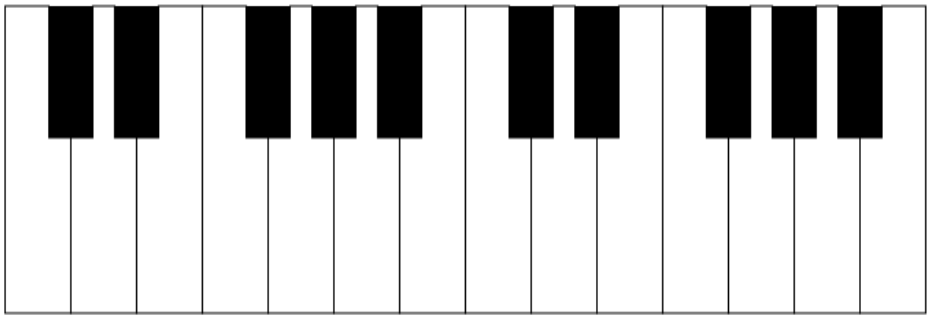
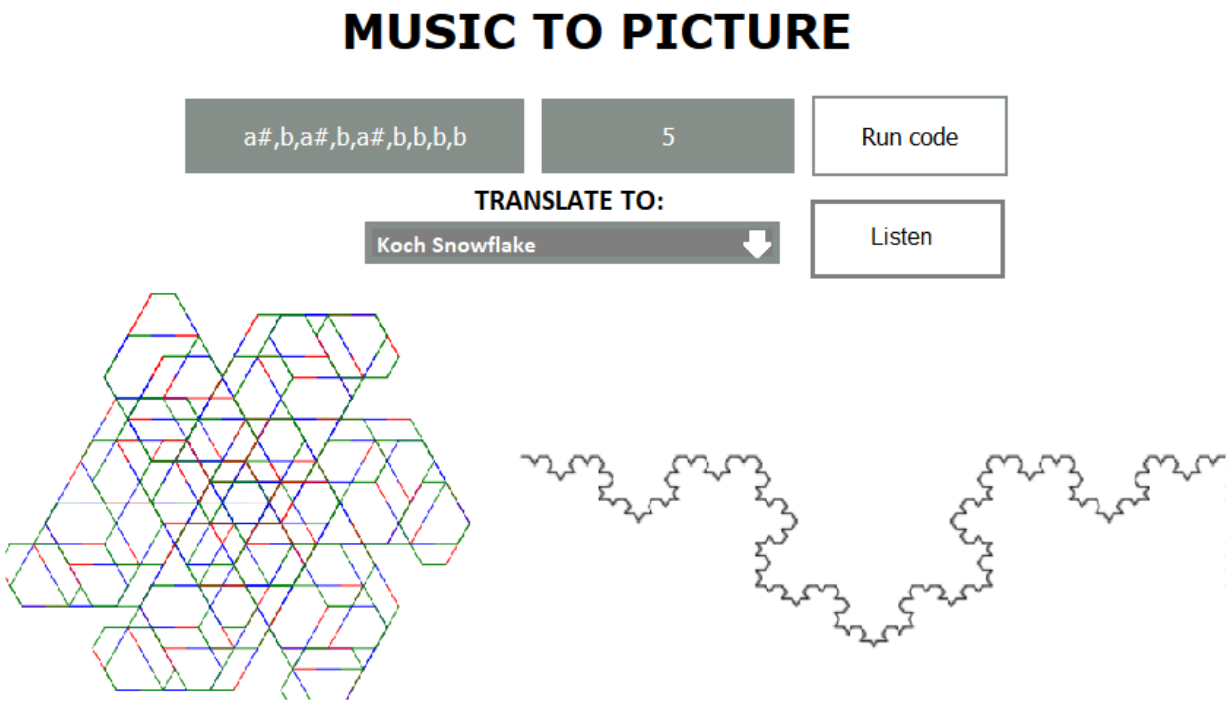
CHALMERS
UNIVERSITY OF TECHNOLOGY





Translation between fractal images and music using Grammatical Framework

Bachelor's thesis: Planning report

DATX02-19-26, Supervisor: Krasimir Angelov

H. ANDERSSON, A. BERGSTEN, B. BRANDSTRÖM, G. ENGSMYRE, E. KNOPH, E. MEIJER

ABSTRACT

An Informative Headline describing the Content of the Report
A Subtitle that can be Very Much Longer if Necessary
NAME FAMILYNAME
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Keywords: lorem, ipsum, dolor, sit, amet, consectetur, adipisicing, elit, sed, do.

