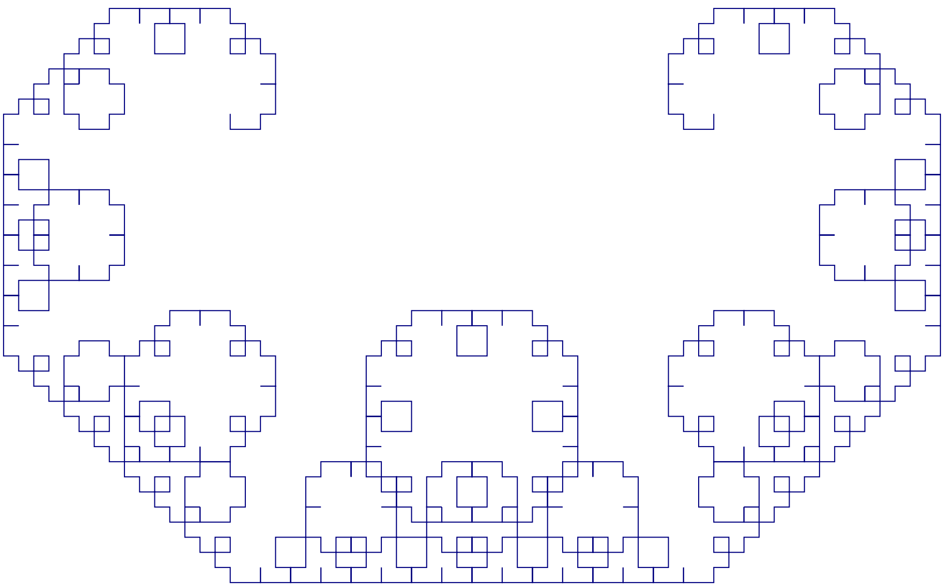
Lindenmayer's Garden

Scuola d'Arti e Mestieri di Trevano (SAMT)
Documentation

Paolo Bettelini



I would like to thank my professor Geo Petrini for this amazing project.

Contents

1 Introduction **4**
1.1 Abstract . . . 4
1.2 Information . . . 4

2 L-systems **5**
2.1 Formal grammar . . . 5
2.2 Context-free grammar . . . 5
2.3 Fractals . . . 5
2.4 Definition . . . 5
2.5 Semantics . . . 5
2.6 Turing Completeness of W . . . 7
2.7 Examples . . . 8

3 Analysis **9**
3.1 Requirements . . . 9
3.2 Use Cases . . . 11
3.3 GUI Design . . . 12

4 Technologies **13**
4.1 Rust . . . 13
4.2 GTK4 . . . 13
4.3 Cairo . . . 13

5 Planning **14**
5.1 Initial Gantt Chart . . . 14
5.2 Final Gantt Chart . . . 15

6 Compilation and usage **16**

7 Implementation **17**
7.1 Textual configuration . . . 17
7.1.1 Sintax . . . 17
7.1.2 Examples . . . 20
7.2 Meval crate . . . 20
7.3 Separation of concerns . . . 20
7.3.1 lindenmayer-engine . . . 21
7.3.2 lindenmayer-renderer . . . 21
7.3.3 lindenmayer-renderer-cairo . . . 23
7.3.4 lindenmayer-parser . . . 23
7.3.5 lindenmayer-gui . . . 23
7.4 Graphical User Interface . . . 24
7.5 Error Handling . . . 29
7.6 Project Structure . . . 30
7.7 Logging system . . . 31
7.8 Dependencies . . . 32

8 Tips and tricks **33**
8.1 Random values . . . 33
8.2 Natural growth . . . 33
8.3 Stochastic grammar . . . 33

9 Tests **34**
9.1 Testing protocol . . . 34
9.2 Test results . . . 39

10 Conclusions **40**
10.1 Future development . . . 40
10.2 Personal conclusions . . . 40

1 Introduction

1.1 Abstract

Lindenmayer Systems, or commonly L-systems, are formal grammars used for generating complex patterns and structures. They were introduced in 1968 by Aristid Lindenmayer, a Hungarian theoretical biologist and botanist. These systems are exceptionally good at representing natural growth of trees, algae, bushes and such. L-systems can also be used to draw any sort of fractal, with a variable level of detail. The goal of this project is to stimulate creativity by creating a L-system playground. The program provides a sophisticated deterministic context-free grammar with advanced drawing features, stochastic behavior and an animation system.
Furthermore, L-systems are Turing-complete, meaning that anything can be computed with them. Any geometrical shape can be mathematically modelled with these systems.
There are many programs that can be used to render L-systems, but none of them approaches the topic the same way this project is intended to. The philosophy behind this implementation is to let the user experiment with arbitrary mathematical expressions, and be as flexible as possible.

1.2 Information

This is a project of the Scuola Arti e Mestieri di Trevano (SAMT) under the following circumstances

- **Section**: Computer Science
- **Year:** Fourth
- **Class:** LPI
- **Supervisor:** Geo Petrini
- **Expert:** Gionata Genazzi
- **Title:** Lindenmayer's Garden
- **Start date**: 2023-05-02
- **Deadline**: 2023-05-26

and the following requirements

- **Documentation**: a full documentation of the work done
- **Diary**: constant changelog for each working session
- **Source code**: source code of the project

All the source code and documents can be found at http://gitsam.cpt.local/lavoro_finale_lpi_2023/giardino-lindenmayer [9].

2 L-systems

2.1 Formal grammar

In theory of computation and formal language theory, a formal grammar[7] is a system describing a language of a given alphabet using a set of production rules.

2.2 Context-free grammar

A context-free grammar[4] is a formal grammar where the production rules for a given symbol are always the same. For each symbol, the production rule for that symbol is independent of the other symbols in the string or any other external factor.

2.3 Fractals

A fractal[8] is a geometrical shape that is self-similar and contains an arbitrary large amount of detail.

2.4 Definition

An *L-system*[12] or *Lindenmayer system* is a type of formal grammar which can be used to draw fractals and other shapes.
The grammar of an L-system may vary. The language of such systems may be generated using grammars such as context-free grammars, stochastic grammars, sensitive grammars and many more.

An L-system may be expressed as a tuple $G = (\Sigma, \omega, P)$ where Σ is the alphabet, $\omega \in \Sigma^*$ is the *initial string*, *axiom* or *initiator* and $P \subseteq \Sigma \times \Sigma^*$ is a set of production rules.

Any L-system can produce a string by applying its productions rules P to every symbol of a string. Let this operation be defined as $\pi : P \times \Sigma^* \to \Sigma^*$. The result of $\pi(P, v)$ will be a copy of v, but if $\exists\, (a, b) \in P \mid v$ contains a, then the symbol a in v will be replaced by the string b.

In general, this operation can be applied iteratively starting with the axiom ω.

$$\begin{aligned}
G_0 &= \omega \\
G_1 &= \pi(P, \omega) \\
G_2 &= \pi(P, \pi(P, \omega)) \\
G_3 &= \pi(P, \pi(P, \pi(P, \omega))) \\
&\vdots \\
G_n &= \pi(P, G_{n-1})
\end{aligned}$$

where $n \in \mathbb{N}$ is the number of iterations.

2.5 Semantics

In order to give meaning to $v \in \Sigma^*$ I first define my own structure, W, which is an extension of a classical L-system $G = (\Sigma, \omega, P)$.
The language of W is generated using a deterministic context-free grammar, but the semantics can behave in a stochastic manner.

W is a tuple (G, V, O, c) where $V = \mathbb{R}^k$ is a tuple of *variables*, O is a set of *operations*, and c is a tuple of *configurations*.

The set I is defined as $I = \mathbb{N} \times \Sigma^*$.

The tuple c contains the following values:

1. `iter` $\in \mathbb{N}$.
2. `initial rotation` $\in [0; 2\pi]$.
3. `initial position` $\in \mathbb{R}^2$.
4. `initial thickness` $\in \mathbb{R}$.
5. `initial color` $\in \mathbb{R}^4$.
6. `background color` $\in \mathbb{R}^4$.
7. `canvas` $\in \mathbb{R}^2$.
8. `injections` $\in I^n$.

Let Expr be an algebraic data type[1] representing any mathematical expression, including stochastic functions.

Let Ξ_S be an algebraic data type defined as

$$\begin{aligned}\Xi_S = \;&\text{Forward(Expr)} \quad\mid\quad \text{Jump(Expr)} \quad\mid\quad \text{Dot(Expr)} \quad\mid\\ &\text{Rotate(Expr)} \quad\mid\quad \text{Thickness(Expr)} \quad\mid\quad \text{Color}(\mathbb{R}^4) \quad\mid\\ &\text{Ignore(Expr)} \quad\mid\quad \text{Push()} \quad\mid\quad \text{Pop()} \quad\mid\\ &\text{Update}(\mathbb{N}, \text{Expr})\end{aligned}$$

This data type is based and acts on a mutable globally shared state S. There exists a function $\text{eval} : \Xi_S \to \mathbb{R}$ that evaluates the value of Ξ_S.
Any Expr type may use variables in the state S. The state S initially contains every variable in V. The evaluation of $\Xi_S \equiv \text{Update}(i, j)$ will update the value i-th value of V in S to $\text{eval}(j)$.
S also acts as a stack data structure. Whenever $\Xi_S \equiv \text{Pop}()$, the state of S becomes the state last of S before the last instance where $\Xi_S \equiv \text{Push}()$.
Every other constructor of Ξ_S, except for $\Xi_S \equiv \text{Ignore}(j)$, has a graphical meaning.

The set O is defined $O \subseteq \Sigma \times \mathcal{P}([\Xi])$ where $[\Xi]$ denotes all possible values for Ξ.

Let $\phi : I \times \Sigma^* \to \Sigma^*$ be the `inject` function. The function $\phi(i, v)$ where $i = (\gamma, \zeta)$ will construct a string by injecting the string ζ at the position γ of the string v if $\gamma \leq |v|$.

