',
        'D': 'BAA',  'E': 'A',    'F': 'AABA',
```

```
    'G': 'BBA',  'H': 'AAAA', 'I': 'AA',
    'J': 'ABBB', 'K': 'BAB',  'L': 'ABAA',
    'M': 'BB',   'N': 'BA',   'O': 'BBB',
    'P': 'ABBA', 'Q': 'BBAB', 'R': 'ABA',
    'S': 'AAA',  'T': 'B',    'U': 'AAB',
    'V': 'AAAB', 'W': 'ABB',  'X': 'BAAB',
    'Y': 'BABB', 'Z': 'BBAA',

    '0': 'BBBB', '1': 'ABBBB', '2': 'AABBB',
    '3': 'AAABB', '4': 'AAAAB', '5': 'AAAAA',
    '6': 'BAAAA', '7': 'BBAAA', '8': 'BBBAA',
    '9': 'BBBBA',

    }
#The above dictionary can be converted for general function.
#For example; 'A': uniquestring[0]+uniquestring[0] (in my project, it symbolises 'AA' or 'BB')
#We preferred short way to illustrate the cryption method.
#Therefore, the user can study on various strings or characters by creating own dictionary or codes
#such as 'D', 'X', 'a', '+', '-', '[', ']' etc.


# The morse alpahet translated from http://code.activestate.com/recipes/578407-simple-morse-code-translator-in-python/ to our desired alphabet.
InvMorse = {v: k for k, v in MORSE.items()}
MyGUI = Tk()
MyGUI.title("L-System")
MyGUI.geometry("1350x800+0+0")

MyGUI.configure(background='black')

#frame
frame = tki.Frame(MyGUI, bg='black')
frame.pack(fill='both', expand='yes')

#background panel
image1 = ImageTk.PhotoImage(file="matrix.jpg")
w = image1.width()
h = image1.height()
bgpanel = tki.Label(frame, image=image1)
bgpanel.pack(side='top', fill='both', expand='yes')
bgpanel.image = image1

#Text Area
T = Text(frame, bg="white" , height=15, width=116,undo=True)
scrollbar = Scrollbar(frame)
scrollbar.pack(side=RIGHT, fill=Y)
```

```
scrollbar.config(command=T.yview)
T.config(yscrollcommand=scrollbar.set)
T['font'] = ('consolas', '12')
T.place(x=500,y=550)
T.config(background='black', fg='#67F383')
hcanvsize=510
wcanvsize=810
frame2 = tki.Frame(frame, bg='black', width=810, height=510)
frame2.place(x=500,y=5)
cv = turtle.ScrolledCanvas(frame2,  canvwidth=wcanvsize, canvheight=hcanvsize)
cv.config(background="black")
cv.place(x=0, y=0)
screen = turtle.TurtleScreen(cv)
screen.bgcolor('black')
screen.delay(0)
t = turtle.RawTurtle(screen)
cv.config(width=800,height=500)
cv.pack(side=LEFT,expand=True,fill=BOTH)


#ENTRIES

#axiom settings
axiom=Label(frame, text="Axiom:")
axiom.place(x=22, y=20)
axe = Entry(frame, bd=2)
axe.place(x=82, y=15)
axe.config(bg="black", fg="#00ff00")
axiom.config(bg="black",fg="#00ff00")
axiom.config(font=('Comic Sans MS',13,"bold"))

#rule1 settings
label1=Label(frame, text="Rule1:")
label1.place(x=22 ,y=50)
label1.config(bg="black",fg="#00ff00")
label1.config(font=('Comic Sans MS',13,"bold"))
rle1 = Entry(frame, bd=2, width=10)
rle1.place(x=82, y=45)
rle1.config(bg="black", fg="#00ff00")

label1To=Label(frame, text="to:")
label1To.place(x=189, y=50)
label1To.config(bg="black",fg="#00ff00")
label1To.config(font=('Comic Sans MS',13,"bold"))
rlecon1 = Entry(frame, bd=2)
```

```
rlecon1.place(x=220, y=45)
rlecon1.config(bg="black", fg="#00ff00")



#rule2 settings
label2=Label(frame, text="Rule2:")
label2.place(x=22 ,y=80)
label2.config(bg="black",fg="#00ff00")
label2.config(font=('Comic Sans MS',13,"bold"))
rle2 = Entry(frame, bd=2, width=10)
rle2.place(x=82, y=75)
rle2.config(bg="black", fg="#00ff00")
label2To=Label(frame, text="to:")
label2To.place(x=189, y=80)
label2To.config(bg="black",fg="#00ff00")
label2To.config(font=('Comic Sans MS',13,"bold"))
rlecon2 = Entry(frame, bd=2)
rlecon2.place(x=220, y=75)
rlecon2.config(bg="black", fg="#00ff00")