ACKNOWLEDGEMENTS

Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Name Familyname, Gothenburg, Month Year

Contents

List of Figures

List of Tables

1
Introduction

Fractals have always been fascinating to mathematicians, programmers and people in general. There are multiple ways of generating fractals, and these ways vary among different fractal types, but self-similar fractals are possible to define using a *grammar* (see section 2.2), similar to how a grammar can define a natural language. Grammatical Framework (see section 2.3) is a programming language mainly used for translating between natural languages by defining a thorough grammar for each of the languages one wishes to translate between. Since fractals also can be described by a grammar, if these grammars were defined in Grammatical Framework it should be possible to translate between fractals. For music that is self-similar it should also be possible to define a grammar for a musical fractal, making it possible to translate between fractal images and music.

To be able to translate between fractal images and/or music is mainly of academic interest. It is a way to possibly extend the use of Grammatical Framework and examine fractals from a different point of view. It could also be used to help people further understand fractals by comparing them with each other, study their grammars and visualise their construction. Depending on the user interface, the musical part could also be used to help composers by letting them be inspired by the fractal nature of music. Therefore, three main target groups have been identified: people interested in different applications of Grammatical Framework, students trying to understand fractals and people composing music. The purpose of this report is to give a brief introduction to – as well as providing a time plan for – a project for developing a web application that will take a Grammatical Framework generated string and interpret it as fractal images or music. The project also aims to enable translation between fractal images and/or music [1].

2
Background

2.1 Fractals

Benoit. B. Mandelbrot gives the following definition of a fractal (not needed to be understood for this text):

"A *fractal* is by definition a set for which the Hausdorff Besicovitch dimension strictly exceeds the topological dimension." [2]

However, as Kenneth Falconer states in his book on the topic, this definition and many others prove unsatisfactory as they do not apply for certain sets that should be considered to be fractals. Instead, Falconer argues, a set F can be thought of as a fractal if:

- "F has a fine structure, that is, detail on arbitrarily small scales."
- "F is too irregular to be described in traditional geometrical language, both locally and globally."
- "Often F has some form of self-similarity, perhaps approximate or statistical."
- "Usually, the 'fractal dimension' of F (defined in some way) is greater than its topological dimension."
- "In most cases of interest, F is defined in a very simple way, perhaps recursively." [3]

In this text, the exact definition is not very relevant. All examples of fractals that will be considered during the project will be self-similar (exactly or statistically) and defined recursively.
A set F is called *self-similar* if it is made up of disjoint subsets that are all similar to F.
A set F is called *statistically self-similar* if it is made up of disjoint subsets with F and the subsets all being possible outcomes to some random distribution. [2]

2.2 Grammar

A grammar is according to the book *Grammatical Framework – Programming with Multilingual Grammars* by Aarne Ranta defined as: "a set of rules for analysing and producing text and speech" [4]. For natural languages (i.e. Swedish), grammars are a consequence of the language they describe. For formal languages, the language is

defined by its grammar and has thus been designed with a specific purpose. [4]

The type of grammar that will be considered for this project is formal grammar. A formal grammar consists of three components: an *alphabet*, an *axiom* and a set of *rules* that should be applied to the axiom and the following iterations of it. The alphabet is the set of characters that can be used in the particular grammar. Among many things it can be letters, descriptions of drawing operations or musical notes, depending of what type of grammar it is. The axiom, as stated above, is the initial "sentence" consisting of characters from the alphabet and it will be changed with respect to the rules stating which changes should be made to which part of the string. [5]

It is possible to define *stochastic rules* X, meaning rules that have $k \in \mathbb{N}$ different possible outcomes x_i $(i = 1, ..., k)$ with each outcome having a probability $P(X = x_i) = p_i$, satisfying $0 \leq p_i \leq 1$ and $P(X = \bigcup_{i=1}^{k} x_i) = \sum_{i=1}^{k} p_i = 1$. [6]

