Pontifical Catholic University of Rio de Janeiro

Generation Of Music For Games Integrated To The Plot With Deep Learning Techniques

Gustavo Amaral Costa dos Santos

Graduation Final Project

Center of Science and Technology

Department of Informatics

Undergraduate Course in Computer Engineering

Rio de Janeiro, December 2021

Gustavo Amaral Costa dos Santos

Generation Of Music For Games Integrated To The Plot With Deep Learning Techniques

Final Project Report, presented to the Computer Engineering Program at PUC-Rio as a partial requirement for obtaining the degree of Computer Engineer.

Advisor: Bruno Feijó

Co-advisor: Augusto Cesar Espíndola Baffa

Rio de Janeiro, December 2021

“Any sufficiently advanced technology is indistinguishable from magic.”

Clarke, Arthur C. Profiles of the Future: An Inquiry into the Limits of the Possible. 1973.

My Sincere Thanks To

My family, who always supported me through all my difficulties. This project was only possible because they believed in me from beginning to end.

To my advisors who accepted the challenge of guiding me in this project, always helping me in every way possible.

To my friends who were present with me throughout my academic journey helping whenever they could. Especially to Rodrigo, André and Iago, friendships from college that were always present before and during the project and that I will carry with me for the rest of my life.

To my friends from Teresópolis, Gabriel, Rafael and Ilon, who were always with me despite the constant absences due to distance.

To my friends Maria Carolina, Eduarda and Ana Clara who always listened to me in the most difficult moments, somehow bringing a smile to my face.

And last but not least, to my friends at work, who always helped me when I needed to focus on the project and couldn't fully dedicate myself to my assignments.

Abstract

Amaral, Gustavo. Feijó, Bruno. Baffa, Augusto. Generation Of Music For Games Integrated To The Plot With Deep Learning Techniques. Rio de Janeiro, 2021. 49p. Final Graduation Project Report – Center of Science and Technology, Department of Informatics. Pontifical Catholic University of Rio de Janeiro.

In this project, a study was developed regarding the generation of musical content for games through different deep learning techniques. In it, in addition to the construction of a theoretical framework to support future projects, the implementation of a system capable of parameterizing feelings, through the Arousal/Valence model, and thus, able to materialize the musical generation integrated to the plot, was also addressed. Therefore, for the generation of the musical content, the Transformer was used as deep learning model. Moreover, aiming to optimize it, the process was integrated with a multilayer music generation technique and the system itself was implemented in Typescript and Python with NestJS and TensorFlow/Magenta as main frameworks respectively.

Keywords

Musical Generation. Games. Deep Learning. Transformer. Multilayer. Arousal Valence.

Resumo

Amaral, Gustavo. Feijó, Bruno. Baffa, Augusto. Geração de Música Para Jogos Integrada Ao Enredo Com Técnicas De Aprendizado Profundo. Rio de Janeiro, 2021. 49p. Relatório do Projeto Final de Graduação – Centro Técnico Científico, Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Neste projeto foi desenvolvido um estudo referente a geração de conteúdo musical para jogos por meio de diferentes técnicas de aprendizado profundo. Nele, além da construção de um arcabouço teórico para a fundamentação de projetos futuros, abordou-se também a implementação de um sistema capaz de parametrizar sentimentos, por meio do modelo de Excitação/Valência, e assim, concretizar a geração musical integrada ao enredo. Por conseguinte, para a confecção da geração musical, utilizou-se o *Transformer* como modelo de aprendizado profundo. Em somatório, visando otimizá-lo, o processo foi integrado a uma técnica de geração musical multicamadas e o sistema em si foi implementado em Typescript e Python com NestJS e TensorFlow/Magenta como principais frameworks respectivamente.

Palavras-Chave

Geração Musical. Jogos. Aprendizado Profundo. Transformer. Multicamadas. Excitação Valência.

List Of Images

List Of Tables

Summary

---

1. Introduction

Video games are an ever-growing part of our world. As the industry evolves, the quality and standards of video games are improving rapidly, making it an everlasting experience for gamers. That said, there are many factors that contribute to these enhanced environments. Among them, sound effects stand out as one of the most important features to increase the players' immersion.

In a game, we have visual, narrative and musical manifestations intertwined. However, this entanglement does not reach its full potential, as it does not take into account parameters inherent to the player himself, and his interactions with several other elements of the game. With this in mind, using deep learning techniques, this project aims to explore the scientific and technical challenges of generating music for games, intending to investigate the integration of the game's plot with the musical narrative.