#rule 3 settings
label3=Label(frame, text="Rule3:")
label3.place(x=22 ,y=110)
label3.config(bg="black",fg="#00ff00")
label3.config(font=('Comic Sans MS',13,"bold"))
rle3 = Entry(frame, bd=2, width=10)
rle3.place(x=82, y=105)
rle3.config(bg="black", fg="#00ff00")
label3To=Label(frame, text="to:")
label3To.place(x=189, y=110)
label3To.config(bg="black",fg="#00ff00")
label3To.config(font=('Comic Sans MS',13,"bold"))
rlecon3 = Entry(frame, bd=2)
rlecon3.place(x=220, y=105)
rlecon3.config(bg="black", fg="#00ff00")



#Generation settings
gen3=Label(frame, text="Level:")
gen3.place(x=22, y=140)
gen3.config(bg="black",fg="#00ff00")
gen3.config(font=('Comic Sans MS',13,"bold"))
gen = Entry(frame, bd=2, width=5)
gen.config(bg="black", fg="#00ff00")
```

```
gen.place(x=82, y=135)

#segment settings
segmentLabel=Label(frame, text="Segment:")
segmentLabel.place(x=10, y=170)
segmentLabel.config(bg="black",fg="#00ff00")
segmentLabel.config(font=('Comic Sans MS',13,"bold"))
segment = Entry(frame, bd=2, width=8)
segment.place(x=82, y=165)
segment.config(bg="black", fg="#00ff00")

#Angle settings
angleLabel=Label(frame, text="Angle:")
angleLabel.place(x=22, y=200)
angleLabel.config(bg="black",fg="#00ff00")
angleLabel.config(font=('Comic Sans MS',13,"bold"))
angle = Entry(frame, bd=2, width=8)
angle.place(x=82, y=195)
angle.config(bg="black", fg="#00ff00")

#name
Vural=Label(frame, text="Vural Erdogan")
Vural.place(x=50 ,y=750)
Vural.config(bg="black",fg="cyan")
Vural.config(font=('Comic Sans MS',20))

#Functions

#compare function helps to generate previous L system.
def compare():
    # clearing boxes

    text=combo.get()
    axe.delete(0, END)
    rle1.delete(0, END)
    rlecon1.delete(0, END)
    rle2.delete(0, END)
    rlecon2.delete(0, END)
    rle3.delete(0, END)
    rlecon3.delete(0, END)
    gen.delete(0, END)
    segment.delete(0, END)
    angle.delete(0, END)

    #calling ready methods
```

```
if (text== 'Sierpinsky'):
    axe.insert(0, 'F')
    rle1.insert(0, 'F')
    rlecon1.insert(0,'Y-F-Y')
    rle2.insert(0, 'Y')
    rlecon2.insert(0, 'F+Y+F')
    segment.insert(0, '5')
    angle.insert(0, '60')
    gen.insert(0, '5')
if (text=='Koch'):
    axe.insert(0, 'F')
    rle1.insert(0, 'F')
    rlecon1.insert(0, 'F-F++F-F')
    gen.insert(0, '5')
    segment.insert(0, '5')
    angle.insert(0, '60')
if (text=='Square'):
    axe.insert(0, 'F')
    rle1.insert(0, 'F')
    rlecon1.insert(0, 'F-F+F+F-F')
    gen.insert(0, '5')
    segment.insert(0, '5')
    angle.insert(0, '90')
if (text=='Plant'):
    axe.insert(0, 'Y')
    rle1.insert(0, 'Y')
    rlecon1.insert(0, 'F-[[Y]+Y]+F[+FY]-Y')
    rle2.insert(0, 'F')
    rlecon2.insert(0, 'FF')
    gen.insert(0, '5')
    segment.insert(0, '5')
    angle.insert(0, '25')
if (text=='Fibonacci'):
    axe.insert(0, 'Y')
    rle1.insert(0, 'F')
    rlecon1.insert(0, 'FF')
    rle2.insert(0, 'Y')
    rlecon2.insert(0, 'F[-Y]+Y')
    gen.insert(0, '5')
    segment.insert(0, '5')
    angle.insert(0, '30')
if (text=='Dcurve'):
    axe.insert(0, 'FV')
    rle1.insert(0, 'V')
```

