AlphaXmas: A Final Project for Creative Programming 2020-2021

Matteo Bernardini (10743181) Yilin Zhu (10702368)

Abstract

This project, AlphaXmas, is a multimedia installation aimed to bring the audience into an experience of watching a computer generated tree growing, with a computer generated music melody in the background. The growing plant is generated using a 3D model built from an L-system and the music melody is generated using a LSTM neural networks. The generated music melodies are short monophonic Christmas carol compositions obtained from a LSTM-RNN Model. We train the model using the Hymns and Carols of Christmas dataset with around 1k historical compositions. The final project is available on `https://github.com/bubblefishstudio/alphaXmas`

Keywords: creative programming, plant development, computer graphics, modeling of plant, animation through simulation, LSTM, music generation, animation and simulation

1 Introduction

Creative Programming provides us with the chance to design and develop a project to discover something interesting rather than something functional. Our goal is largely inspired by the beauty of computational creativity, we want to explore how to create a live graphical animation and live music in this project. Discovering how to generate a tree structure and designing a computer generated melody, especially in the final stage of completing this project, turns out to be a difficult but an awarding research path.

In the graphical part, we choose to generate a tree on a 3D canvas. The task we are faced with is in the field of grammar and formal languages, plant development, and computer graphics. As for the computer generated music part, the task is in the field of artificial intelligence, especially LSTM neural networks. Also, for the interaction between human and computer, we choose to add a few interactive options on the 3D canvas part.

1.1 Plant Development

According to our research, L-systems have been widely used since 1960s as a formalism language to describe and model a plant. The hidden beauty of nature in botany later stimulated the interest of computer scientists in formal languages or in the field of computer graphics. More results came out in simulating the growing process of a plant with more mathematical theory and the extension of L-systems [7].

In the formalism of L-systems, plants are modelled by a sequence of symbols, obtained by starting from an axiom and applying some production rules, or rewriting rules. The production rules in L-systems are used to define how the plant should iterate to form more complex structure in the next time stage. Branching structures are modelled using bracketed strings, where the matching pairs of brackets `[` and `]` is used to delimit branches. Moreover, the graphical interpretation of L-systems has also been used since then [2]. After defining the structure of a plant, a Turtle system interprets the symbol used in the L-system as a command to draw the branches and leaves of the plant on a graphical canvas or a 3D canvas.

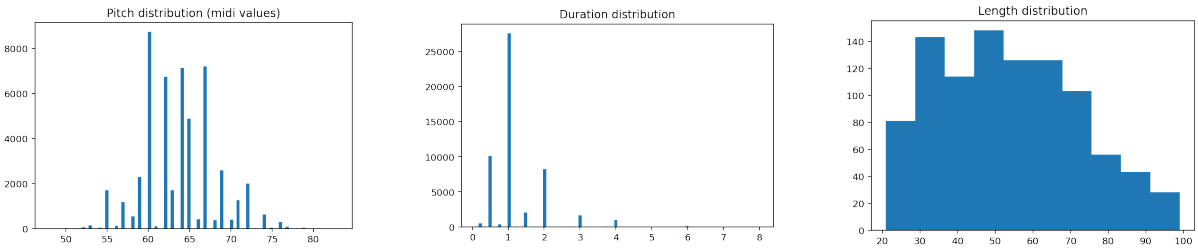

Figure 1: Distributions in filtered dataset

1.2 Music Generation

Deep learning has become a mature technology because it benefits from the development of data, hardware and software environment in recent years. They are all becoming pervasive and supportive. LSTM is a kind of Recurrent Neural Network structure used in the field of deep learning. Tasks with sequential data are suited to be solved by RNNs, because of the feedback connections in their design. Researchers have been applying LSTM model in dealing with text, speech and audio data sets for some years and have already achieved promising results. Applying LSTM in automatic music generation has also been proved to possible in the past years [8][4].

2 Music Part

We employ artificial intelligence in order to obtain non-trivial computer generated music. In this field, the usual process consist in designing a suitable model for the task, collecting and processing an appropriate dataset for the training of the model and finally evaluating the trained model over relevant parameters.

The task we are faced with is the generation on a monophonic melody. Recurrent Neural Networks provide the basis for this kind of task, since they are suited to generate sequences of correlated information. In particular, Long Short-Term Memory architectures are suited to the generation of music melodies, since they are better fitted at capturing long-term dependencies, like musical context, because they don't suffer from the gradient vanishing problem.