Not only is this topic innovative, but the area of games in computing itself is not very explored, with few specialists, which makes the present project a strategic opportunity of research. Furthermore, the importance to explore this research theme arises from a wide spectrum of needs, ranging from cost reduction and deadlines during the development of games until the realization of interactive storytelling integrated to the narrative musical. In this project, computer games refer to video games in their various genres, excluding board games (e.g. Chess, Go). As for the various purposes that a game can have, the present project refers to games in a broad way, whether they are games for entertainment or for more general applications like “serious games” type.

2. Current Situation

Recent deep learning techniques, which are the recent evolution of artificial neural networks and are part of the arsenal of artificial intelligence techniques, have become reference techniques for various applications, such as face or voice recognition, translation, synthesis of voice, weather, games, etc. A recent area of application is the creation of content (music, image, text, etc.), which has become increasingly important on the international stage, as witnessed by the recent arrival of “big players”, such as Google (with the creation, in 2017, of the Magenta Project[1], focused on artistic creation, especially music) and Spotify (with the creation, in 2018, of the Creator Technology Research Lab[2], focused on musical creation). Also noteworthy is the recent creation of several micro-companies dedicated to the potential market (AIVA[3], Amper Music[4], Hexachords[5] etc.), to create music for videos and documentaries.

The basic idea is to learn a musical style from a corpus of musical examples (for example, Celtic melodies, choirs by J. S. Bach[6], jazz harmonies etc.) and then generate new music from the learned style. For example, supervised learning techniques and feedforward architectures can be used, for example, to generate polyphonic accompaniments (counterpoint) in the style of Bach's choirs. Systems such as DeepBach (Hadjeres et al. 2017) pass the musical Turing test[7]. Unsupervised learning techniques and new types of generative architectures, such as variational autoencoders (Doersch 2016) or antagonistic architectures (generative adversarial networks) (J. Goodfellow et al. 2014), can be used to generate new music inspired by the learned style. Several current techniques and research challenges focus on the ability to control the generation (for example, to follow a pre-existing harmony, to adjust the number of notes, etc.), to impose or emerge a structure and to encourage originality (Briot and Pachet 2018).

A domain with great potential is the generation of music for games. Modern games have very sophisticated visual characteristics and also complex and interactive narratives that are complemented by sounds and soundtracks.

---

[1] Magenta Project website: https://magenta.tensorflow.org/
[2] Spotify for Artists website: https://artists.spotify.com/
[3] AIVA website: https://www.aiva.ai/
[4] Amper Music website: https://www.ampermusic.com/
[5] Hexachords website: http://hexachords.com/
[6] Johann Sebastian Bach was a German composer and musician of the late Baroque period.
[7] Difficult to decide whether the composer is human or artificial

However, the quality of current soundtracks is usually limited by two reasons: the track is generated from relatively simple models; or the musical quality is good but the soundtrack sound is predefined and with little capacity for dynamic adaptation to the context of interaction with the player(s).

Also there are very few publications on automatic music generation for (or with) games. For example, Prechtl et al. 2014 propose the generation of chord sequences influenced by the danger of the current situation[8]. This emotional parameter influences the generation parameters[9] by a model of Markov via a change in the values of the transition matrix. Engels et al. 2015 present another system, also based on a hierarchical Markov model, to model various sections of the narrative, but without the interactivity of the control proposed by the predicted system. Another approach, tried previously by Prof. Feijó (Soares de Lima et al. 2005), consists of the dynamic and automatic selection of predefined musical excerpts, depending on the current situation in the game. A technique used in practice by professionals is “layered” production, with the recording separately from the string part, the metal part and percussions in order to dynamically modulate the orchestration (for example, adding metals and percussions to emphasize a dramatic situation) (Stuart 2019), but with the score remaining static.

[8] The metric used is the distance between the player and the enemies.
[9] Time, volume, major or minor chord, etc.

3. Proposal And Objectives

The main objective of this project is to explore the scientific and technical challenges of music generation for games using deep learning techniques. The project intends to present a critical view of this new area and propose new research directions. In particular, this project aims to investigate the integration of the game's plot with the musical narrative.

In this context, we intend to map the parameters of the game's narrative[10] to the control of the generation of the music, through a conditioned neural network architecture.

As explained in the current situation section, there are very few publications on music generation for games and even then, with very preliminary experiments. Thus, due to its innovative content, this project will be of an exploratory type and with some "proof of concept".