```
        rlecon1.insert(0, 'V+YF')
        rle2.insert(0, 'Y')
        rlecon2.insert(0, 'FV-Y')
        gen.insert(0, '5')
        segment.insert(0, '5')
        angle.insert(0, '90')
    if (text=='ArrowPlant'):
        axe.insert(0, 'X')
        rle1.insert(0, 'X')
        rlecon1.insert(0, 'F[+X][-X]FX')
        rle2.insert(0, 'F')
        rlecon2.insert(0, 'FF')
        gen.insert(0, '7')
        segment.insert(0, '3')
        angle.insert(0, '25.7')
    if (text=='Plant2'):
        axe.insert(0, 'X')
        rle1.insert(0, 'X')
        rlecon1.insert(0, 'F[+X]F[-X]+X')
        rle2.insert(0, 'F')
        rlecon2.insert(0, 'FF')
        gen.insert(0, '7')
        segment.insert(0, '4')
        angle.insert(0, '20')
    if (text=='CoolPlant'):
        axe.insert(0, 'F')
        rle1.insert(0, 'F')
        rlecon1.insert(0, 'FF-[-F+F+F]+[+F-F-F]')
        gen.insert(0, '4')
        segment.insert(0, '4')
        angle.insert(0, '22.5')
    if (text=='CurlyPlant'):
        axe.insert(0, 'F')
        rle1.insert(0, 'F')
        rlecon1.insert(0, 'F[+F]F[-F][F]')
        gen.insert(0, '4')
        segment.insert(0, '5')
        angle.insert(0, '20')

#change background of Turtle screen
def changecolour():
    screen.bgcolor(str(combo2.get()))

#restart the GUI
def restart():
```

```
    gui = sys.executable
    os.execl(gui, gui, * sys.argv)


#rulefinder finds the rule by obtaining from last generations : Glast and Glast-1
def rulefinder():
    global externalrul1
    global externalrul2

    # The size of your "expanding permutations"
    if len(rlecon1.get())>=len(rlecon2.get()):
        n = len(rlecon1.get())+1
    elif len(rlecon2.get()) > len(rlecon1.get()):
        n = len(rlecon2.get())+1

    res = []
    for i in range(1, n):
        res.extend(map(''.join, list(itertools.product(uniquestring1, repeat=i))))
    print(res)

    for y in range(0,len(res)):
        rul2 = res[y]
        for x in  range(0,len(res)):
                    rul1 = res[x]
                    code = ""
                    for cha in Genx1:

                        if cha==uniquestring1[0]:
                            code +=codereplacer(rul1)
                        elif cha==uniquestring1[1]:
                            code +=codereplacer(rul2)
                    print(code)
                    print(uniquestring1[0],rul1)
                    print(uniquestring1[1],rul2)
                    print(LastString)
                    if code == LastString:
                        axiom1=uniquestring1[0]
                        axiom2=uniquestring1[1]
                        externalrul1=rul1
                        externalrul2=rul2
                        print('rules are found')
                        print("First RULE:", uniquestring1[0], rul1)
                        print("Second RULE:", uniquestring1[1], rul2)
                        findsubgeneration(code, axiom1, externalrul1, axiom2, externalrul2)
```

```
                return

#findsubgeneration finds previous generations untill finds the axiom of rule.
def findsubgeneration(generation, ax1, rul1, ax2, rul2):
    nextgeneration = ''
    i = 0
    state1 = 0

    while (i < (len(generation))):

        if generation[i:i + len(rul1)] == rul1:
            nextgeneration = nextgeneration + ax1
            i = i + len(rul1)
            state1=1
        elif generation[i:i + len(rul2)] == rul2:
            nextgeneration = nextgeneration + ax2
            i = i + len(rul2)
            state1=2
        else:
            if  state1==1  :
                nextgeneration = nextgeneration[:-len(ax1)]+''
                i = i - len(rul1)

            elif state1==2  :
                nextgeneration = nextgeneration[:-len(ax2)]+''
                i = i - len(rul2)
            swaprules = rul1
            rul1 = rul2
            rul2 = swaprules
            swapax = ax1
            ax1 = ax2
            ax2= swapax
            state1 = 0

    if (len(rul1) > len(nextgeneration)) or (len(rul2) > len(nextgeneration)):
        print ('Generation number:'+ str(z))
        print('Axoim:' + nextgeneration)
        return nextgeneration
    else:

      print('Generations:'+ nextgeneration)
      findsubgeneration(nextgeneration, ax1, rul1, ax2, rul2)