2.2.1 Lindenmayer Systems

A Lindenmayer system, or *L-system*, is a system that uses *rewriting* e.g. for generating fractals. Rewriting is a process where the axiom is changed to a more complex one by iteratively replacing parts of it. To determine what parts of the axiom should be changed and in what way, one uses a grammar. [7] In an L-system, all possible rules are applied at every iteration [8]. An example of a simple L-system is:

$$\begin{aligned} \text{Alphabet} &: AB \\ \text{Axiom} &: A \\ \text{Rules} &: A \rightarrow AB, B \rightarrow A \end{aligned}$$

This system takes the initial string "A" and replaces the A with AB, generating the new string "AB". In this string the A is replaced with AB and the B is replaced with A, generating the string "ABA". The iterations continue as follows:

$$\begin{aligned} &0 : A \\ &1 : AB \\ &2 : ABA \\ &3 : ABAAB \\ &\vdots \\ &7 : ABAABABAABAABABAABABAABAABABAABAAB \end{aligned}$$

This string can then be interpreted differently depending on what operations A and B represent. To know how to translate a string from an L-system, the following example with another alphabet can be used. This alphabet, and its interpretations, is commonly used for L-systems.

F : draw a line and move forward
G : move forward without drawing a line
$+$: turn right
$-$: turn left
$[$: save current location
$]$: restore previous location

The alphabet can be translated into code by using the following commands:

F : line(0,0,0,len); translate(0,len);
G : translate(0,len);
$+$: rotate(angle);
$-$: rotate(-angle);
$[$: pushMatrix();
$]$: popMatrix();

A string generated from a grammar in GF could thus be interpreted in another programming language, e.g. Python, to draw a fractal or generate musical notes. [9] A possible way to interpret the alphabet as music could be as follows:

F : play the current tone
G : rest
$+$: move up one tone in the scale
$-$: move down one tone in the scale
$[$: save current tone
$]$: restore previous tone

2.3 Grammatical Framework

Grammatical Framework (GF) is a programming language developed in 1999 which is used for defining different types of grammars in a programming context. It is mainly used for natural languages and serves as a tool for translating between them by defining complex grammatical structures for each language. The design of GF is based on typed functional programming languages such as Haskell, but since the users of GF vary from programmers to linguists it needs to be accessible to people of various backgrounds. [4] Though the main application of GF lies within natural

languages, it can also be used for other purposes where a grammar is a core component. One example is fractals based on Lindenmayer systems (see section 2.2.1).

When translating between two languages/grammars in GF one defines an *abstract syntax* where the axiom and the rules for the grammar is defined. This is inherited by one or many *concrete syntaxes*, where the different alphabets are defined. A system with one abstract syntax and several concrete syntaxes is called a *multilingual grammar*. [4]

2.3.1 Simulated recursion in GF

To achieve a fractal by GF generation the defined grammar needs to be called in a recursive manner. However, recursion is not supported by any type of grammar and will therefore be simulated by repeatedly calling a function on itself, any number of times.
An example of recursion in GF reads as follows:

$$S(S(S(S(z(\text{axiom})))))$$

This call would have a recursive depth of 4, where the function for simulating the recursion is called S, the function for describing the fractal rules is called z, and the whole expression generates a full fractal.

2.4 MIDI files

MIDI (Musical Instrument Digital Interface) is a communication protocol invented in 1983 and is used as a tool for communication between digital musical instruments. There are three major parts of the MIDI system: a message format, a storage format and a physical connector. The message format is used for communication between musical instruments and computers by translating the music into programming language, making it possible to e.g. change recorded music using a computer. The file extension *.mid* is an industry standard that can be used by many different music programs. To connect the musical instruments to each other, or to a computer, physical connectors such as MIDI cables are used. MIDI messages are received through the in-port and MIDI data is transmitted through the out-port and can thereafter be worked on. [10]

2.5 Turtle graphics for Python

Turtle is a graphical tool first developed in 1966 for introducing programming to children in the LOGO programming language. It is based on that the user gives commands as: ”turtle.left(45)”, and ”turtle.forward(25)” - in Python. The Turtle library in Python is mainly used for primitive graphical representations, as lines and simple geometrical shapes. Since many common fractals are created using straight lines that differ in angle and length, Turtle is sufficient for visualising them. [11]

3

Purpose

The main purpose of this project is to examine how GF can be used as a tool for translating between different fractals, as well as between fractals and music, much like how translation between natural languages work. Furthermore, a secondary purpose of the project is to examine how to make the final product interactive and interesting for the target groups mentioned in section **??**.