Let $\Phi : I^n \times \Sigma^* \to \Sigma^*$ be the `multi-inject` function. The function $\Phi(\Psi, v)$ will construct the string v_n, starting from $v_0 = v$, by applying $v_i = \phi(\psi, v_{i-1})$ for all values ψ in Ψ considering the order in Ψ.

Let $\theta : \mathbb{N} \times \mathbb{N} \times \Sigma^* \to \Sigma^*$ be the `ignore` function. The function $\theta(i, l, v)$ will construct a string that is a copy of v but replacing the symbols from the index $\min(i, |v|)$ to $\min(i + l, |v|)$ of v with the empty string λ.

Let $\Theta : \Sigma^* \to \Sigma^*$ be the `multi-ignore` function. The function $\Theta(v)$ will construct a string, starting from $v_0 = v$, by applying $v_i = \theta(i, l, v_{i-1})$ as long as $\exists\, (s, o) \in O \,|\, v$ contains $s \wedge o \equiv$ $\text{Ignore}(j)$ where i is the position of s in v and $l = \text{eval(j)}$.

The structure W can also produce a string $W_n \in \Sigma^*$ which is defined as

$$W_n = \Theta(\Phi(\text{injections}, G_n))$$

where $n \in \mathbb{N}$ is the number of iterations. The application of $\text{eval}(v)\, \forall v \in W_{\text{iter}}$ will produce a graphical drawing.

2.6 Turing Completeness of W

Theorem 1. *The system $W = (G, V, O, c)$ is Turing-complete.*

Proof. Since the Lindenmayer system $G = (\Sigma, \omega, P)$ is Turing-complete, we can construct W such that every G_n can be generated by W_n.
Let $W = (G, V, O, c)$ where $V = \emptyset$, $O = \emptyset$ and `injections` $= \emptyset$.
As a corollary, we have:

$$\text{injections} = \emptyset \implies \Phi(\text{injections}, v) \equiv \Phi(\emptyset, v) \equiv v$$

and

$$O = \emptyset \implies \Theta(v) \equiv v$$

Using this properties, we can simplify the expression for W_n

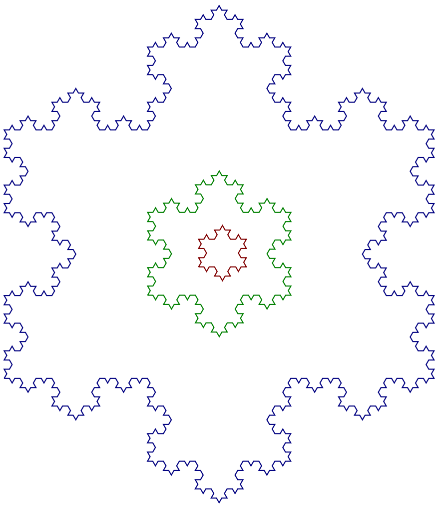
$$W_n = \Theta(\Phi(\text{injections}, G_n)) = G_n$$

Thus, W is Turing-complete. ■

2.7 Examples

The following section shows some examples of L-systems fractal drawings.

Note that by convention, `F` means draw a line, `+` and `-` mean rotate by ±angle, `[` and `]` mean `push` and `pop` on the stack respectively.

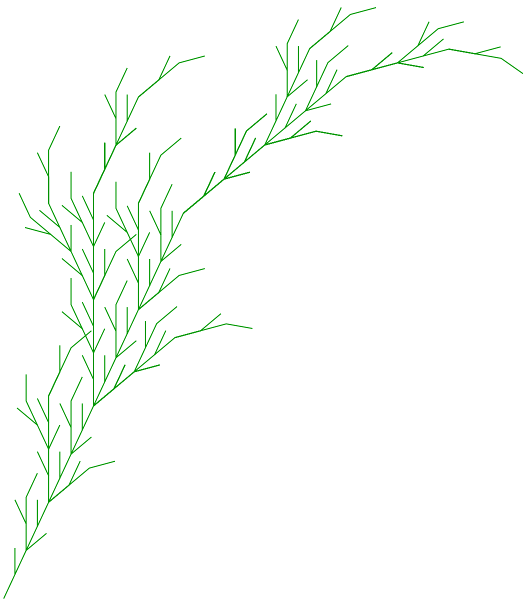


$G = (\Sigma, \omega, P)$ where

- $\Sigma = \{'F', '+', '-'\}$
- $\omega = F + +F + +F$
- $P = \{('F', F - F + +F - F)\}$

and angle $= \frac{\pi}{3}$

Figure 1: Koch curve for $n = 4, 3, 2$.



$G = (\Sigma, \omega, P)$ where

- $\Sigma = \{'F', '+', '-', '[', ']'\}$
- $\omega = F$
- $P = \{('F', F[+F]F[-F])\}$

and angle $= \frac{5\pi}{36}$

Figure 2: Plant for $n = 4$.

3 Analysis

3.1 Requirements

Req-00	
Name	Functionality
Priority	1
Version	1.0
Notes	none
Description	The program must be able to render on a GUI any L-system, within the computational power of the host.

Req-01	
Name	Grammar
Priority	1
Version	1.0
Notes	none
Description	The grammar must support the basic operations of turtle graphics[16] for D0l-systems.
Subrequirements	
Req-01__0	The grammar must be able to set the color of lines.
Req-01__1	The grammar must be able to set thickness of lines.
Req-01__2	The grammar must be able to ignore the behavior of any given character at any position.

Req-02	
Name	Stochastic Behavior
Priority	1
Version	1.0
Notes	none
Description	Variables, such as line lengths or rotation angles, must be able to follow stochastic behavior.
Subrequirements	
Req-02__0	Stochasic behavior must be based on a deterministic seed.

Req-03	
Name	Animations
Priority	2
Version	1.0
Notes	This implies the use of a variable `frames` and `depth`
Description	L-systems must be able to support optional animations.
Subrequirements	
Req-03__0	Variables can be dependent on the current animation frame index.
Req-03__1	Variables can be dependent on the current stack depth.

Req-04	
Name	Graphical User Interface
Priority	1
Version	1.0
Notes	none
Description	The GUI must contain a canvas on which the fractals are rendered.
Subrequirements	
Req-04_0	The GUI must provide functionality to modify every rule and aspect of a given L-system.
Req-04_1	The GUI must provide functionality to import a file, representing L-systems.
Req-04_2	The GUGUII must provide functionality to export a file, representing L-systems.
Req-04_3	The GUI must have an animation playback.

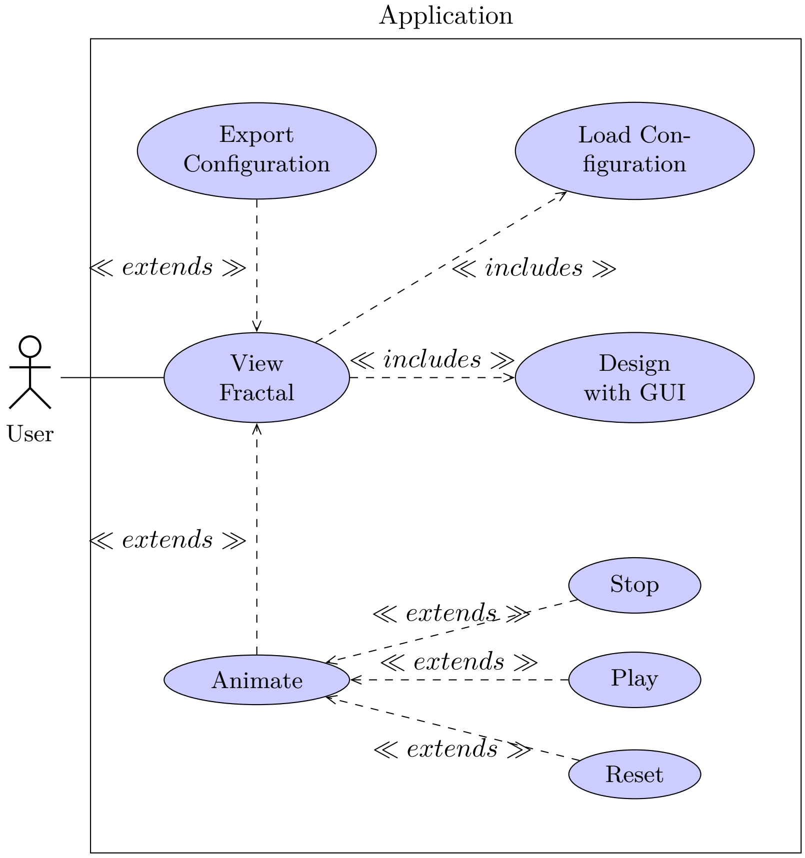
Req-05	
Name	Logging
Priority	1
Version	1.0
Notes	none
Description	The program must provide logging capabilities.

Req-06	
Name	Error Checking
Priority	1
Version	1.0
Notes	none
Description	Erroneous input from the user must be handled correctly with appropriate error messages.
Subrequirements	
Req-06_0	The user must be provided with the information of where he has made a mistake.

3.2 Use Cases

The following diagram shows the use cases of the program. In order to view a fractal, the user can either import a configuration file or design a fractal with the graphical editor.
When the application starts, an empty configuration is imported by default. The user may also design his fractal with the editor starting from any configuration.

The user is also able to export the configuration to a file.
The fractal itself can be animated.



3.3 GUI Design

The design is very simple and consists of a single page. The interface has an editor (right side) and a canvas where the image is displayed. Beneath the canvas is an animation playback to control the animation system along with some information.