[10] e.g. story genre, dramatic arc of the storyline, type of player, emotions, personality and relationships of the characters

4. Preliminary Studies

To evaluate options for automatic music generation, and acquire knowledge about the topic, a study based on (Briot et al. 2019) was made, where a multi-criteria conceptual framework based on five dimensions was evaluated: objective, representation, architecture, challenge and strategy.

This typology is aimed at helping the analysis of the various perspectives (and elements) leading to the design of different deep learning-based music generation systems. Therefore, allowing a better approach of the project implementation.

That said, the next sections analyze each one of these dimensions, since they will be the basis for most of the development in this project.

4.1 Objective

The first dimension, the objective, is the nature of the musical content to be generated, and we may consider five main facets of it:

4.1.2 Type

The musical nature of the generated content. Examples are a melody, a polyphony or an accompaniment.

4.1.3 Destination

The entity aimed at using (processing) the generated content. Examples are a human musician, a software or an audio system.

4.1.4 Use

The way the destination entity will process the generated content. Examples are playing an audio file or performing a music score.

4.1.5 Mode

The way the generation will be conducted, i.e. with some human intervention (interaction) or without any intervention (automation).

4.1.6 Style

The musical style of the content to be generated. Examples are Johann Sebastian Bach chorales, Wolfgang Amadeus Mozart sonatas, Cole Porter songs or Wayne Shorter music. The style will actually be set though the choice of the dataset of musical examples (corpus) used as the training examples.

4.2 Representation

The representation is the nature and format of the information (data) used to train and to generate musical content. Examples are signal, transformed signal[11], piano roll, MIDI or text.

4.2.1 Audio

The first type of representation of musical content is audio signal, either in its raw form (waveform) or transformed.

4.2.1.1 Waveform

The most direct representation is the raw audio signal: the waveform. The advantage of using a waveform is in considering the raw material untransformed, with its full initial resolution. Architectures that process the raw signal are sometimes named end-to-end architectures. The disadvantage is in the computational load: low level raw signal is demanding in terms of both memory and processing.

4.2.1.2 Transformed

Using transformed representations of the audio signal usually leads to data compression and higher-level information, but at the cost of losing some information and introducing some bias. A common example is the spectrum, obtained via a Fourier transform. Another option is the chromagram, a variation of the spectrogram, discretized onto the tempered scale and independent of the octave.

4.2.2 Symbolic

Symbolic representations are concerned with concepts like notes, duration and chords, which will be introduced in the following sections.

---

[11] e.g. a spectrum, via a Fourier transform

4.2.2.1 Note

In a symbolic representation, a note is represented through the following main features, and for each feature there are alternative ways of specifying its value, such as pitch, duration and dynamics.

4.2.2.2 Rest

Rests are important in music as they represent intervals of silence allowing a pause for breath. A rest can be considered as a special case of a note, with only one feature, its duration, and no pitch or dynamics. The duration of a rest may be specified by: absolute value, in milliseconds (ms); or relative value, notated as a division or a multiple of a reference rest duration, the whole rest having the same duration as a whole note.

4.2.2.3 Interval

An interval is a relative transition between two notes.

4.2.2.4 Chord

A chord is a set of at least 3 notes (a triad).

4.2.2.5 Rhythm

Rhythm is fundamental to music and conveys the pulsation as well as the stress on specific beats. Rhythm introduces pulsation, cycles and thus structure in what would otherwise remain a flat linear sequence of notes. We may consider three different levels in terms of the amount and granularity of information about rhythm to be included in a musical representation for a deep learning architecture: None, only notes and their durations are represented, without any explicit representation of measures, which is the case for most systems; Measures, measures are explicitly represented; Beats, information about meter, beats, etc.

4.2.3 Format

The format is the language (i.e. grammar and syntax) in which a piece of music is expressed (specified) in order to be interpreted by a computer.

4.2.3.1 MIDI

Musical Instrument Digital Interface (MIDI) is a technical standard that describes a protocol, a digital interface and connectors for interoperability between various electronic musical instruments, softwares and devices. MIDI carries event messages that specify real-time note performance data as well as control data.

4.2.3.2 Piano Roll

The piano roll representation of a melody (monophonic or polyphonic) is inspired from automated pianos. This was a continuous roll of paper with perforations (holes) punched into it. Each perforation represents a piece of note control information, to trigger a given note. The length of the perforation corresponds to the duration of a note. In the other dimension, the localization of a perforation corresponds to its pitch.

4.2.3.3 Text

Music can also be formatted as text, below we introduce some important formats analyzed in this project.

4.2.3.3.1 Melody

A melody can be encoded in a textual representation and processed as a text.