4

Task

When interpreting the project description – with input from the supervision meetings – some initial problems were identified:

- How can fractals be generated using GF and L-systems?
  - How can fractals with different grammars be described in GF to make them translateable?
  - What should the output from GF look like, how will the grammar alphabet be defined?
- How should the GF output string be parsed for adaptation to visualisation/-generating music?
  - How will the parser get and read the output from GF?
  - How will the parser adapt the format from the GF string to the visualisation/music generation?
- How can music and/or fractal visualisation be generated?
  - To make the product more interesting, how can an interactive Graphical User Interface (GUI) be achieved?
- How will the product be adapted so that it can run on multiple operating systems?

During internal group discussions – whilst trying to refine the scope of the product – new ideas surfaced, mainly focused on the interaction with the product, which led to further problems:

- How can more interactive elements be added, that aim at the products musical aspect?
  - Can a piano interface be added to the UI, for playing melodies to make it more interactive and fun to use?
  - How will the fractal generation/visualisation be affected by the played melodies?
  - Will it be possible for the users to create their own grammars from the GUI that will affect the fractal(s)?
- If fractals are generated through stochastically chosen rules, will it feel like they have a larger variation, and will the product entertain for a longer period of time?

These later problems will only be studied if the first set of problems have been solved.

5
Scope

5.1 Fractals

The fractals that will be generated in this project will be limited to those that can be represented as an L-system (see section 2.2.1), since the main purpose is to translate between different fractals and not to generate very advanced ones. The focus will therefore be on fractals like the *Sierpinski triangle* (Figure 5.1) or the *Koch snowflake* (Figure 5.2) and not for example the *Mandelbrot set* (Figure 5.3).

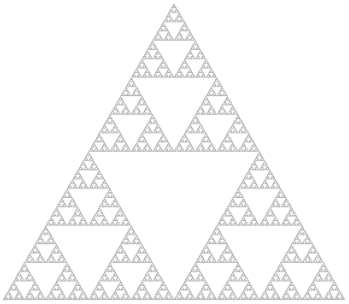


Figure 5.1: An example of the Sierpinski triangle. From [12]. CC-BY-SA.

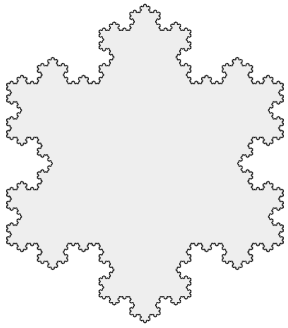


Figure 5.2: An example of a Koch snowflake after seven iterations. From [13]. Reproduced with permission.

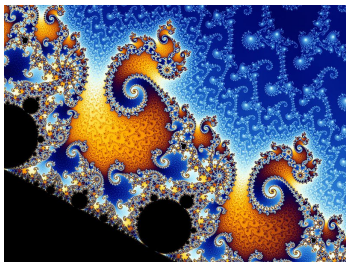


Figure 5.3: An example of the Mandelbrot set. From [14]. CC-BY-SA.

5.2 Graphical representation

In the beginning of the project, the graphical representation of the fractals will be simple and only in one colour. This is because it seems more important to get all the parts of the project working before one part is further developed. If the rest of the project is working fine, the graphical representation might be developed further with different colours etc.

5.3 Musical input

As stated in chapter **??**, it is important not to use up too much research funds that could have better use elsewhere. Therefore, this project will initially not be using

any components that need to be bought for this project only. If however the result is satisfying, a small piano keyboard with a MIDI connection will be purchased and used as part of the user interface. Still, no other musical instruments – or singing – will be taken as input since that would have impact on both the expenses and the complexity of the project.