#codereplacer adds new character to old string
def codereplacer(rules):
```

```
    NewString=""
    NewString = NewString + rules
    return NewString

#cyphersolver reveals the cypher by comparing hidden message with L-system generated message.
def cyphersolver(word1,word2):

    word1 = ''.join(x for x, y in zip(word1, word2) if x != y)
    print("Message has been found:" + word1)
    return word1

#createmessage enables an user to enter a message and hide it.
def createmessage(allchas):


    global cypher
    cypher=""
    s=''
    msg = input('SECRET MESSAGE: ')
    randomarray=[]
    for char in msg:
        print(MORSE[char.upper()])
        s= s + MORSE[char.upper()]
        s= s + '.'

    print ('Translation:' + s)
    print('message is encrypting...Please wait...')
    codelist = list(allchas)
    x=0
    randomarray = random.sample(range(1, len(codelist) - len(Genx1) - len(Genx2)), len(s))


    while (x < len(s)):
        randomarray.sort()

        if (s[x]=='A' ) :
            if (codelist[randomarray[x]]=='B'):

                codelist[randomarray[x]] = s[x]
                x = x + 1
            else:
                randomarray[x] = randomarray[x] + 1
        elif (s[x]=='B' ) :
            if (codelist[randomarray[x]]=='A'):
                codelist[randomarray[x]] = s[x]
```

```
            x = x + 1
        else:
            randomarray[x] = randomarray[x] + 1
    elif (s[x]=='.'):
        codelist[randomarray[x]]=' '
        x= x + 1



    allchas = "".join(codelist)
    cypher=allchas
    T.insert(END,'Message is uploaded:'+ cypher)
    T.yview_pickplace("end")


#finds message in the encrypted code.
def findmessage():
    x=''
    thesecretcode =""
    secretarray=[]
    thesecretcode = cyphersolver(cypher, Message)
    secretarray = thesecretcode.split()
    for char in secretarray:
        print(InvMorse[char.upper()])
        x = x + InvMorse[char.upper()]

    print (x)

# a simple word adder
def sum(Wordsum,Generation):

    return Wordsum + Generation

#decypher reveals the details of generations.
def decypher():
    global uniquecount2, uniquecount1
    #1
    uniquecount1 = len(uniquestring1)
    print('Gen1:')
    print(*Genx1)
    n=""
    n +='Unique cha counts:' + str(uniquecount1)+ "\n"
    n += str(uniquestring1) +"\n"
    for cha in uniquestring1:
        n += cha + ':' + str(Genx1.count(cha)) + "\n"
```

```
    T.insert(END, n)
    #2
    uniquecount2 = len(uniquestring2)
    print('Gen2:')
    print(*Genx2)
    m=""
    m +='Unique cha counts:' + str(uniquecount2) + "\n"
    m += str(uniquestring2) + "\n"
    for cha2 in uniquestring2:
        m += cha2 + ':' + str(Genx2.count(cha2)) + "\n"
    T.insert(END, m)
    T.yview_pickplace("end")


#convert function changes the characters to the rules. E.g. A ---> AB
def Convert(String):
    convertedString = ""
    if String == rle1.get() :
        rule1 = rlecon1.get()
        convertedString = rule1
    elif String == rle2.get():
        rule2 = rlecon2.get()
        convertedString = rule2
    elif String == rle3.get():
        rule3 = rlecon3.get()
        convertedString = rule3
    else: convertedString=String
    return convertedString

#string adder
def AddingToNewString(changingString):
    NewString = ""

    for character in changingString:
        NewString = NewString + Convert(character)
    return NewString

#creator function creates L-system strings.
def CreatorFunction(Generation, axiom):

    global LastString
    global Message
    global Genx1, StringCount1, uniquestring1
    global Genx2, StringCount2, uniquestring2
    Message = ""
```

```
    LastString = ""

    for x in range(1, Generation+1):

        LastString = AddingToNewString(axiom)
        axiom = LastString
        Message=sum(Message,LastString)
        s ="G" + str(x) + ":" + LastString + "\n" + 'Character Count:' + str(len(LastString)) + "\n"

        if x==Generation-1:
            Genx1=list(LastString)
            uniquestring1 = list(set(LastString))

        if x==Generation:
            Genx2=list(LastString)
            uniquestring2 = list(set(LastString))