4.2.3.3.2 Chord and Polyphony

When represented extensionally, chords are usually encoded with simultaneous notes as a vector. Another representation is to represent chords horizontally, as sequences of constituent notes, i.e. a chord is represented as an arbitrary length-ordered sequence of notes, and they are separated by a special symbol, as with sentence markers in natural language processing.

4.2.3.3.3 Markup Language

This is the case of general text-based structured representations based on markup languages, like for instance the open standard MusicXML[12].

[12] MusicXML is the standard open format for exchanging digital sheet music. It allows you to collaborate with musicians using different music applications.

4.2.3.3.4 Lead Sheet

Lead sheets are an important representation format for popular music (jazz, pop, etc.). A lead sheet conveys in upto a few pages the score of a melody and its corresponding chord progression via an intensional notation. Lyrics may also be added. Some important information for the performer, such as the composer, author, style and tempo, is often also present.

4.2.4 Temporal Scope

An initial design decision concerns the temporal scope of the representation used for the generation data and for the generated data, that is the way the representation will be interpreted by the architecture with respect to time.

4.2.4.1 Global

In this first case, the temporal scope of the representation is the whole musical piece. The deep network architecture (typically a feedforward or an autoencoder architecture) will process the input and produce the output within a global single step.

4.2.4.2 Time Step

In this second case, the most frequent one, the temporal scope of the representation is a local time slice of the musical piece, corresponding to a specific temporal moment (time step). The granularity of the processing by the deep network architecture (typically a recurrent network) is a time step and generation is iterative. Note that the time step is usually set to the shortest note duration, but it may be larger.

4.2.4.3 Note Step

In this approach there is no fixed time step. The granularity of processing by the deep network architecture is a note. This strategy uses a distributed encoding of duration that allows to process a note of any duration in a single network processing step. Note that, by considering as a single processing step a note rather than a time step, the number of processing steps to be bridged by the network is greatly reduced.

4.2.5 Meta Data

In some systems, additional information from the score may also be explicitly represented and used as metadata, such as: note tie; fermata; harmonics; key; meter; and the instrument associated to a voice.

4.2.5.1 Note Hold Or Ending

An important issue is how to represent if a note is held, i.e. tied to the previous note. This is actually equivalent to the issue of how to represent the ending of a note.

4.2.5.2 No Enharmony (Note Denotation)

Most systems consider enharmony, i.e. in the tempered system A♯ is enharmonically equivalent to (i.e. has the same pitch as) B♭, although harmonically and in the composer's intention they are different.

4.2.5.3 Feature Extraction

Although deep learning is good at processing raw unstructured data, from which its hierarchy of layers will extract higher level representations adapted to the task, some systems include a preliminary step of automatic feature extraction, in order to represent the data in a more compact, characteristic and discriminative form. One motivation could be to gain efficiency and accuracy for the training and for the generation. Moreover, this feature based representation is also useful for indexing data, in order to control generation through compact labeling, or for indexing musical units to be queried and concatenated.

4.2.6 Expressiveness

4.2.6.1 Tempo

If training examples are processed from conventional scores or MIDI-format libraries, there is a good chance that the music is perfectly quantized, i.e. note onsets are exactly aligned onto the tempo, resulting in a mechanical sound without expressiveness. One approach is to consider symbolic records, in most cases recorded directly in MIDI, from real human performances, with the musician interpreting the tempo.

4.2.6.2 Dynamics

Another common limitation is that many MIDI-format libraries do not include dynamics (the volume of the sound produced by an instrument), which stays fixed throughout the whole piece. One option is to take into consideration (if present on the score) the annotations made by the composer about the dynamics, from pianissimo ***ppp*** to fortissimo ***fff***. As for tempo expressiveness, another option is to use real human performances, recorded with explicit dynamics variation, the velocity field in MIDI.

4.2.7 Encoding

Once the format of a representation has been chosen, the issue still remains of how to encode this representation. The encoding of a representation (of a musical content) consists in the mapping of the representation, composed of a set of variables, e.g. pitch or dynamics, into a set of inputs[13] for the neural network architecture.

4.2.7.1 Value Encoding

Value encoding is to directly encode the variable as a scalar whose domain is real values (continuous variables). An example is the pitch of a note defined by its frequency in Hertz, that is a real value within the]0,+∞[interval. Or, encode the variable as a real value scalar, by casting the integer into a real (discrete integer variables). An example is the pitch of a note defined by its MIDI note number, that is an integer value within the {0,1,... ,127} discrete set. Or, lastly, encode the variable as a real value scalar, with two possible values: 1, for true, and 0, for false (boolean/binary variables). An example is the specification of a note ending.