5.4 Stochastic rules

In the beginning of the project, stochastic rules will not be considered but if it would turn out that there is time, stochastic rules for the generation of fractals can be introduced. This will allow for more varied pictures that still show (statistical) self-similarity.

5.5 Grammar input

Initially the grammars will not be taken as input from the user; conversely, they will be pre-defined in GF. This is to ensure that the communication between GF and the user interface is working properly before complexifying the GF structure. If or when this is working properly, it would be interesting to explore the possibilities of letting the user define grammars. That way the product could become more interesting to the users as it will be possible to generate a lot of different fractals, both images and music.

6
Method

The fractals in this project will be represented by strings generated from a grammar in Grammatical Framework. The strings will be the blueprints for drawing the fractals – or playing the music – generated from the grammar. The project has been divided into three different main parts. The first part is generating the strings and translating from one fractal image to another, and from image to music. The second part concerns parsing the strings, and the third part concerns drawing the fractal images and playing the music as well as the user interface (a web application). If these three tasks are successfully completed, how to further develop user interactivity will be examined.

6.1 Generating the strings and translating between different fractals and music

To generate the strings, which will later be graphically represented with Python and the Turtle package (section 2.5), a grammar for fractals in the form of an L-system (section 2.2.1), will be used. This grammar will be implemented with GF (section 2.3). Provided an axiom, the rules of the fractal L-system will be applied iteratively in GF, thus generating strings that represent the two-dimensional growth of the fractals [8].
As music can also be viewed as being two-dimensional, with notes having both the property of "height" (pitch) and "length" (duration), an interpretation of the strings as music should be possible. As long as the grammars for the different fractal images and music implement the same abstract grammar in GF, translation between different fractal images and from fractal image to music should be possible.
Already existing grammars for fractals [15] will be the starting point for this project. These grammars will be modified and then implemented in GF to create the strings representing the fractals. A grammar for music will also be created, either by modifying an already existing system, or by creating a new one.

Another option for generating the strings would be using a grammar with stochastic functions in GF. This would enable rendering more random fractals; the same L-system could render different fractals or notes, as opposed to a non-stochastic L-system, which would result in the same fractal every time [8]. If the project progresses well, this aspect will be incorporated.

6.2 Parsing the strings

In the product, the *parser component* is responsible for reading the string generated by GF grammar and translating/interpreting it to commands that Turtle or a music generation component can understand. The parser thus has the ability to interpret a set of symbols in the string, and then map these to a known command for generating music and/or visualising the fractals. For generating music, this could mean that a set of symbols in the fractal string would correspond to "*play the tone A as a half note and then go to the next note in the A-major scale*", and for visualisation it could correspond to "*draw a line for 2 cm then rotate 45° and draw a line for 1 cm*". Though a parser in theory is a simple construction, it requires a clear understanding of both the string, its format, and how the generation of music and/or visualisation of fractals works. This leads to one of the major issues with constructing the parser component: creating a grammar alphabet which generates strings that seem reasonable to parse, in comparison to the commands that should be given for generating music and/or visualising the fractals. Creating the alphabet is probably a task that will reoccur during the project every time new insights are given, and the alphabet will have to be refined.

6.3 Drawing the fractals and the user interface

The user interface will, at the start, consist of a web application, where the user will be able to choose from a list of different fractal types which are to be drawn on the screen. There will also be an option to translate the first fractal image into a different kind of fractal (e.g. from a Sierpinski triangle to a Koch snowflake). The user will also have the option to hear the fractals interpreted as music.

6.4 Taking the project further

In the initial stage of the project, focus will be on the creation and interpretation of strings as different fractals, as well as creating a functioning grammar for music. Parsing the strings and the visual representation, will also be one of the main focus areas. After this has been accomplished in a satisfactory way, the goal is to develop the user interface so that it will provide the option to take user input via a piano keyboard (either a virtual or a real one). The user will be able to play a few notes on the keyboard, and this input would then impact the creation of the fractal images and/or music in someway. It could for instance impact the graphical representation of the fractals or change the probability of the stochastic functions in the grammars, if such functions have been implemented.