2.1 Data Set

Since our goal is to generate Christmas melodies, we found the dataset *ABC Christmas Carols and Hymns*[1], which contains around 1.6k historical compositions in ABC notation suitable for Christmas mood.

We performed some further processing on the dataset in order to simplify the training and get more accurate results. In particular, we filtered music pieces that are not in 4/4, we transposed everything to C major, removed chords, extracted individual voices, removed melodies which were too low or too high in octave (since they would have different musical contour, e.g. an accompanying bass line) and removed melodies that are too short or too long. After this process, we obtained a dataset of slightly less than 1k samples, with distributions in fig. 1.

[1] `http://www.stephenmerrony.co.uk/ABC/Carols/`

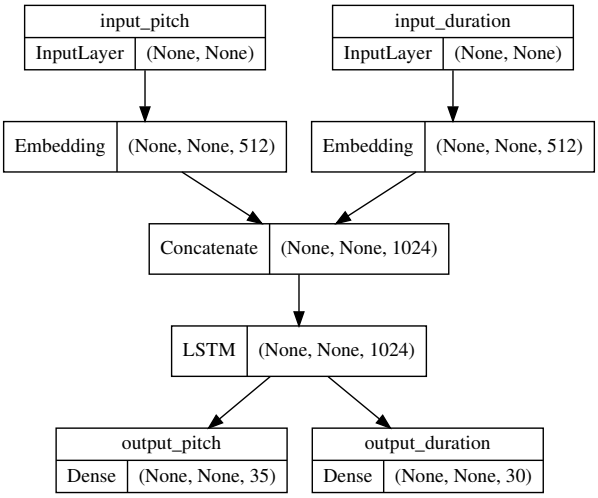

Figure 2: RNN model for music melody generation

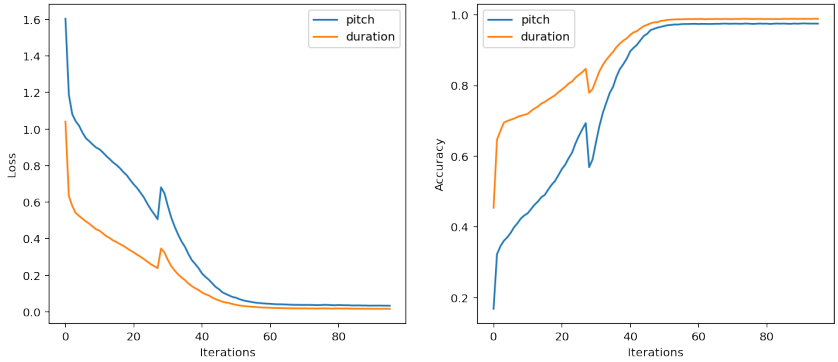

Figure 3: RNN training measures

2.2 RNN Modelling

For our task, the chosen domain is the one of melodies defined as sequences of tuples $\langle p, d \rangle$, where p is the pitch expressed as midi note (with -1 indicating a rest) and d is the duration expressed as quarter note ratio. In order to capture pitch and duration information separately, yet contextually, we decided to design a 2-input 2-output model, as depicted in fig. 2.

In particular, two input layers are used for pitch and durations, followed by two embedding layers. The embedding layers are concatenated and fed to the LSTM layer, which has a sigmoid activation function to model non-linearities. Finally, two Dense layers are used to sample the two categorical distributions of pitches and durations.

2.3 Training and Results

The model is trained to predict the same melodies which are fed as input. This is done by generating a validation dataset as the shifted version of the training dataset (i.e. anticipated by a single time-step). For each iteration, the sparse categorical cross-entropy is used as loss function, since the output is obtained as the sampling of two correlated categorical distributions.

Loss optimization is performed using the Adam algorithm, which computes the gradients needed to adjust the weights of the network in order to obtain the minimum loss. As shown in fig. 3, we obtained an optimal model after around 95 iterations.