        T.insert(END, s)
    return LastString


def drawFunction(rules, angle, segment, Turt, canvsizew, canvsizeh):

        location=[]
        Turt.hideturtle()
        Turt.setheading(90)
        Turt.back(1)
        Turt.speed(0)

        #turtle colours arrays
        colours=["#BFFF00", "#00CC00", "white", "#FFFF00"]
        set2=['green3','green2', '#4CC417']
        set1=['#4CC417','yellow2']
        green=['yellow2']

        y=0
        x=0
        for cha in rules:
            if  abs(2*Turt.ycor())> canvsizeh:
                canvsizeh +=300
                cv.reset(canvheight=canvsizeh)
                cv.yview_moveto(-(Turt.ycor()))


            if abs(2 * Turt.xcor())>canvsizew:
```

```
            canvsizew += 300
            cv.reset(canvwidth=canvsizew)
            cv.xview_moveto((Turt.xcor()))

        if cha == 'F':
            Turt.forward(segment)
            x = x + 1
            x = x % 3
            Turt.forward(segment)
            Turt.color(set2[x])
        elif cha=='Y':
            y = y + 1
            y = y % 2
            Turt.forward(segment)
            Turt.color(set1[y])
        elif cha == 'X':
            Turt.backward(segment)
        elif cha == '[':
            location.append((Turt.heading(), Turt.pos()))
        elif cha == ']':
            heading, position = location.pop()
            Turt.penup()
            Turt.goto(position)
            Turt.setheading(heading)
            Turt.pendown()
        elif cha == '+':
            Turt.right(angle)
        elif cha == '-':
            Turt.left(angle)

# 'AddingtoNewString', 'CreatorFunction', 'DrawFunction' and 'Convert' inpired by below website.
# http://interactivepython.org/courselib/static/thinkcspy/Strings/TurtlesandStringsandLSystems.html

#Buttons
print1 = Button(frame, text="Print",borderwidth=5, command=lambda: CreatorFunction(int(gen.get()), str(axe.get())))
print1.place(x=35, y=250)
print1.config(highlightbackground='black')

quit1 = Button(frame, text="Exit", command=lambda:quit())
quit1.place(x=220, y=250)
quit1.config(highlightbackground='black')

clear2 = Button(frame, text="Clear Text", command=lambda: T.delete('1.0', END))
clear2.place(x=390, y=750)
clear2.config(highlightbackground='black')
```

```
clear1 = Button(frame, text="Clear Screen", command=lambda: screen.resetscreen())
clear1.place(x=380, y=490)
clear1.config(highlightbackground='black')

Draw1 = Button(frame, text="Draw!", command=lambda:drawFunction(CreatorFunction(int(gen.get()), str(axe.get())), float(angle.get()), float(segment.get()),t,
wcanvsize, hcanvsize))
Draw1.place(x=35, y=285)
Draw1.config(highlightbackground='black')

Restart=Button(frame, text="Restart", command=restart)
Restart.place(x=140,y=250)
Restart.config(highlightbackground='black')

combo=ttk.Combobox(MyGUI)
combo.place(x=35, y= 330)
combo['values']=('Koch','Plant','Square','Sierpinsky','Fibonacci','Dcurve', 'ArrowPlant', 'Plant2', 'CoolPlant','CurlyPlant')
combo.current(1)

combo2=ttk.Combobox(MyGUI)
combo2.place(x=35, y= 360)
combo2['values']=('black','white','gray16','gray71','MediumOrchid3','light cyan','wheat1','azure2')
combo2.current(0)



Generate=Button(frame, text="Generate!", command=lambda: compare())
Generate.place(x=280,y=250)
Generate.config(highlightbackground='black')

changebg=Button(frame, text="Change Colour", command=lambda: changecolour())
changebg.place(x=280,y=280)
changebg.config(highlightbackground='black')

Detail=Button(frame, text="Detail", command=lambda: decypher())
Detail.place(x=280,y=310)
Detail.config(highlightbackground='black')

Solve=Button(frame, text="Solve!", command=lambda: rulefinder())
Solve.place(x=280,y=340)
Solve.config(highlightbackground='black')

Upload=Button(frame, text="Encrypt a Message", command=lambda: createmessage(Message))
Upload.place(x=280,y=370)
Upload.config(highlightbackground='black')
```

```
FindMessage=Button(frame, text="Decrypt!", command=lambda: findmessage())
FindMessage.place(x=280,y=400)
FindMessage.config(highlightbackground='black')



MyGUI.mainloop()
```