There are a multitude of other options for developing the project further. The graphical representation could be evolved, for instance adding colours, different levels of user interactivity, etc. The user being able to select smaller fractions of one of the two generated fractals (corresponding to certain iterations of the grammar in GF), and then seeing the corresponding part in the second fractal highlighted,

choosing how many iterations will be used when creating the fractals and so on. Another option could be letting the user provide their own L-system for generating fractals. To be able to understand the finished project and the process to get there a mock-up and a prototype was made.

6.4.1 Mock-up

The mock-up includes all different aspects of our planned project. The piano, translation between fractals, possibility to choose number of iterations and to listen to the fractals. Figure 6.1 shows that the piano notes and the number of iterations have influenced the visualisation of the left fractal. The user has then chosen to translate the right fractal to a version of a Koch Snowflake. There is also a possibility to listen to the fractal(s) as music.

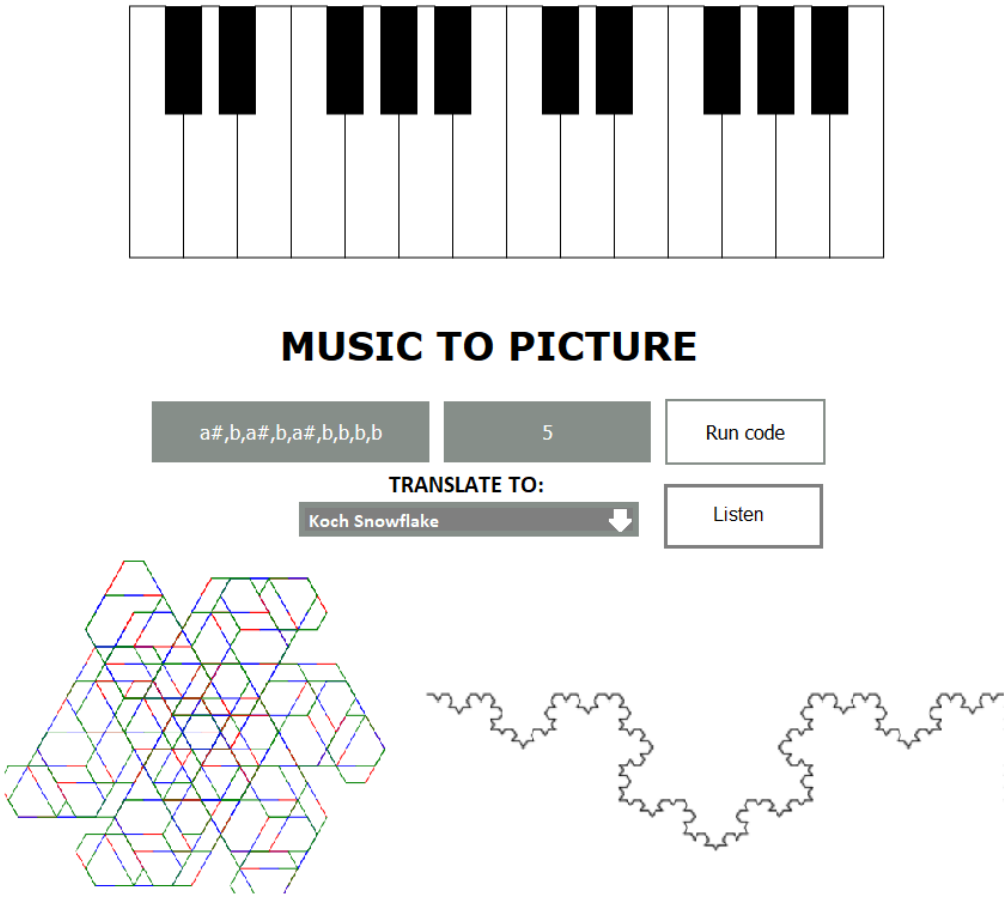


Figure 6.1: A mock-up of all different parts of the final product. When the keys are pressed, the respective tones will appear in the grey box to the left. The number of iterations should be written in the second grey box. When the user is finished filling in these boxes, he/she presses the ”Run code” button. When the first fractal has been drawn, the user can choose to translate it to another fractal by the drop down menu and pressing "Run code" again. If the user want to listen to the fractal(s) he/she can press the ”Listen” button.