`F`	make a stroke by moving forward
`f`	move forward
`^ &`	pitch up/down
`/ \`	roll clockwise/anticlockwise
`+ -`	yaw right/left
`|`	turn around (i.e. yaw 180°)
`[ ]`	save/reset turtle state to/from stack

Table 1: Standard turtle interpretation. Parameters indicate the length of motions or the angle of rotations

3 Graphical Part

While the melody is being played, the audience can observe the growth of a Christmas tree. We choose to model an approximation of Picea abies[2] by exploiting an L-system to model its growth. A turtle interpretation is used to convert the sentences into a set vertices of a 3D space, which are finally rendered in a canvas using a suitable 3D engine.

3.1 Parametric L-system

For our project we use a *parametric L-system*, which is a type of generative grammar [1] characterized by recursive production rules, leading to an infinite rewriting sequence, and a parameter associated to each symbol of the alphabet. Formally:

$$L = \langle G, \omega, P \rangle \tag{1}$$

where G is the alphabet, ω is the axiom sentence and P is the set of production rules. Each symbol of the alphabet is in the form A_n where n is the parameter. Production rules are therefore functions of n, for example:

$$A_n \longrightarrow B_n\, C_{n-1} \tag{2}$$

The L-system defined in fig. 4 is used to repeatedly generate sentences representing turtle movements to obtain each growing step of the tree.

3.2 Turtle Interpretation

A turtle interpretation is needed to convert each rewriting step of the L-system (fig. 4) into a set of vertices in a 3D space. A turtle state is identified by the turtle current position and its current orientation in space, given by three versors indicating the head, the right side and the ground directions in respect to the turtle (see fig. 5).

Each symbol of a rewriting step (sentence) of the L-system is then interpreted as a movement or rotation of the turtle, according to a variation of the standard interpretation in table 1. In particular the following symbols are added or modified according to our specific case:

`F(n):` draws a branch stroke of length n

`L(n):` draws a leaf stroke of length n

`O(p):` adds a light to the current turtle position with a probability p

`*:` places the star at the current turtle position

[2] `https://en.wikipedia.org/wiki/Picea_abies`

```
ω:
!(2) H T X *

P:
-- trunk --
H    →  F(0.4) ! H ?
T    →  /(30 + r*10) ! D(3.2) B B B T
B    →  /(120 + r*20) [ | ^(50 + r*10) D(2.5) ! Y ? ] D(2 + r*0.5)

-- branches --
Y    →  X I ! V ?
I    →  +(r*4 - 2) X !(0.6) I ?(0.6)
V    →  D(0.2) [ /(20) +(50) X ! I I ? ]
        D(0.1) [ \(20) -(50) X ! I I ? ]
        !(0.8) Y ?(0.8)

-- leafs --
X    →  ^(r*3) E E E E E E D(0.05) O(0.3)
E    →  /(60) [ &(70) L(0.5) ] F(0.05)

-- delay stroke --
D(n) →  if n > 1 then D(n-1) else F(n)
```

Figure 4: L-system used to generate the tree animation. r is a random value in the range $[0, 1)$, used to add angle and length variability to each growing branch

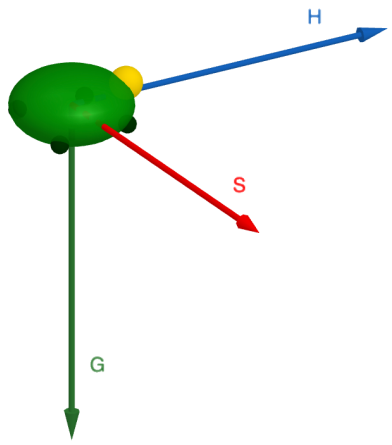

Figure 5: Turtle in space with head, side and ground versors

Any other symbol of the sentence that is not in the table is simply ignored and does not update the turtle state.

Pitch, roll and yaw are rotations about the side, head and ground axes respectively, and their parameter is expressed in degrees. Stroke/movements parameters are expressed as the ratio of a predefined unit distance, which is computed based on the canvas size.

Square brackets are used to create a stack of saved turtle states. This approach simplifies the creation of branches, by allowing to save a turtle state, draw a branch and then reset the turtle to the previous position and orientation.