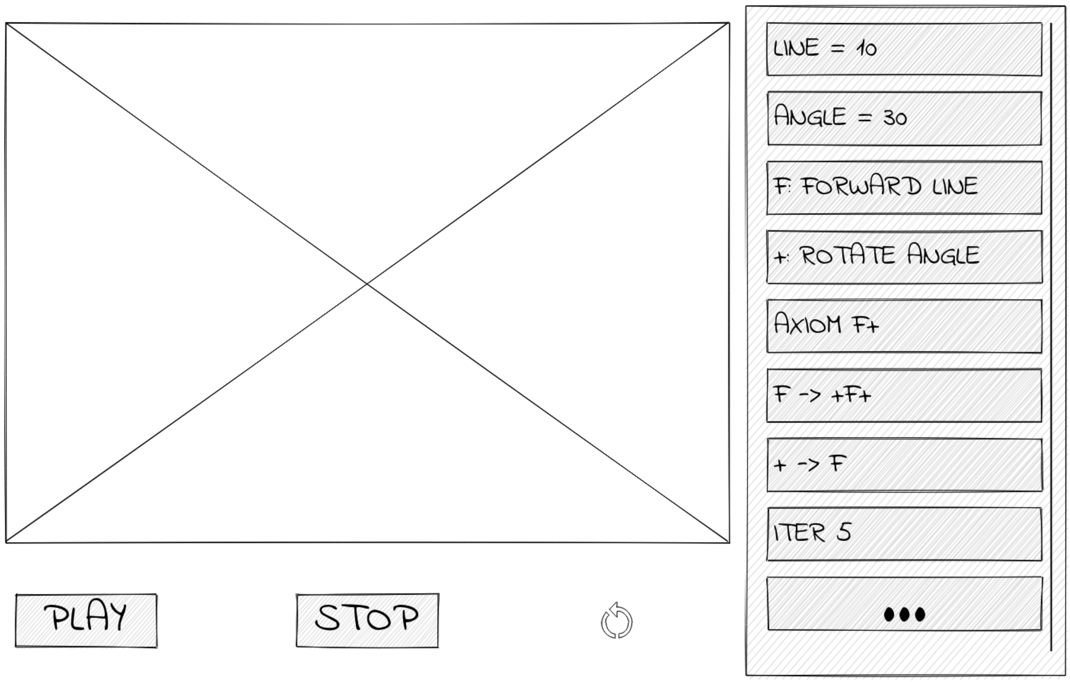


Figure 3: GUI Design sketch

4 Technologies

4.1 Rust

Rust[15] is a generic compiled programming language. The code is compiled using LLVM to machine code and its speed is comparable to C and C++. Rust is the first programming language to guarantee memory safety; memory is not manually freed nor garbage collected. It is not possible to dereference a null pointer, cause memory segfauls, core dumps and memory leaks. Code that could cause undefined behavior can still be written, but it is strictly bounded in blocks where the compiler is relaxed. This relaxation implies that the language is also low-level. Another key feature to the performance of Rust is zero cost abstraction, which means that generic types and function abstractions are resolved at compile-time. Conditional compilation and compile-time computations are also extensively used.
There are also many features concerning the programming experience, such as advanced metaprogramming and code generation using macros, intelligent compiler, dependency system (Cargo), modern syntax and many tools to ease development.

Note: a Rust *crate* refers to a library. A *feature* is an optional component of library. A *module* is a logical section of a program or library.

4.2 GTK4

GTK (GIMP Toolkit) is a free and open-source widget toolkit for creating graphical user interfaces (GUIs). GTK is written in C and has bindings for many programming languages, including Rust. GTK is widely used in Linux desktop environments and applications, but it also runs on other platforms, including Windows and macOS.

GTK4 was released in December 2020. It introduces several new features and improvements over its predecessor, GTK3.

4.3 Cairo

Cairo[2] is a widely used 2D graphics library. This library is written in C but there exist bindings for almost every language.

5 Planning

5.1 Initial Gantt Chart

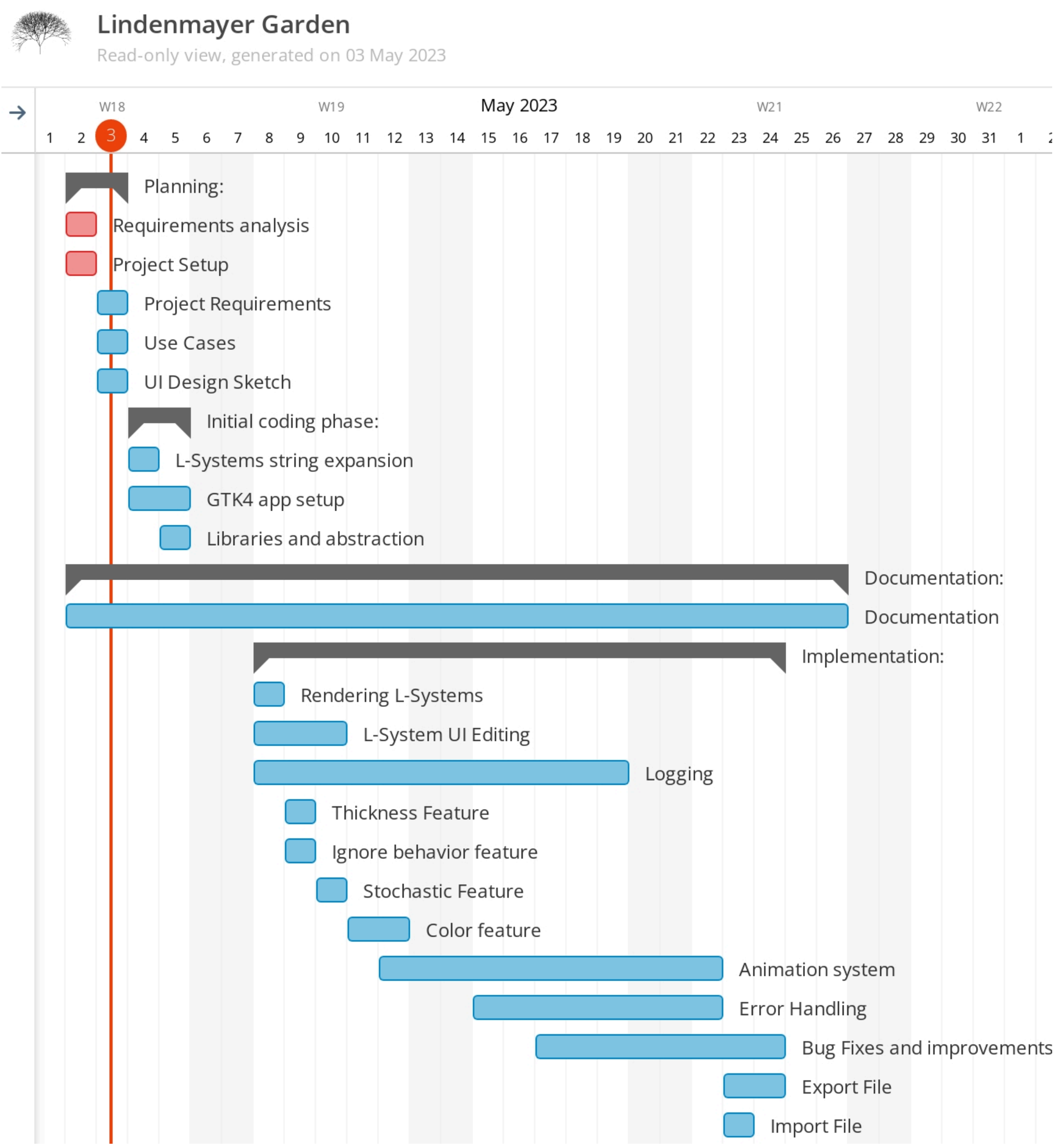


Figure 4: Initial Gantt Chart

I chose the waterfall model with a Gantt chart for my planning.

At the start of the project, I was pretty familiar with what I had to do and how I was going to do it. I tried to estimate the duration of the tasks based on this information. I included a bit of margin towards the end to still have time to solve some eventual problems.