Because uncertainty of how this product will turn out, this mock-up is a way to understand how all these components may interact with each other.

6.4.2 Prototype

As the final product is supposed to be a web application to avoid troubles with different operating systems, there needs to be a way to get Python to work in the web browser. The main purpose of this prototype was to understand of how the implementation with Turtle would work in a web application, as Python normally isn't used for web programming. This prototype is programmed with Python and Brython [16]. The user simply plays the axiom of notes, enters how many iterations they want to do and click the "Run code" button, (see Figure 6.2). The program uses these parameters to draw an image; the rendering of the image depending on the L-system.

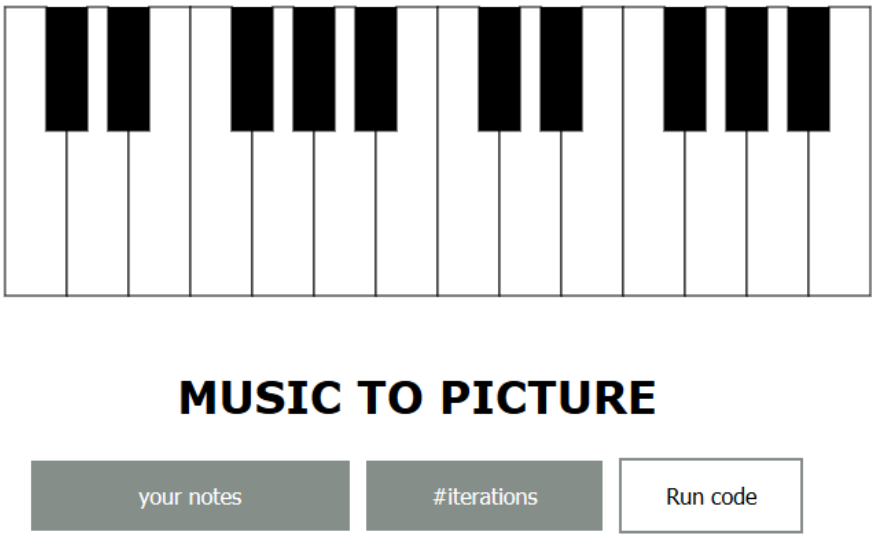


Figure 6.2: A snapshot from the prototype before any inputs has been written. Piano partly (CSS and HTML) from [17]. Prototype available at [18].

7
Results

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

8

Conclusion

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

9

Timeplan

1: SW4-SW5	- Supervision 1 with fackspråk 12/2 - Finishing planning report - Deadline planning report 15/2 - Split task into smaller subtasks and distribute these - Start working with GF, integration with Python (visualisation and music generation) - Start writing report - Start working with visualisation
2: SW6-SW7	- Checkup and interim of report - Information lab at the library 26/2 - Prepare half-time presentation - Oral half-time presentation 5/3 - Write self-evaluation - Deadline self-evaluation 8/3 - Start working on web application - Develop and complexify for fractals in GF - Continue writing/revising the report - Develop the visualisation
3: SW8-SW1	- Checkup and interim of report - Continue developing the final parts of the product - Continue writing and revising the report - If on schedule, start working on piano frontend - Pause for studying for exams - After exams, restart the project by doing a checkup and re-planning
4: SW2-SW3	- Product finished before Easter, possibly some final details left - Evaluation of the product - Focus on finishing the report
5: SW4-SW5	- Supervision 2 with fackspråk 16/4 - Write report - Deadline pre-evaluation report 2/5
6: SW6-SW7	- Write report - Deadline final report 16/5 - Deadline title 20/5 - Exhibit 21/5 - Deadline written individual opposition 23/5 - Begin preparing final presentation
7: SW8-End	- Finalise report - Continue preparing and rehearse the final presentation - Final presentation 28-29/5 - Write a personal evaluation - Personal evaluation submission 30/5 - Final submission of report 9/6

Bibliography

[1] K. Angelov, " Grammars for music/poetry/fractals ", 2018. [Online]. Available: https://www.chalmers.se/sv/institutioner/cse/utbildning/Grundutbildning/kandidatprojekt/Sidor/DATX02_19_26.aspx, downloaded: 2019-02-01.