3.3 3D Model Computation

Given a 3D Cartesian frame of reference, the turtle has its initial position at the origin and its original orientation is given by the head, side and ground versors being aligned respectively with the z-axis, x-axis and y-axis. In practice, a turtle state T_n can be univocally identified by a position vector and a quaternion indicating the orientation:

$$T_n = \langle \vec{\boldsymbol{P}}_{\boldsymbol{n}}, q_n \rangle \tag{3}$$

$$\vec{\boldsymbol{P}}_{\boldsymbol{0}} = (0, 0, 0) \tag{4}$$

$$q_0 = 1 + 0i + 0j + 0k \tag{5}$$

We chose to use quaternions since they are mathematically simple and stable (i.e. they are easy to re-normalize and they don't suffer from the gimbal lock problem) [3]. Orientation updates can be then simply performed in terms of quaternion pre-multiplication, and the three orientation versors can be obtained by rotating accordingly the reference frame axes to the current orientation.

Finally, turtle positions are collected at each stroke symbol to create the sets of branch vertices and leaf vertices needed to render the shape in the canvas.

3.3.1 Quaternions

Given the function $\text{Pure}(z) = (\Im_i(z), \Im_j(z), \Im_k(z))$ which extracts the vector component of a quaternion z, and using the notation z^* to indicate the complex conjugate of z, the orientation turtle axes can be obtained as follows:

$$\hat{\boldsymbol{H}}_{\boldsymbol{n}} = \text{Pure}(q_n \cdot k \cdot q_n^*) \tag{6}$$

$$\hat{\boldsymbol{S}}_{\boldsymbol{n}} = \text{Pure}(q_n \cdot i \cdot q_n^*) \tag{7}$$

$$\hat{\boldsymbol{G}}_{\boldsymbol{n}} = \text{Pure}(q_n \cdot j \cdot q_n^*) \tag{8}$$

Position updates caused by turtle movements `F(d)`, `L(d)`, `f(d)` can be therefore obtained as follows:

$$\vec{\boldsymbol{P}}_{\boldsymbol{n+1}} = \vec{\boldsymbol{P}}_{\boldsymbol{n}} + d\hat{\boldsymbol{H}}_{\boldsymbol{n}} \tag{9}$$

while orientation updates caused by turtle rotations of parameter α about the axis $\hat{\boldsymbol{u}}$, with ρ representing the radians-equivalent of α, can be obtained as follows:

$$q_{n+1} = r \cdot q_n \qquad r = \cos\left(\frac{\rho}{2}\right) + \sin\left(\frac{\rho}{2}\right)(i, j, k) \cdot \hat{\boldsymbol{u}} \tag{10}$$

Figure 6: Rendered tree with lights and star

3.3.2 Final shape

The 3D model of the tree is defined by a set of pairs of vertices identifying the segments composing the tree trunk and branches V_b and a set of pairs of vertices identifying the segments composing the leaves V_l. Additionally, a set of vertices for lights positions V_x and a single vertex for the position of the star V_s are needed to add decorations. The final tree shape is thus obtained by rendering the segments of each of the two sets V_b and V_l in different colors and different stroke weight (thicker brown for the branches, thinner green for the leafs), and by rendering colored dots for every vertex in V_x and by rendering a star in position V_s according to the interaction rules (see section 4).

In particular for each `F(d)` or `L(d)` symbol moving the turtle from state T_n to state T_{n+1}, the segment $\langle \vec{\boldsymbol{P_n}}, \vec{\boldsymbol{P_{n+1}}} \rangle$ is added to V_b or V_l respectively. In reality, a simplification step is also performed in order to merge consecutive segments thus reducing the number of needed vertices.

4 User Interaction

In order to capture the interest of the audience, the canvas is animated using a virtual camera system. In particular, the nose of the viewer is tracked using the webcam of the device and its position is used to update the $\langle x, z \rangle$ coordinates of the camera (which is kept at a constant distance y from the center of the frame of reference). This interaction allows the user to view the tree from different angles and provides a 3D illusion of the tree. The view is also continuously and slowly rotated about the z-axis in order to show every branch of the growing tree.

Additionally, the tree is decorated with lights of four different colors. Every light alternates among the four colors every time a note is being played, effectively linking the animation to the melody playback. This effect simulates the usual decorations added to Christmas trees.

Finally, after the tree stops growing, a star might appear on top of the tree depending on whether the generated melody is "good enough". The used criterion to determine if a melody is "good" is whether the melodic sequence contains a cadence in the form II-V-I, VII-I or II-I. If present, the star will appear on top of the tree at the same time the cadence is played. This last interaction makes every experience unique, since both the tree and the melody are generated randomly every time the application is loaded. An example of rendered tree with lights and star is depicted in fig. 6.