4.2.7.2 One-hot Encoding

One-hot encoding is to encode a categorical variable as a vector having as its length the number of possible elements, in other words the cardinality of the set of possible values. Then, in order to represent a given element, the corresponding element of the encoding vector is set to 1 and all other elements to 0. This frequently used strategy is also often employed for encoding discrete integer variables, such as MIDI note numbers.

---

[13] Also named input nodes or input variables.

4.2.7.3 Many-hot encoding

Similar to one-hot encoding, but where all elements of the vector corresponding to the notes or to the active components are set to 1.

4.2.8 Dataset

The choice of a dataset is fundamental for good music generation. At first, a dataset should be of sufficient size[14] to guarantee accurate learning. If the dataset is very heterogeneous, a good generative model should be able to distinguish the different subcategories and manage to generalize well. On the contrary, if there are only slight differences between subcategories, it is important to know if the “averaged model” can produce musically-interesting results.

4.2.8.1 Transposition and Alignment

A common technique in machine learning is to generate synthetic data as a way to artificially augment the size of the dataset, in order to improve accuracy and generalization of the learnt model. In the musical domain, a natural and easy way is transposition, i.e. to transpose all examples in all keys. In addition to artificially augmenting the dataset, this provides a key (tonality) invariance of all examples and thus makes the examples more generic. Moreover, this also reduces sparsity in the training data. An alternative approach is to transpose (align) all examples into a single common key.

4.3 Architecture

The architecture is the nature of the assemblage of processing units (the artificial neurons) and their connexions. Examples are a feedforward architecture, a recurrent architecture, an autoencoder architecture and generative adversarial networks.

4.3.1 Feedforward

A multilayer neural network, also named a feedforward neural network, is an assemblage of successive layers of basic building blocks. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes, there are no cycles or loops in the network.

---

[14] i.e. contain a sufficient number of examples.

4.3.2 Autoencoder

An autoencoder is a neural network with one hidden layer and with an additional constraint: the number of output nodes is equal to the number of input nodes.

4.3.3 Variational

A variational autoencoder (VAE) has the added constraint that the encoded representation, the latent variables, by convention denoted by variable z, follow some prior probability distribution P(z). Usually, a Gaussian distribution is chosen for its generality. This constraint is implemented by adding a specific term to the cost function, by computing the cross-entropy between the values of the latent variables and the prior distribution. As with an autoencoder, a VAE will learn the identity function, but furthermore the decoder part will learn the relation between a Gaussian distribution of the latent variables and the learnt examples.

4.3.4 Restricted Boltzmann Machine (RBM)

A restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs.

4.3.5 Recurrent (RNN)

A recurrent neural network (RNN) is a feedforward neural network extended with recurrent connexions in order to learn a series of items (e.g., a melody as a sequence of notes). The input of the RNN is an element x_t of the sequence, where t represents the index or the time, and the expected output is the next element x_{t+1}. In other words the RNN will be trained to predict the next element of a sequence.

In order to do so, the output of the hidden layer reenters itself as an additional input (with a specific corresponding weight matrix). This way, the RNN can learn, not only based on the current item but also on its previous own state, and thus, recursively, on the whole of the previous sequence. Therefore, an RNN can learn sequences, notably temporal sequences, as in the case of musical content.

4.3.6 Convolutional

Convolutional neural network (CNN or ConvNet) architectures for deep learning have become commonplace for image applications. The concept was

originally inspired by both a model of human vision and the convolution mathematical operator. This resulted in efficient and accurate architectures for pattern recognition, exploiting the spatial local correlation present in natural images.

The basic idea is to slide a matrix (named a filter, a kernel or a feature detector) through the entire image (seen as the input matrix) and for each mapping position, compute the dot product of the filter with each mapped portion of the image, and then sum up all elements of the resulting matrix.

This will result in a new matrix (composed of the different sums for each sliding/mapping position), named “convolved feature”, or also “feature map”. The size of the feature map is controlled by three hyperparameters: depth (the number of filters used); stride (the number of pixels by which we slide the filter matrix over the input matrix); and zero-padding (the padding of the input matrix with zeros around its border).

4.3.7 Conditioning

The idea of a conditioning (sometimes also named conditional) architecture is to parametrize the architecture based on some extra conditioning information, which could be arbitrary, e.g., a class label or data from other modalities. The objective is to have some control over the data generation process. In practice, the conditioning information is usually fed into the architecture as an additional and specific input layer.