[2] Wikipedia,"L-system" 2018. [Online]. Available: https://en.wikipedia.org/wiki/L-systemhttps://en.wikipedia.org/wiki/L-system, downloaded: 2019-02-04.

[3] Source Making, "Bridge Design Pattern". [Online]. Available: https://sourcemaking.com/design_patterns/bridgehttps://sourcemaking.com/design_patterns/bridge downloaded: 2019-02-05.

[4] K. Angelov, "Graftals", GrammaticalFramework/gf-contrib. [Online]. Available: https://github.com/GrammaticalFramework/gf-contrib/tree/master/graftals. Accessed: 2019-02-06.

Bibliography

[1] K. Angelov. *Grammars for music/poetry/fractals*. Available at https://www.chalmers.se/sv/institutioner/cse/utbildning/Grundutbildning/kandidatprojekt/Sidor/DATX02_19_26.aspx (2019/02/01).

[2] B. B. Mandelbrot. *The fractal geometry of nature*. New York, United States: W.H. Freeman and company, 1983.

[3] K. Falconer. *Fractal Geometry : Mathematical Foundations and Applications*. 3rd ed. University of St Andrews, United Kingdom: John Wiley & Sons, Incorporated, 2013.

[4] A. Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. 1st ed. Stanford University, United States: CSLI Publications, 2011.

[5] J. Hopcroft, J. Ullman, and R. Motwani. *Introduction to Automata Theory, Languages, and Computation*. 2nd ed. Addison-Wesley, 2000.

[6] D. Stirzaker G. Grimmett. *Probability and Random Processes*. 3rd ed. University of Oxford, United Kingdom: Oxford university press, 2011.

[7] P. Prusinkiewicz and J. Hanan. *Lindenmayer Systems, Fractals, and Plants*. Vol. 79. Lecture Notes in Biomathematics. New York, United States: Springer Science & Business Media, 1989.

[8] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty Of Plants*. 2nd ed. Available at http://algorithmicbotany.org/papers/#abop (2019/02/04). New York: Springer-Verlag, 1996.

[9] D. Shiffman. *The Nature of Code*. Available at https://natureofcode.com/book/(2019/02/04). Cambridge MA, 2012.

[10] E. Mohn. *Musical Instrument Digital Interface (MIDI)*. Available at http://eds.b.ebscohost.com/eds/detail/detail?vid=1&sid=b9836e91-8425-4d98-8abd-31561d6c261c%40sessionmgr102&bdata=JnNpdGU9ZWRzLWxpdmUmc2NvcGU9c2l0Z3d%3d#AN=87323150&db=ers(2019/02/07).

[11] Python Software Foundation. *turtle — Turtle graphics*. Available at https://docs.python.org/3.3/library/turtle.html?highlight=turtle (2017/09/19).

[12] A. Colón. *Sierpinski Triangle Curve.svg*. Available at https://commons.wikimedia.org/wiki/File:Sierpinski_Triangle_Curve.svg (2019/02/13).

[13] Wrtlprnft. *Koch Snowflake 7th iteration.svg*. Available at https://commons.wikimedia.org/wiki/File:Koch_Snowflake_7th_iteration.svg (2019/02/13).

[14] W. Beyer. *File:Mandel zoom 11 satellite double spiral.jpg*. Available at `https://commons.wikimedia.org/wiki/File:Mandel_zoom_11_satellite_double_spiral.jpg` (2019/02/13).

[15] K. Angelov. *Graftals*. Available at `https://github.com/GrammaticalFramework/gf-contrib/tree/master/graftals` (2019/02/06).

[16] Brython. *Brython Info*. Available at `http://www.brython.info/` (2019/02/15).

[17] E. Jarrell. *Create a Piano App with JavaScript*. Available at `https://hackernoon.com/create-a-piano-app-with-javascript-97dbad1ff28c` (2019/02/14).

[18] A. Bergsten. et al. *Prototype*. Available at `https://github.com/Gurr1/Grammars-for-music-poetry-fractals` (2019/02/15).