5 System Design and Implementation

The project is implemented as a single-page Web Application, built using *webpack*. At page load, a loading animation is displayed and when the melody and tree model are loaded, a start button is displayed to let the user start the growing animation and music playback. Autoplay is avoided because of autoplay-blocking policies commonly found in modern browsers[3].

The music is generated from a previously trained *Keras* model which is loaded via JavaScript on page load. The *Keras* model was prepared using a *Jupyter* Notebook[4], using *music21* for dataset analysis and processing. Playback is achieved using *music21j* by converting each $\langle \text{note}, \text{duration} \rangle$ tuple to actual MIDI notes, which are rendered using a Music Box Soundfont[5].

The graphical rendering is performed on a HTML `<canvas>` using *p5.js*. In order to optimize performance, every growing step of the tree is pre-computed during the loading stage, by running the L-system rewriting and turtle interpretation. In particular, classes `Grammar`, `Turtle`, `Quaternion`, `Star` and `Tree` have been implemented: for every growing step, suitable `p5.Geometry` objects (for tree branches and tree leaves) and arrays of vertices (for light points and star position) are computed. At every `draw` cycle the needed objects are then referenced using the `model` and `point` primitives of *p5.js* This approach has the advantage of leveraging the GPU for the actual rendering, but it's not documented[6].

Finally, face detection is performed using *Keras PoseNet* library, by accessing to the video stream of the front camera using `createCapture` primitive of *p5.js*. An `Observer` class is implemented to keep track of the position of the nose, which is used to update the position of the camera at every `draw` cycle.

6 Future developments

Right now, the animation of growing tree is not smooth because the model is discrete in time states. We know other researchers have discovered how to make a smooth animation of plant development in continuous time states using a differential mathematical model like differential L-systems [6] or timed L-system [5]. In those advanced model, they overcome the limits of simple L-system and can capture continuous growing process on a single branch or internodes to make time-lapse style animation.

Also, limited by the equipment and the time we have for this project, not all the interesting design we have thought about at the beginning of the Hackathon day turned out to be implemented. In the future, this project has the potential to be extended to a Virtual Reality application. With more help of physical sensors on the location and movement of the audience, the graphical perspective of the project can be improved to provide a better emerging experience: the audience's view of the tree will be more realistic with the camera moving together with the perspective from the audience.

As for the generated music, the model can be improved by doing further processing on the dataset. The playback can be extended to 3D audio, by taking advantage of the distance and location between the audience and the tree.

[3] `https://developer.mozilla.org/en-US/docs/Web/Media/Autoplay_guide`
[4] `https://github.com/bubblefishstudio/alphaXmas/blob/main/colab/RNN_Model_Test.ipynb`
[5] `https://github.com/gleitz/midi-js-soundfonts/tree/gh-pages/MusyngKite/music_box-mp3`
[6] `https://github.com/processing/p5.js/issues/5393`