4.3.8 Generative Adversarial Networks (GAN)

The idea is to train simultaneously two neural networks: a generative model (or generator) G, whose objective is to transform a random noise vector into a synthetic (faked) sample, which resembles real samples drawn from a distribution of real images; and a discriminative model (or discriminator) D, which estimates the probability that a sample came from the real data rather than from the generator G.

This corresponds o a minimax two-player game, where one agent's gain is another agent's loss. The core idea of a GAN is based on the "indirect" training through the discriminator, which itself is also being updated dynamically. This basically means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.

4.3.9 Reinforcement Learning (RL)

The basics of reinforcement learning can be summarized by: an agent within an environment sequentially selects and performs actions in an environment; each action performed brings it to a new state; the agent receives a reward (reinforcement signal), which represents the fitness of the action to the environment (current situation); the objective of the agent being to learn a near optimal policy (sequence of actions) in order to maximize its accumulated rewards (named its gain).

4.3.10 Compound

Often compound architectures are used. Some cases are homogeneous compound architectures, combining various instances of the same architecture, e.g., a stacked autoencoder, and most cases are heterogeneous compound architectures, combining various types of architectures, e.g., an RNN Encoder-Decoder which combines an RNN and an autoencoder.

4.4 Challenge

A challenge is one of the qualities (requirements) that may be desired for music generation.

4.4.1 Ex Nihilo Generation

Suppose that our objective is to generate a melody on its own (not as an accompaniment of some input melody) while being based on a style learnt from a corpus of melodies.

4.4.1.1 Decoder Feedforward

The first strategy is based on an autoencoder architecture. Through the training phase an autoencoder will specialize its hidden layer into a detector of features characterizing the type of music learnt and its variations.

One can then use these features as an input interface to parameterize the generation of musical content. The idea is then to: choose a seed as a vector of values corresponding to the hidden layer units; insert it in the hidden layer; and feedforward it through the decoder.

This strategy, that we name decoder feedforward, will produce a new musical content corresponding to the features, in the same format as the training

examples. In order to have a minimal and high-level vector of features, a stacked autoencoder is often used.

The seed is then inserted at the bottleneck hidden layer of the stacked autoencoder and feed forwarded through the chain of decoders. Therefore, a simple seed information can generate an arbitrarily long, although fixed-length, musical content.

4.4.1.2 Sampling

Another strategy is based on sampling. Sampling is the action of generating an element (a sample) from a stochastic model according to a probability distribution.

The main issue for sampling is to ensure that the samples generated match a given distribution. The basic idea is to generate a sequence of sample values in such a way that, as more and more sample values are generated, the distribution of values more closely approximates the target distribution. Sample values are thus produced iteratively,

For musical content, we may consider two different levels of probability distribution (and sampling):

- Item level or vertical dimension – at the level of a compound musical item, e.g., a chord. In this case, the distribution is about the relations between the components of the chord, i.e. describing the probability of notes to occur together.
- Sequence-level or horizontal dimension – at the level of a sequence of items, e.g., a melody composed of successive notes. In this case, the distribution is about the sequence of notes, i.e. it describes the probability of the occurrence of a specific note after a given note.

With that in mind, an RBM (restricted Boltzmann machine) architecture is generally used to model the vertical dimension, i.e. which notes should be played together. And an RBM (restricted Boltzmann machine) architecture is generally used to model the vertical dimension, i.e. which notes should be played together.

4.4.2 Length Variability

An important limitation of the single-step feedforward strategy and of the decoder feedforward strategy is that the length of the music generated (more precisely the number of times steps or measures) is fixed. It is actually fixed by the architecture, namely the number of nodes of the output layer. To generate a

longer (or shorter) piece of music, one needs to reconfigure the architecture and its corresponding representation.

4.4.2.1 Iterative Feedforward

The standard solution to this limitation is to use a recurrent neural network (RNN). The typical usage is to:

- Select some seed information as the first item (e.g., the first note of a melody);
- Feedforward it into the recurrent network in order to produce the next item (e.g., next note);
- Use this next item as the next input to produce the next next item;
- And repeat this process iteratively until a sequence (e.g., of notes, i.e. a melody) of the desired length is produced.

4.4.3 Content variability

A limitation of the iterative feedforward strategy on an RNN is that generation is deterministic. Indeed, a neural network is deterministic. As a consequence, feed forwarding the same input will always produce the same output. As the generation of the next note, the next next note, etc., is deterministic, the same seed note will lead to the same generated series of notes. Moreover, as there are only 12 possible input values (the 12 pitch classes), there are only 12 possible melodies.