5.2 Final Gantt Chart

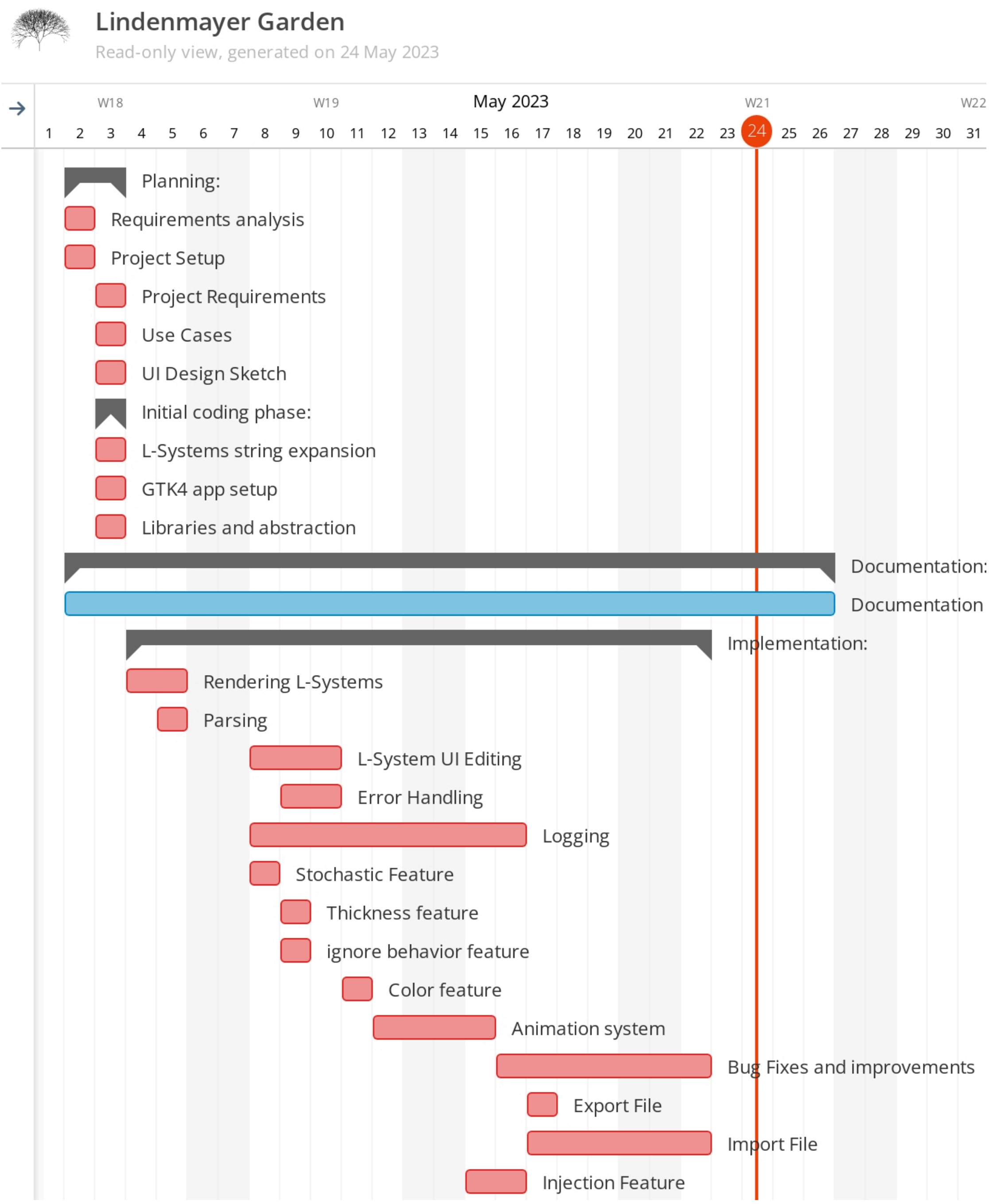


Figure 5: Final Gantt Chart

I had a great start by completing early the whole `Initial Coding Phase` section, which I had overestimated. Thus, I was always 2-3 days early with respect to my initial planning.
In general, almost every task was completed early. The `Error Handling` task was done with the `UI Editing`.

Towards the end of the project, I spent much time implementing minor features and fixing bugs. However, I still finished programming a couple of days early.

6 Compilation and usage

In order to compile the application, you need to have GTK4 installed on your system.

Using pacman:

```
$ pacman -S gtk4 base-devel
```

Using dnf:

```
$ dnf install gtk4-devel gcc
```

Using apt:

```
$ apt install libgtk-4-dev build-essential
```

For other platforms and information, visit this website[10].

The executable can be compiled using the `cargo` package manager, which can be installed at the official website[11].

```
$ cd lindenmayer-gui
$ cargo build --release
```

This will generate an executable (`lindenmayer-gui`) in `./target/release`. In order to make this executable globally available, you can move it into a folder in the $PATH environment variable, such as `/usr/bin`. You may also modify the executable file name to change its invocation name.

```
$ sudo mv target/release/lindenmayer-gui /usr/bin/
```

The executable can now be invoked by just writing

```
$ lindenmayer-gui
```

7 Implementation

7.1 Textual configuration

The L-systems are represented by a textual configuration, where each line indicates a property. The conventional extension for this format is the `.lsys` extension.

There are four types of lines: `configurations`, `variables`, `operations` and `rules`.

7.1.1 Sintax

The configuration files have the following form.

```
[
    <Config>
    |
    <Variable>
    |
    <Operation>
    |
    <Rule>
]*
```

Different types of lines may be written in any order.

A **Configuration** element is defined as one of the following expressions:

```
    axiom <String>
    iter <Integer>
    initial_rot <Float>
    initial_pos <Position>
    initial_thickness <Float>
    initial_color <Color>
    background <Color>
    canvas <Dimension>
    seed <String>
    inject [<Injection>]*
```

If the same configuration command is repeated multiple times, only the last one will be considered.

A **Variable** element is defined as

```
    VAR_NAME = <Number>
```

An Operation element is defined as one of the following expressions:

```
    <Char>: forward <Expression>
    <Char>: jump <Expression>
    <Char>: dot <Expression>
    <Char>: rotate <Expression>
    <Char>: thickness <Expression>
    <Char>: ignore <Expression>
    <Char>: push
    <Char>: pop
    <Char>: color <Color>
    <Char>: <Variable> = <Expression>
```

A **Rule** element is defined as

```
    <Char> -> <String>
```

A **Position** element is defined as a tuple of floating point numbers

```
Float,Float
```

A **Dimension** element is defined as a tuple of integers

```
Integer,Integer
```

An **Injection** element is defined as a tuple of an integer (index) and a string (injection).

```
Integer,String
```

A **Color** element is defined as a subset of strings that represent a valid color in the CSS format[3].

An **Expression** element is a CAS (Computer Algebra System) expression. The expression may use elementary functions and variables. The expression may use the following functions

1. `sqrt`: $\sqrt{x}$. E.g. `sqrt(1)`.
2. `exp`: e^x. E.g. `exp(1)`.
3. `ln`: $\ln(x)$. E.g. `ln(1)`.
4. `abs`: $|x|$. E.g. `abs(1)`.
5. `sin`: $\sin(x)$. E.g. `sin(1)`.
6. `cos`: $\cos(x)$. E.g. `cos(1)`.
7. `tan`: $\tan(x)$. E.g. `tan(1)`.
8. `asin`: $\arcsin(x)$. E.g. `asin(1)`.
9. `acos`: $\arccos(x)$. E.g. `acos(1)`.
10. `atan`: $\arctan(x)$. E.g. `atan(1)`.
11. `sinh`: $\sinh(x)$. E.g. `sinh(1)`.
12. `cosh`: $\cosh(x)$. E.g. `cosh(1)`.
13. `tanh`: $\tanh(x)$. E.g. `tanh(1)`.
14. `asinh`: $\mathrm{asinh}(x)$. E.g. `asinh(1)`.
15. `acosh`: $\mathrm{acosh}(x)$. E.g. `acosh(1)`.
16. `atanh`: $\mathrm{atanh}(x)$. E.g. `atanh(0.1)`.
17. `floor`: $\lfloor x \rfloor$. E.g. `floor(1)`.
18. `ceil`: $\lceil x \rceil$. E.g. `ceil(1)`.
19. `round`: $\mathrm{round}(x)$. E.g. `round(1.1)`.
20. `signum`: $\mathrm{sgn}(x)$. E.g. `signum(1)`.
21. `atan2`: $\mathrm{atan2}(y, x)$. E.g. `atan2(10,1)`.
22. `min`: $\min(x_1, x_2, \cdots, x_n)$. E.g. `min(1, 10)`.
23. `max`: $\max(x_1, x_2, \cdots, x_n)$. E.g. `max(1, 10)`.
24. `rand`: E.g. `rand(1, 10)`.

An expression may use variables in its syntax. The available variables are the ones defined using variable expressions, and a finite set of hard-coded values, namely:

1. `pi`: $\pi \approx 3.1415926$.
2. `e`: $e \approx 2.7182818$.
3. `FRAME`: The current animation frame index.
4. `TIME`: The elapsed time from the beginning of the animation.
5. `INDEX`: The index of the symbol in the expanded fractal string that is being evaluated.
6. `LENGTH`: The length of the expanded fractal string.
7. `DEPTH`: The current stack depth.

7.1.2 Examples

The following shows an example of the `lsys` extension.

```
axiom F
iter 4
initial_pos 400,820
initial_rot 0
initial_thickness 1
background rgb(156, 198, 236)
initial_color #006400ff
canvas 750,800
seed Windy Algae
inject 5999,! 7500,!

LINE = 10
ANGLE = 0.36
WIND_STRENGTH = 0.005
SPEED = 0.005

F: forward LINE
+: rotate ANGLE + sin(SPEED*TIME*INDEX/LENGTH)*WIND_STRENGTH
-: rotate -ANGLE
[: push
]: pop
!: ignore 1

F -> FF+[+F-F-F]-[-F+F+F]
```

7.2 Meval crate

The meval crate[14] provides a simple math expression parsing and evaluation.
It is able to parse and evaluate math expressions with custom variables and functions.

```rust
use meval::{Expr, Context};

let y = 1.;
let expr: Expr = "phi(-2 * zeta + 1)".parse().unwrap();

// Custom functions and variables
let mut ctx = Context::new();
ctx.func("phi", |x| x + y);
ctx.var("zeta", -1.);

let value = expr.eval_with_context(ctx).unwrap();
```

7.3 Separation of concerns

The application has been separated into four crates and the main application.

The following commutative diagram shows the functional dependencies between the crates. The arrow $A \to B$ means "A depends on B".

$$
\begin{tikzcd}
\text{renderer-cairo} \arrow[r] & \text{renderer} \arrow[r] & \text{engine} \\
 & \text{gui} \arrow[ul] \arrow[u] \arrow[r] & \text{parser} \arrow[ul]
\end{tikzcd}
$$

7.3.1 lindenmayer-engine

The purpose of this crate is to provide a basic L-system interface capable of applying the rules to a string.

This crate only handles the `axiom` and the `rules`.

It exports the following struct

```rust
#[derive(Debug)]
pub struct LSystem {
    pub axiom: String,
    pub rules: HashMap<char, String>,
}

impl LSystem {
    pub fn new(axiom: &str, rules: &[(char, String)]) -> Self;

    // Expand from axiom
    pub fn expand(&self, iter: usize) -> String;

    // Expand any string
    pub fn apply_rules(&self, expression: String) -> String;
}
```

7.3.2 lindenmayer-renderer

This crate is a wrapper around `lindenmayer_engine::LSystem` by handling all the rendering-related variables.
Its purpose is to handle the graphical meaning of each symbol and represent variables and values.