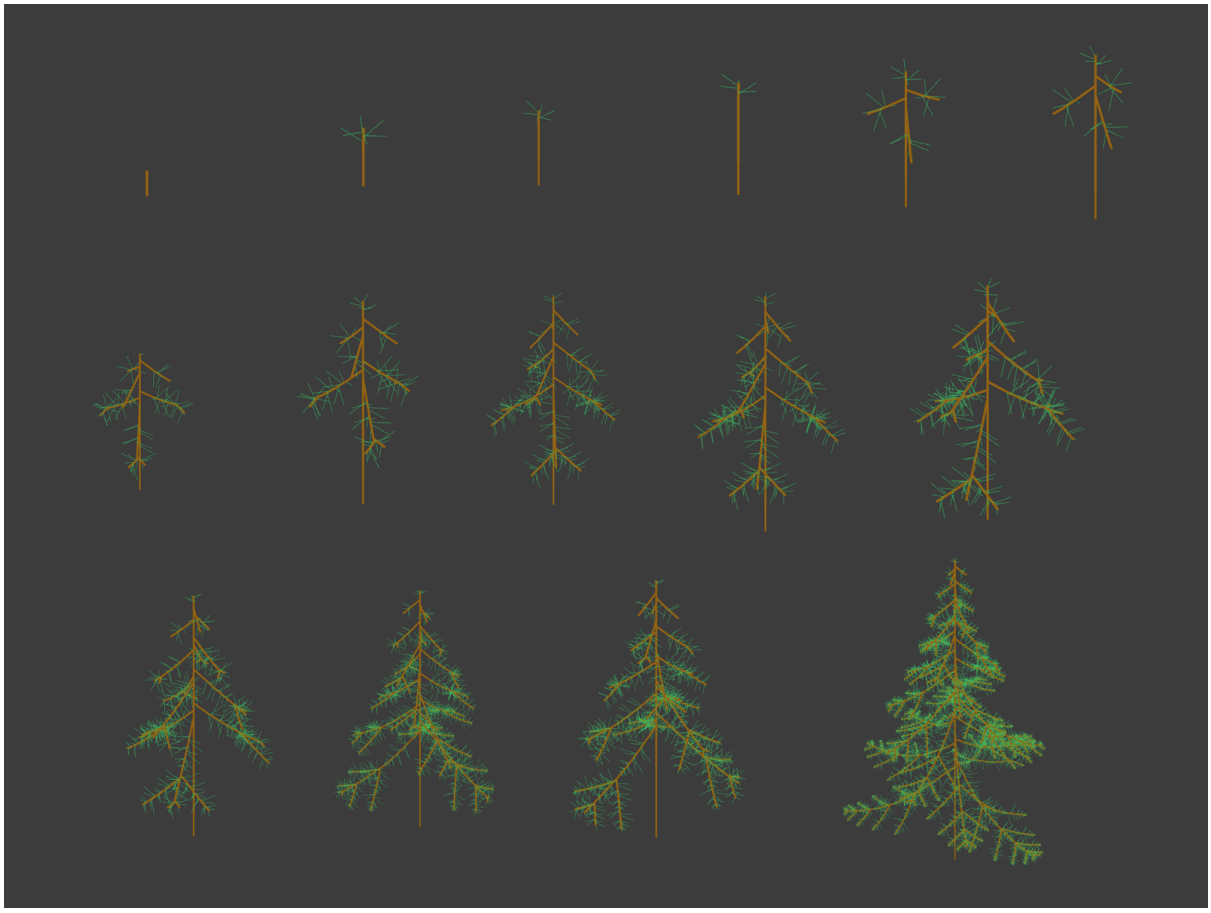

Figure 7: Growing steps of the tree

7 Conclusions

In this project, we created a tree animation in a parametric, bracketed extension of L-system model. At the same time, a Christmas style music melody is generated in real-time from an LSTM neural network model and played back using a music box sound. The final result of the tree animation is depicted in fig. 7, by showing some snapshoots of the growing stage in a discrete time.

As for the listening impression for the generated music, the result is generally considered to be suitable for Christmas. In general, the project provides a peaceful and happy impression from the audience's opinion.

Future works can be included both from the graphical part and the music part. The rendering of the tree can be improved by adding more texture and environmental lights. The LSTM model for generating music can also be improved. We finally hope to improve this project in order to be used in a Christmas exhibition or some other event.

References

[1] Noam Chomsky. "Three models for the description of language". In: *IRE Transactions on information theory* 2.3 (1956), pp. 113–124.

[2] James Hanan. *Parametric L-systems and their application to the modelling and visualization of plants*. Citeseer, 1992.

[3] Yan-Bin Jia. "Quaternions and rotations". In: *Com S* 477.577 (2008), p. 15.

[4] Qi Lyu et al. "Modelling high-dimensional sequences with lstm-rtrbm: Application to polyphonic music generation". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[5] Jon McCormack et al. "Interactive evolution of L-system grammars for computer graphics modelling". In: *Complex Systems: from biology to computation* (1993), pp. 118–130.

[6] Przemyslaw Prusinkiewicz, Mark S Hammel, and Eric Mjolsness. "Animation of plant development". In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. 1993, pp. 351–360.

[7] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer Science & Business Media, 2012.

[8] Falak Shah, Twisha Naik, and Nisarg Vyas. "LSTM based music generation". In: *2019 International Conference on Machine Learning and Data Engineering (iCMLDE)*. IEEE. 2019, pp. 48–53.

[9] Massimiliano Zanoni. *Creative Programming and Computing: Course material of MSc in Music and Acousting Engineering*. 2020.