4.4.3.1 Sampling

Fortunately, the usual solution is quite simple. The assumption is that the output representation of the melody is one-hot encoded. In other words, the output representation is of a piano roll type, the output activation layer is softmax and generation is modeled as a classification task.

The default deterministic strategy consists in choosing the class (the note) with the highest probability. We can then easily switch to a nondeterministic strategy, by sampling the output which corresponds to a probability distribution between possible notes.

By sampling a note following the distribution generated, we introduce stochasticity in the process and thus variability in the generation.

4.4.4 Expressiveness

One limitation of most existing systems is that they consider fixed dynamics (amplitude) for all notes as well as an exact quantization (a fixed tempo), which makes the music generated too mechanical, without expressiveness or nuance.

A natural approach resides in considering representations recorded from real performances and not simply scores, and therefore with musically grounded (by skilled human musicians) variations of tempo and of dynamics.

Note that an alternative approach is to automatically augment the generated music information (e.g., a standard MIDI piece) with slight transformations on the amplitude and/or the tempo.

In the case of an audio representation, expressiveness is implicit to the representation. However, it is difficult to separately control the expressiveness (dynamics or tempo) of a single instrument or voice as the representation is global.

4.4.5 Melody-harmony Consistency

When the objective is to simultaneously generate a melody with an accompaniment, expressed through some harmony or counterpoint, an issue is the musical consistency between the melody and the harmony.

Although a general architecture, such as MiniBach, is supposed to have learnt correlations, interactions between vertical and horizontal dimensions are not explicitly considered.

4.4.6 Control

A deep architecture generates musical content matching the style learnt from the corpus. This capacity of induction from a corpus without any explicit modeling or programming is an important ability.

However, like a fast car that needs a good steering wheel, control is also needed as musicians usually want to adapt ideas and patterns borrowed from other contexts to their own objective and context, e.g., transposition to another key, minimizing the number of notes, finishing with a given note, etc.

4.4.6.1 Dimensions of Control Strategies

Arbitrary control is a difficult issue for deep learning architectures and techniques because neural networks have not been designed to be controlled. In

the case of Markov chains, they have an operational model on which one can attach constraints to control the generation.

However, neural networks do not offer such an operational entry point and the distributed nature of their representation does not provide a clear relation to the structure of the content generated.

Therefore, most strategies for controlling deep learning generation rely on external intervention at various entry points (hooks) and levels: input; output; input and output; encapsulation/reformulation.

4.4.6.1.1 Sampling

Sampling from a stochastic architecture (such as a restricted Boltzmann machine (RBM)), or from a deterministic architecture (in order to introduce variability), may be an entry point for control if we introduce constraints into the sampling process. This is called constrained sampling.

Constrained sampling is usually implemented by a generate-and-test approach, where valid solutions are picked from a set of random samples generated from the model. But this could be a very costly process and, moreover, with no guarantee of success. A key and difficult issue is therefore how to guide the sampling process in order to fulfill the constraints.

4.4.6.1.2 Conditioning

The idea of conditioning (sometimes also named conditional architecture) is to condition the architecture on some extra information, which could be arbitrary, e.g., a class label or data from other modalities.

In practice, the conditioning information is usually fed into the architecture as an additional input layer. This distinction between standard input and conditioning input follows a good architectural modularity principle. Conditioning is a way to have some degree of parametrized control over the generation process.

The conditioning layer could be a simple input layer. An example is a tag specifying a musical genre or an instrument in the WaveNet system. Or, some output of some architecture, being the same architecture, as a way to condition the architecture on some history, or even another architecture.

4.4.6.1.3 Input Manipulation

The idea is that the initial input content, or a brand new (randomly generated) input content, is incrementally manipulated in order to match a target

property. Note that control of the generation is indirect, as it is not applied to the output but to the input, before generation.

4.4.6.1.4 Input Manipulation and Sampling

Another option is the combination of the input manipulation strategy with the sampling strategy, thus acting both on the input and the output

4.4.6.1.5 Reinforcement

The idea of the reinforcement strategy is to reformulate the generation of musical content as a reinforcement learning problem: using the similarity to the output of a recurrent network trained on the dataset as a reward and adding user defined constraints, e.g., some tonality rules according to music theory, as an additional reward.

4.4.6.1.6 Unit Selection

The unit selection strategy is about querying successive musical units (e.g., one measures long melodies) from a database and concatenating them in order to generate a sequence according to some user characteristics.

Querying is using features which have been automatically extracted by an autoencoder. Concatenation, i.e. “what unit next?”, is controlled by two LSTMs, each one for a different criterium, in order to achieve a balance between direction and transition.