This crate exports a drawing abstraction for a `Canvas`.

```rust
pub trait Canvas {
    fn move_to(&self, x: f64, y: f64);

    fn line_to(&self, x: f64, y: f64);

    fn stroke(&self);

    fn save(&self);

    fn restore(&self);

    fn set_line_width(&self, thickness: f64);

    fn rectangle(&self, x: f64, y: f64, width: f64, height: f64);

    fn set_color(&self, r: f64, g: f64, b: f64, a: f64);

    fn arc(&self, x: f64, y: f64, r: f64);

    fn fill(&self);
}
```

The following shows the wrapper around `lindenmayer_engine::LSystem` along with the needed structures.

```
#[derive(Debug)]
pub enum Operation {
    Forward(Expr),
    Jump(Expr),
    Dot(Expr),
    Rotate(Expr),
    Thickness(Expr),
    Ignore(Expr),
    PushStack,
    PopStack,
    SetColor((f64, f64, f64, f64)),
    SetVar(String, Expr),
}

#[derive(Debug)]
pub struct LSystemRenderer {
    pub lsystem: LSystem,
    pub iter: usize,
    pub initial_pos: (f64, f64),
    pub initial_rot: f64,
    pub initial_thickness: f64,
    pub background_color: (f64, f64, f64, f64),
    pub initial_color: (f64, f64, f64, f64),
    pub canvas: (i32, i32),
    pub seed: String,
    pub injections: Vec<(u32, String)>,
    pub variables: HashMap<String, f64>,
    pub operations: HashMap<char, Operation>,
    pub expression: String,
    pub rng: Rc<RefCell<Pcg64>>,
}

impl Default for LSystemRenderer;

impl LSystemRenderer {
    pub fn new(
        pub lsystem: LSystem,
        pub iter: usize,
        pub initial_pos: (f64, f64),
        pub initial_rot: f64,
        pub initial_thickness: f64,
        pub background_color: (f64, f64, f64, f64),
        pub initial_color: (f64, f64, f64, f64),
        pub canvas: (i32, i32),
        pub seed: String,
        pub injections: Vec<(u32, String)>,
        pub variables: HashMap<String, f64>,
        pub operations: HashMap<char, Operation>,
        pub expression: String,
        pub rng: Rc<RefCell<Pcg64>>,
    ) -> Self;

    pub fn update_expr(&mut self);

    pub fn update_rng(&mut self);
}
```

Any structure implementing the `Canvas` trait has an implementation of the L-system rendering algorithm.

```
impl dyn Canvas {
    pub fn draw_fractal<'a>(
        &self,
        fractal: &LSystemRenderer,
        variables: &'a mut ExprContext,
    ) -> Result<(), meval::Error>;
}
```

This implementation is responsible for the injection of all the CAS functions and the variables `DEPTH`, `INDEX`, `LENGTH`, `pi` and `e`.

7.3.3 lindenmayer-renderer-cairo

This crate is an implementation of the `lindenmayer_renderer::Canvas` trait for the `cairo::Context` type.

7.3.4 lindenmayer-parser

This crate is an import and export layer for `lindenmayer_renderer::LSystemRenderer` using textual configuration.

This crate defines the syntax of the `lsys` file.

7.3.5 lindenmayer-gui

This project is an executable which starts the graphical user interface upon execution.

The GUI adds a layer of abstraction to the renderer, which handles animations.
The hard-coded variables `TIME` and `FRAME` are indeed injected into the grammar by the GUI program.

7.4 Graphical User Interface

The following image shows the application upon start. The editor is empty, thus a white canvas is rendered.

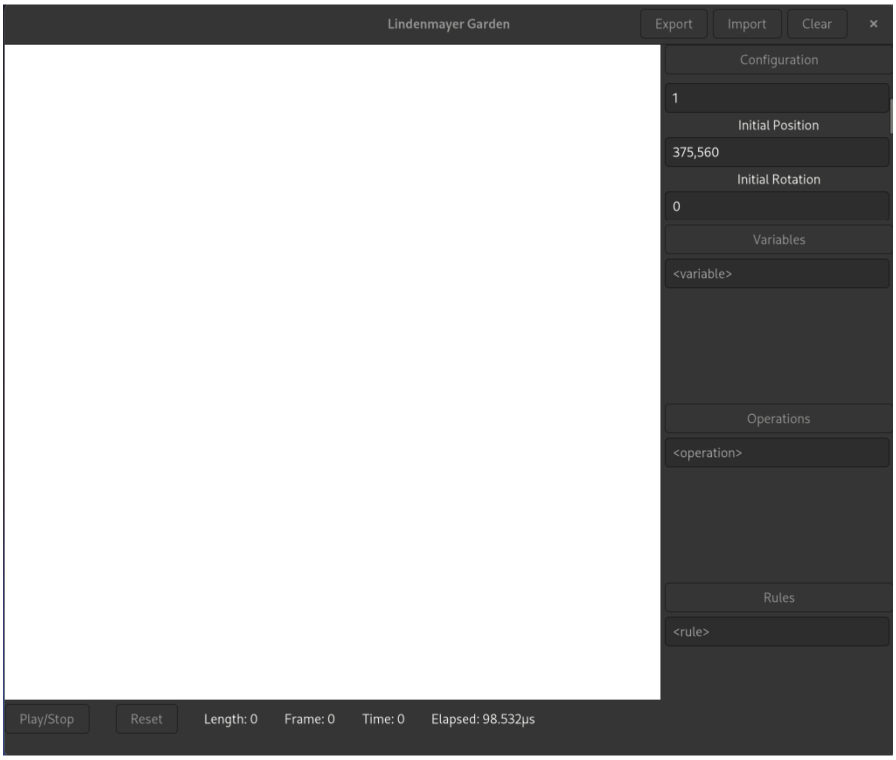


Figure 6: Graphical User Interface - Empty

The graphical user interfaces is mainly composed of 3 sections:

- A canvas where the fractal is rendered.
- A playback section to control the animation and display some properties about it.
- An editor section to edit the fractal in real-time according to the `lsys` format.

The editor section consists of four retractable subsections. The subsections contain the *configurations*, *variables*, *operations* and *rules* values respectively.
The latter 3 subsections are dynamic, meaning that entries are added or removed as needed. The *configurations* subsection is static. Every entry in this subsection is hard-coded and the user is not required to prefix each value with its command.

Here are more examples of the GUI with different configurations

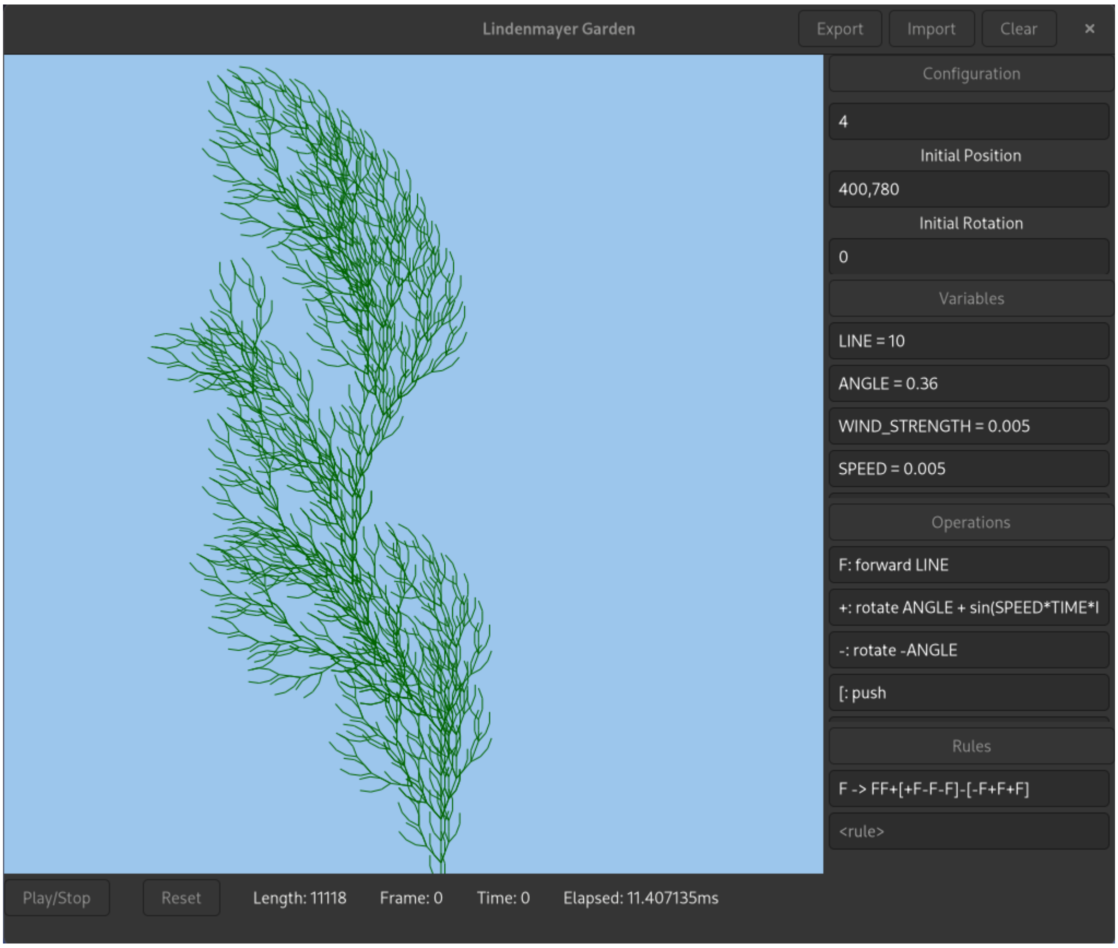


Figure 7: Algae Fractal

```
iter 4
initial_pos 400,820
background rgb(156, 198, 236)
initial_color #006400ff
canvas 750,800
inject 5999,! 7500,!

LINE = 10
ANGLE = 0.36
WIND_STRENGTH = 0.005
SPEED = 0.005

F: forward LINE
+: rotate ANGLE + sin(SPEED*TIME*INDEX/LENGTH)*WIND_STRENGTH
-: rotate -ANGLE
[: push
]: pop
!: ignore 1

F -> FF+[+F-F-F]-[-F+F+F]
```

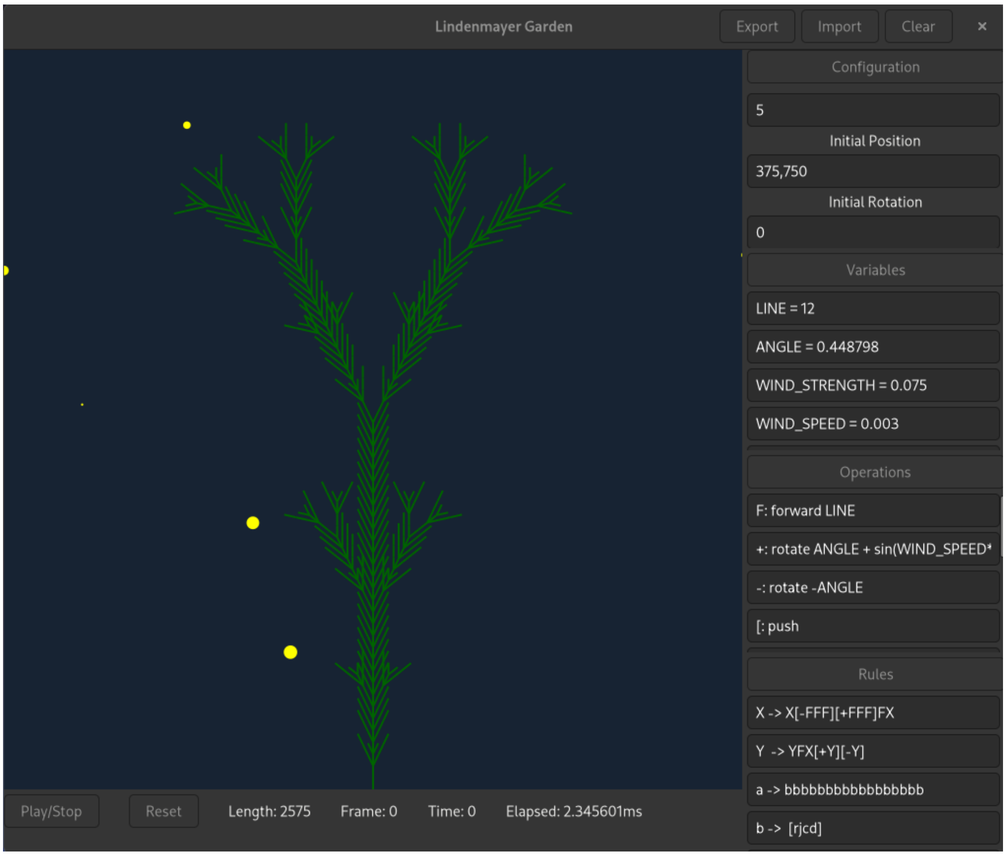


Figure 8: Fireflies Fractal

```
axiom [Y]a
iter 5
initial_pos 375,750
initial_thickness 2
background #172333
initial_color #006400ff
canvas 750,750

LINE = 12
ANGLE = 0.448798
WIND_STRENGTH = 0.075
WIND_SPEED = 0.003

F: forward LINE
+: rotate ANGLE + sin(WIND_SPEED*TIME*INDEX/LENGTH)*WIND_STRENGTH
-: rotate -ANGLE
[: push
]: pop
r: rotate pi*0.2*sin(INDEX) + 0.02*cos(FRAME*INDEX*0.000001)
j: jump 100+300*(1+sin(INDEX*51))+ 10*sin(TIME*INDEX*0.000001)
c: color yellow
d: dot 7*sin((100000 + TIME)*INDEX*0.000001)

X -> X[-FFF][+FFF]FX
Y -> YFX[+Y][-Y]
a -> bbbbbbbbbbbbbbbb
b -> [rjcd]
```

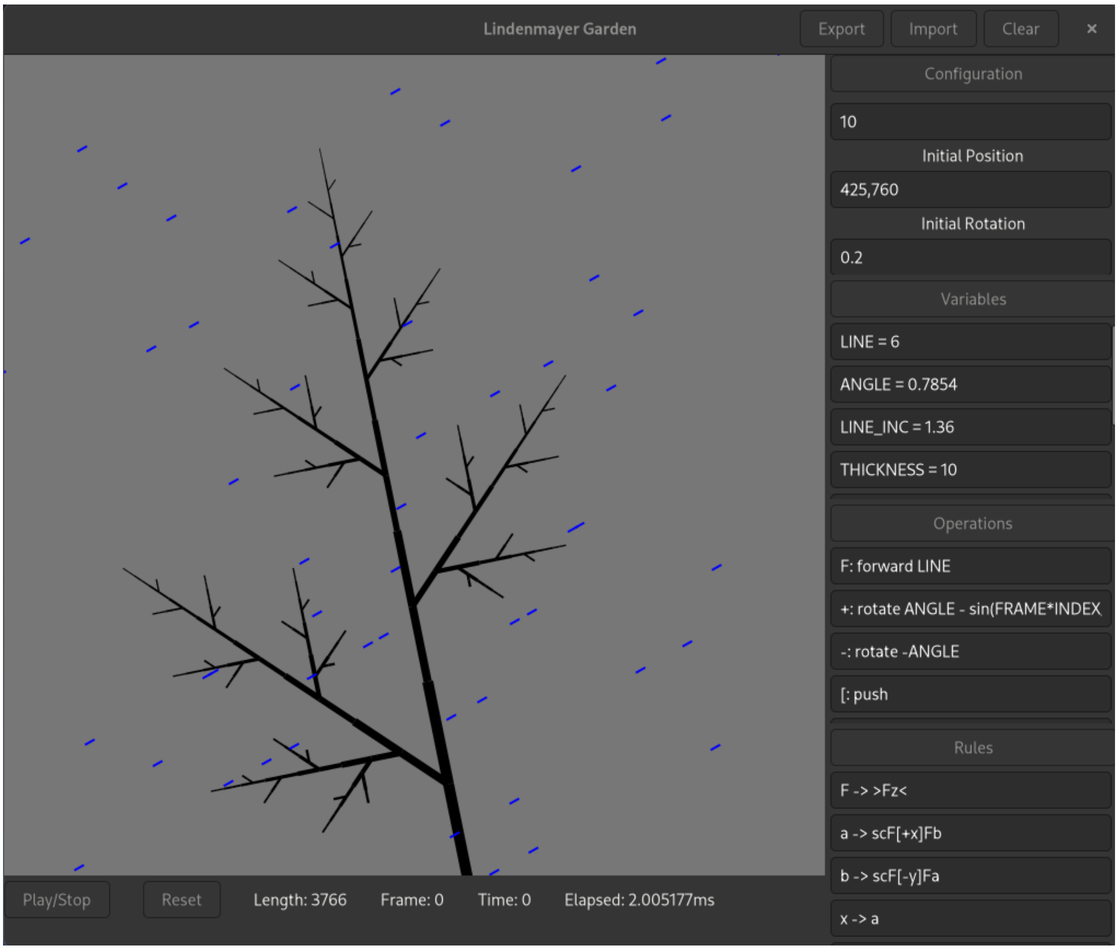


Figure 9: Storm Fractal

```
axiom [a]X
iter 10
initial_pos 425,760
initial_rot 0.2
background #777
initial_color #000000ff
canvas 750,750

LINE = 6
ANGLE = 0.7854
LINE_INC = 1.36
THICKNESS = 10
WIND_STRENGTH = 0.01
TURBULENCE = 0.2

F: forward LINE
+: rotate ANGLE - sin(FRAME*INDEX/LENGTH)/50
-: rotate -ANGLE
[: push
]: pop
>: LINE = LINE * LINE_INC
<: LINE = LINE / LINE_INC
s: thickness THICKNESS
c: THICKNESS = THICKNESS * 0.75
z: rotate abs(sin(TURBULENCE*FRAME*INDEX/LENGTH)*WIND_STRENGTH)
```

```
P: jump -400
V: rotate -pi*0.4
C: color blue
f: jump sin(INDEX)*500
J: jump 50
d: forward max(0, signum(sin((1000000+TIME)*INDEX*INDEX*0.000000002))
   )*10
T: thickness 2

F -> >Fz<
a -> scF[+x]Fb
b -> scF[-y]Fa
x -> a
y -> b
X -> TC[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B]J[B
   ]J[B]
B -> VPJDJDJDJDJDJDJDJDJDJDJDJDJDJDJ
D -> f[d]
```

7.5 Error Handling

Whenever the editor notifies a change in any line, the whole configuration is parsed from scratched. If there aren't any errors, the renderer is updated to render the new fractal.

The graphical user interface program extends the errors given by the parser in order to signal an error if a `lsys` line is written in the wrong section.

The following image shows two correct `lsys` lines in the wrong section.

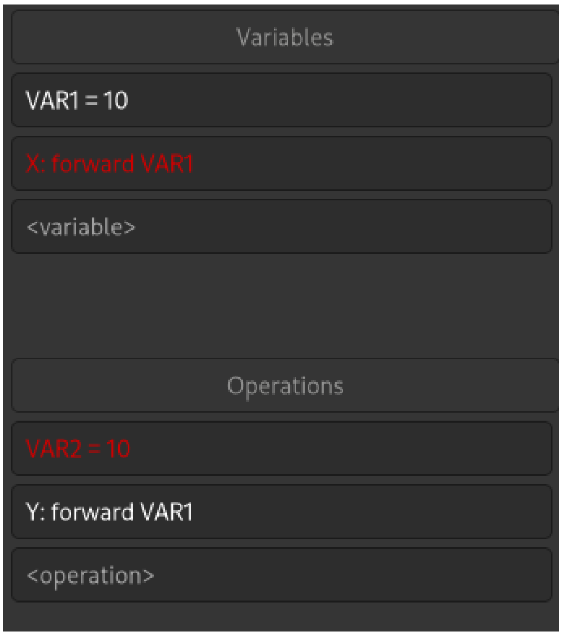


Figure 10: Graphical User Interface - Error

7.6 Project Structure

The following section shows the meaning of each file in the project source code.

lindenmayer-engine

```
src
└── lib.rs | Exports the basic LSystem implementation (string expansion)
```

lindenmayer-renderer

```
src
├── lib.rs | Exports LSystemRenderer structure
├── canvas.rs | Contains the rendering implementation
└── expressions
    └── mod.rs | Contains the rand functino
```

lindenmayer-renderer-cairo

```
src
└── lib.rs | Exports the drawing implementation
```

lindenmayer-parser

```
src
├── import.rs | Contains the import logic
├── export.rs | Contains the export logic
└── lib.rs | Module export
```

lindenmayer-gui

```
src
├── main.rs | Main program entry
├── ui.rs | Contains every element of the UI
├── logic.rs | Contains the main logic of the interactions between the UI and the
│   backend
├── helpers.rs | Generic function helpers
├── config.rs | Utils to handle the configuration of the editor
├── animations.rs | Contains the animation logic layer
└── style
    └── style.css | CSS Styling of the UI
```

7.7 Logging system

The logging level and logging style can be set by exporting the `LSYS_LOG` and `LSYS_LOG_STYLE` environmental variables. The variables represent the log level and the log style respectively.

The `LSYS_LOG` variable may assume the following values

- `error` : Designates very serious errors.
- `warn` : Designates hazardous situations.
- `info` : Designates useful information.
- `debug` : Designates lower priority information.
- `trace` : Designates very lower priority, often extremely verbose, information.

See [5] for more information and more options.

The `LSYS_LOG_STYLE` variable may assume the following values

- `auto` : Will attempt to print style characters if possible.
- `always` : Will always print style characters.
- `never` : Will never print style characters.

See [6] for more information.

If the `LSYS_LOG` environmental variable is not set, the default log value will be `info`.

7.8 Dependencies

The following is a list of all the libraries used within the project crates.

Dependency table (lindenmayer-engine)			
Name	**Description**	**Version**	**Features**

Dependency table (lindenmayer-renderer)			
Name	**Description**	**Version**	**Features**
meval	CAS expressions parsing	0.2.0	
lindenmayer-engine	Basic L-system expansion	(local)	

Dependency table (lindenmayer-renderer-cairo)			
Name	**Description**	**Version**	**Features**
cairo-rs	Cairo bindings	0.17.0	
lindenmayer-renderer	Rendering abstraction	(local)	

Dependency table (lindenmayer-parser)			
Name	**Description**	**Version**	**Features**
csscolorparser	CSS color parsing	0.6.2	
lindenmayer-renderer	Rendering abstraction	(local)	

Dependency table (lindenmayer-gui)			
Name	**Description**	**Version**	**Features**
gtk	GTK4	0.6.0	v4_10
cairo-rs	Cairo bindings	0.17.0	
log	Logging abstraction	0.4.17	
env_logger	Logging implementation	0.10.0	
lindenmayer-renderer	Rendering abstraction	(local)	
lindenmayer-renderer-cairo	Drawing implementation	(local)	
lindenmayer-parsing	`lsys` format parsing	(local)	

8 Tips and tricks

8.1 Random values

The `rand(min, max)` function is pretty useless when designing an animation. Animations cannot generally be smooth when using this function, as whatever value we are randomizing will drastically change between frames, unless the range is very small.
To achieve the same result but keeping the same random values between frames, I found the following method better: setup a variable `SEED = 42` and then use the expression `sin(SEED + INDEX * INDEX)`. This provides a pseudo-random function persistent between frames but different in each point of the fractal. If you want a pseudo-random function persistent in every point of the fractal but different between frames you can use `sin(SEED + FRAME * FRAME)` or `sin(SEED + TIME * TIME)`.

8.2 Natural growth

When designing a natural shaped tree, the symmetry of the fractal can be broken to achieve a more natural growth. This can be achieved by injecting symbols into the fractal string at arbitrary positions. A common choice is the symbol `!: ignore 1`. Alternatively, using the symbol `!: rand(0, 1.005)` in the production rules may also yield good results.

8.3 Stochastic grammar

It is possible to simulate the behavior of a stochastic context-free grammar in a deterministic context-free grammar.
Let's define `X: forward 10` and `Y: dot 10`. We want to define a rule such that the symbol `F` has a certain probability of becoming `X` or `Y`.

Set up the following variables: `PROB = 0.5` (50%) and `MEMORY = 0`.
Redefine `X` as `X: forward 10*MEMORY` and `Y` as `Y: dot 10*(1-MEMORY)`.
Let `M: MEMORY = max(0, signum((rand(0, 1)-PROB)))`.
The production rule is now `F -> MXY`.

The core idea is to always execute both `X` and `Y`, but make their values null in a mutually exclusive way. The expression `max(0, signum((rand(0, 1)-PROB)))` is a distribution of 0 and 1 according to the probability `PROB`.

9 Tests

9.1 Testing protocol

Requirements: The following tests must satisfy the following requirements. The resources can be found in the project source code. Tests must be executed in order.

1. The compiled executable of the `lindenmayer-gui` software must be placed in a folder specified in the PATH environmental variable (e.g. `/usr/local/bin` or `/usr/bin`).
2. The `lindenmayer-gui` file must be an executable. It should be enough to run chmod +x <file>.
3. The user from which the tests will be executed must have permission to execute the program.
4. The project examples must be available in the file system.
5. The user must have the permission to read each resource used and write files to the specified locations.
6. Some compositors do not render the buttons on the header bar of the application whilst in full screen mode. To avoid any problem, do not put the application in full screen mode.
7. A text editor.

Tests:

Test-00 | Application Start	
Reference	Req-04
Steps	1. Start the application by launching the `lindenmayer-gui` command.
Result	A window must appear on the screen with the same look and UI elements as the image (7.4).

Test-01 | Initial error state	
Reference	Req-04, Req-06
Steps	1. Focus on the *Axiom* entry under the *Configuration* section.
Result	The entry must glow red as it is the only error in the default configuration.

Test-02 | Dynamic entries 1	
Reference	
Steps	1. Start the application by launching the `lindenmayer-gui` command. 2. For each section (`Variables`, `Operations` and `Rules`) start typing in the only available entry.
Result	A second entry must appear right beneath the first one.

Test-03 | Dynamic entries 2	
Reference	
Steps	1. Delete the text written in the previous test from the entry of each section.
Result	The second entry must disappear.

Test-04 | Logging 1	
Reference	Req-05
Steps	1. Start the application by launching the `lindenmayer-gui command`.

Result	The output written to the STDOUT must look like this:
	`[<time> INFO lindenmayer_gui] Initializing application`
	`[<time> INFO lindenmayer_gui] Loading CSS stylesheet`
	`[<time> INFO lindenmayer_gui::logic] Initializing UI content`
	`[<time> INFO lindenmayer_gui::ui] Building UI`
	`[<time> INFO lindenmayer_gui::logic] Initializing configuration`
	`[<time> WARN lindenmayer_gui::helpers] Input is incorrect`

Test-05 | Logging 2	
Reference	Req-05
Steps	1. Start the application by launching the `LSYS_LOG=debug lindenmayer-gui` command.

Result	The output written to the STDOUT must look like this:
	`[2023-05-24T07:50:18Z INFO lindenmayer_gui] Initializing application`
	`[2023-05-24T07:50:18Z INFO lindenmayer_gui] Loading CSS stylesheet`
	`[2023-05-24T07:50:18Z INFO lindenmayer_gui::logic] Initializing UI content`
	`[2023-05-24T07:50:18Z INFO lindenmayer_gui::ui] Building UI`
	`[2023-05-24T07:50:18Z INFO lindenmayer_gui::logic] Initializing configuration`
	`[2023-05-24T07:50:18Z WARN lindenmayer_gui::helpers] Input is incorrect`
	`[2023-05-24T07:50:18Z DEBUG lindenmayer_gui::animations] Updating renderer. Generating fractal`
	`[2023-05-24T07:50:18Z DEBUG lindenmayer_gui::animations] String expansion took: 803ns`

Test-06 | Import feature	
Reference	Req-00, Req-01, Req-04
Steps	1. Press the `Import` button at the top of the GUI. 2. Navigate to the examples folder and select the `fireflies.lsys` file. 3. Press `Open`.
Result	A fractal must be rendered onto the canvas as in image (7.4).

Test-07 | Real time editor	
Reference	Req-04, Req-00
Steps	1. Under the *Variables* section, focus on the `LINE` variable entry. 2. Modify the text from `LINE = 12` to `LINE = 6`.
Result	The fractal must shrink to half its original size as soon as the text has been modified.

Test-08 | Export feature	
Reference	Req-04
Steps	1. Press the `Export` button at the top of the GUI. 2. Save the file to the examples location with the name `fireflies2.lsys`. 3. Press `Save`.
Result	Open the saved file with a text editor. The file must contain some data and the correct `LINE = 6` line.

Test-09 | Consistency	
Reference	Req-00, Req-01, Req-04
Steps	1. Press the `Import` button at the top of the GUI. 2. Navigate to the examples folder and select the `fireflies2.lsys` file. 3. Press `Open`.
Result	The same smaller fractal must be rendered onto the canvas.

Test-10 | Animation start	
Reference	Req-01, Req-03, Req-04
Steps	1. Press the `Import` button at the top of the GUI. 2. Navigate to the examples folder and select the `algae.lsys` file. 3. Press `Open`. 4. Press the `Play/Stop` button under the canvas.
Result	An animation must begin where the algae softly sways around.

Test-11 | Reset feature	
Reference	Req-04
Steps	1. Press the `Reset` button under the canvas.
Result	The animation must restart from the beginning. The `Frame` and `Time` label under the canvas must restart from a value of 0.

Test-12 | Animation stop	
Reference	Req-03
Steps	1. Press the `Play/Stop` button under the canvas multiple times.
Result	The animation must stop and resume at every other press of the button.

Test-13 | Clear feature	
Reference	Req-03, Req-04
Steps	1. Press the `Clear` button at the top of the GUI.
Result	The entire GUI must be reset to its initial state of a white canvas and empty editor.

Test-14 | Retractable section	
Reference	Req-04
Steps	1. Click on the titles of each of the four section.
Result	All the sections must hide their contents.

Test-15 | Syntax errors 1	
Reference	Req-01, Req-06
Steps	1. Start the application by launching the `lindenmayer-gui` command. 2. Press the `Import` button at the top of the GUI. 3. Navigate to the examples folder and select the `storm.lsys` file. 4. Press `Open`. 5. Under the *Operations* section, focus on the `F: forward LINE` entry. 6. Modify the text from `F: forward LINE` to `F: hello LINE`.
Result	The entry must glow red as the operation is invalid.

Test-16 | Syntax errors 2	
Reference	Req-06
Steps	1. Modify the text from `F: hello LINE` to `F: forward LINE`.
Result	The entry must stop glowing red as the operation is now valid again.

Test-17 | State errors	
Reference	Req-06
Steps	1. Modify the text from `F: forward LINE` to `F: forward line`.
Result	A red label at the top of the editor must appear with the text: Invalid var: line.

Test-18 | Stochastic behavior	
Reference	Req-02
Steps	1. Modify the text from `F: forward line` to `F: forward LINE`. 2. Under the *Operations* section, focus on the `-: rotate -ANGLE` entry. 3. Modify the text from `-: rotate -ANGLE` to `-: rotate rand(0,1)`.
Result	All the branches of the tree that were on the right side must now be on the left side at random angles

Test-19 | Random seed	
Reference	Req-02
Steps	1. Under the *Configuration* section, focus on the `seed` entry. 2. Continuously add letters to the seed.
Result	As the seed changes, the branches of the tree at random angles must change their angle.

9.2 Test results

Every test has successfully been passed. The application still has some minor bugs, but it is stable.

ID	Result	Note
Test-00	Passed	The application starts correctly.
Test-01	Passed	The entry becomes red.
Test-02	Passed	New entries are generated.
Test-03	Passed	Entries are removed.
Test-04	Passed	The output looks like the given one.
Test-05	Passed	The output looks like the given one.
Test-06	Passed	The fractal is rendered correctly.
Test-07	Passed	The fractal shrinks its size.
Test-08	Passed	The file contains the correct information.
Test-09	Passed	The fractal is rendered correctly.
Test-10	Passed	The animation plays correctly.
Test-11	Passed	The animation restarts.
Test-12	Passed	The animation stops.
Test-13	Passed	The editor and canvas are cleared.
Test-14	Passed	The sections retract correctly.
Test-15	Passed	The entry glows red.
Test-16	Passed	The entry stops glowing red.
Test-17	Passed	The correct label appears.
Test-18	Passed	The fractal changes accordingly.
Test-19	Passed	The branches angles change randomly.

10 Conclusions

10.1 Future development

There are lots of features that could be added to the application and improvements that could be made. There are also some known bugs.

1. **More features**: Additional drawing features could be added to the grammar of the system, such as polygons, line types, curves, functions and so on.
2. **Parallel rendering**: The drawings are rendered by the main graphical thread. This results in the UI being slow if the rendering of the animation is particularly hard.
3. **Resources exhaustion**: The string expansion of the fractal is done by the main graphical thread. If the user, for instance, accidentally adds a digit to the number of iterations, the program will start expanding the fractal string even if there is not enough RAM in the entire world that could contain it, resulting in the hard crash of the program. It would be useful to add a threshold within which the program will stop computing the fractal string, or a "safety mode" that can be activated/deactivated.
4. **Refactor**: The overall structure of the code is, in my opinion, pretty good and flexible. However, if I could go back, I would definitly make it even better. The program relies too much on the "text format parsing". Every time a value is changed in the editor, everything is parsed again from scratch.
5. **Import/Export crash**: Sometimes the import and export features will crash whilst opening a folder in the file dialog. The error is caused by a `free(*ptr)` operation in the underlying libraries, but I have no idea how to fix it or if I even did something wrong, it may just be a library bug.
6. **Stop rendering if error**: If the configuration is correct, but an expression uses an invalid variable or function, the render loop will keep going even thought nothing will be rendered.
7. **Export**: An export feature of the render could be added. The program could export single frames as images, or record a part of an animation and export it as a video.
8. **Grammar types**: It would be useful to support other grammar types, such as context-sensitive grammars or stochastic grammars.
9. **Multiple operations**: Being able to associate multiple operations to a symbol (E.g. `F: ; rotate 0.1`) would drastically reduce the complexity of some systems.
10. **Multiple renders bug**: If the canvas is not wide enough, such that the labels beneath it are larger than the canvas, the rendering of the frame will change the value in the `Elapsed` label. Since the labels occupy more space than the canvas, and the size of the `Elapsed` label changes, the left side of the UI changes its width, triggering another render operation to be called (and so on).

10.2 Personal conclusions

The topic of the project was one of the best ones I had ever worked on. I really enjoyed implementing new features, and I spent hours playing with my own product. I am very glad to have been able to shape the workings of the L-systems with my ideas, which have extended the product beyond the requirements of the project. I had many problems handling the memory within the GTK application, but in the end it all worked out, and I learned much about memory management in Rust and smart pointers.

A picture of the Mandelbrot Set[13] drawn using an L-system.

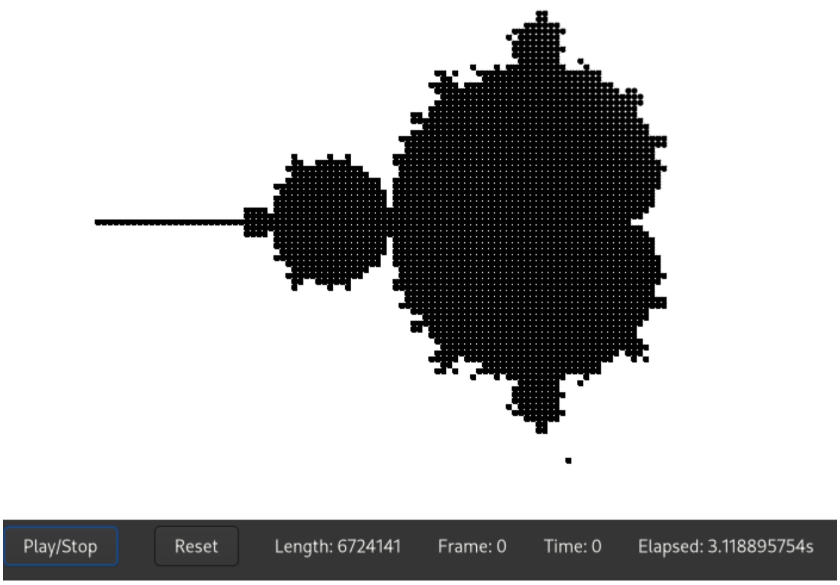


Figure 11: The Mandelbrot Set

List of Figures

1 Koch curve for $n = 4, 3, 2$. 8
2 Plant for $n = 4$. 8
3 GUI Design sketch 12
4 Initial Gantt Chart 14
5 Final Gantt Chart 15
6 Graphical User Interface - Empty 24
7 Algae Fractal 25
8 Fireflies Fractal 26
9 Storm Fractal 27
10 Graphical User Interface - Error 29
11 The Mandelbrot Set 41

References

[1] Wikipedia contributors. *Algebraic data type — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Algebraic_data_type&oldid=1155574153. [Online; accessed 24-May-2023]. 2023.

[2] Behdad Esfahbod Carl Worth. *Cairo*. 2023. URL: https://cairographics.org/.

[3] Mozilla. *<color>*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/color_value#format_syntax.

[4] Wikipedia contributors. *Context-free grammar — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Context-free_grammar&oldid=1142429919. [Online; accessed 25-May-2023]. 2023.

[5] Docs.rs. *Crate env_logger*. URL: https://docs.rs/env_logger/latest/env_logger/#enabling-logging.

[6] Docs.rs. *Crate env_logger*. URL: https://docs.rs/env_logger/latest/env_logger/#disabling-colors.

[7] Wikipedia contributors. *Formal grammar — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Formal_grammar&oldid=1146063605. [Online; accessed 25-May-2023]. 2023.

[8] Wikipedia contributors. *Fractal — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Fractal&oldid=1156384037. [Online; accessed 24-May-2023]. 2023.

[9] Paolo Bettelini. *giardino-lindenmayer*. 2023. URL: http://gitsam.cpt.local/lavoro_finale_lpi_2023/giardino-lindenmayer.

[10] gtk rs. *GUI development with Rust and GTK 4*. 2023. URL: https://gtk-rs.org/gtk4-rs/git/book/installation.html.

[11] Mozilla Research. *Install Rust*. 2023. URL: https://www.rust-lang.org/tools/install.

[12] Wikipedia contributors. *L-system — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=L-system&oldid=1142810895. [Online; accessed 24-May-2023]. 2023.

[13] Wikipedia contributors. *Mandelbrot set — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Mandelbrot_set&oldid=1156064172. [Online; accessed 26-May-2023]. 2023.

[14] rekka. *meval-rs*. 2018. URL: https://github.com/rekka/meval-rs.

[15] Mozilla Research. *Rust*. 2010. URL: https://www.rust-lang.org/.

[16] Wikipedia contributors. *Turtle graphics — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Turtle_graphics&oldid=1098183733. [Online; accessed 24-May-2023]. 2022.

Glossary

C A general-purpose computer programming language. 13

C++ An object-oriented programming language derived from C. 13

cargo Package manager for the Rust programming language. 16

CAS Computer Algebra System. 18, 23, 32

CSS Cascading Style Sheets. a stylesheet language used to describe the look and formatting of HTML documents. 18, 30, 32

Gantt A bar chart that illustrates a project schedule. 14

GTK GIMP Toolkit. 13, 40

GUI Graphical User Interface. A type of user interface that allows users to interact with a computer program using graphical elements. 9, 10, 13, 23, 25, 35, 36, 37

Rust A systems programming language designed for speed, safety, and concurrency, which is known for its memory safety and thread safety features. 13, 40

trait A trait in Rust defines shared behavior among structures. 23

waterfall classical model used in system development life cycle to create a system with a linear and sequential approach. 14