This strategy, as opposed to most of the other ones, which are bottom-up, is top-down, as it starts with a structure and fills it.

4.4.6.2 Style Transfer

The idea in this approach, named style transfer, designed for images, is to use a deep convolutional feedforward architecture to independently capture the features of a first image (named the content), and the style (as a correlation between features) of a second image (named the style).

Gradient-based learning is then used to guide the incremental modification of an initially random third image, with the double objective of matching both the content and the style descriptions.

The style transfer technique for images is effective and relatively straightforward to apply. However, as opposed to paintings, where the common representation is two-dimensional and uniformly digitalized in terms of pixels,

music is a much more complex object with various levels and models of representation.

More specific types of style transfers are: the Composition Style Transfer, working on symbolic representations; the Timbre Style Transfer, using various kinds of sources (various styles of music as well as speech); and the Performance Style Transfer.

4.4.7 Structure

One challenge is that most existing systems have a tendency to generate music with no clear structure or “sense of direction”. In other words, although the style of the generated music corresponds to the corpus learnt, the music appears to wander without any higher organization, as opposed to human composed music which has some global organization (usually named a form) and identified components, such as an overture, an allegro, an adagio or a finale in classical music; or an AABA or an AAB form in jazz; or even a refrain, a verse or a bridge in song music.

4.4.8 Originality

The issue of the originality of the music generated is not only an artistic issue (creativity) but also an economic one, because it raises the issue of copyright.

One approach is a posteriori, by ensuring that the generated music is not too similar[15] to an existing piece of music. Therefore, existing algorithms to detect similarities in texts may be used.

Another approach, more systematic but even more challenging, is a priori, by ensuring that the music generated will not recopy a given portion of music from the training corpus.

A solution for music generation from Markov chains has been proposed, it is based on a variable order Markov model and constraints over the order of the generation through some min order and max order constraints, in order to attain some sweet spot between junk and plagiarism. However, there is not yet a solution for deep learning architectures.

[15] e.g. in not having recopied a significant number of notes of a melody.

4.4.8.1 Conditioning

Two possible methods to control creativity are: restricting the conditioning by inserting the conditioning data only in the intermediate convolution layers of the generator architecture; and decreasing the values of the two control parameters of feature matching regularization, in order to reduce the requirement for the closeness of the distributions of real data and generated distributions of real and generated data.

4.4.8.2 Creative Adversarial Networks

Another more systematic and conceptual direction is the concept of creative adversarial networks (CAN), as an extension of the generative adversarial networks (GAN) architecture.

4.4.9 Incrementality

A straightforward use of deep architectures for generation leads to a one-shot generation of a musical content as a whole in the case of a feedforward or autoencoder network architecture, or to an iterative generation of time slices of a musical content in the case of a recurrent network architecture.

This is a strong limitation if we compare this to the way a human composer creates and generates music, in most cases very incrementally, through successive refinements of arbitrary parts.

4.4.10 Interactivity

An important issue is that, for most current systems, generation of musical content is an automated and autonomous process. Some interactivity with a human user(s) is fundamental to obtaining a companion system to help humans in their musical tasks (composition, counterpoint, harmonization, analysis, arranging, etc.) in an incremental and interactive manner.

4.4.11 Adaptability

One fundamental limitation of current deep learning architectures for the generation of musical content is that they paradoxically do not learn or adapt.

Learning is applied during the training phase of the network, but no learning or adaptation occurs during the generation phase. However, one can imagine some feedback from a user, e.g., the composer, producer, listener, about the quality and the adequacy of the generated music.

This feedback may be explicit, which puts a task on the user, but it could also be, at least partly, implicit and automated. For instance, the fact that the user quickly stops listening to the music just generated could be interpreted as negative feedback. On the contrary, the fact that the user selects a better rendering after a first quick listen to some initial reproduction could be interpreted as positive feedback.

Several approaches are possible. The most straightforward approach, considering the nature of neural networks and supervised learning, would be to add the newly generated musical piece to the training set and eventually retrain the network. This would reinforce the number of positive examples and gradually update the learnt model and, as a consequence, future generations.

However, there is no guarantee that the overall generation quality would improve. This could also lead the model to overfit and loose some generalization. Moreover, there is no direct possibility of negative feedback, as one cannot remove a badly generated example from the dataset because there is almost no chance that it was already present in the dataset. At the junction between adaptability and interactivity, an interesting approach is that of interactive machine learning for music generation

4.4.12 Explainability

A common critique of sub-symbolic approaches of Artificial Intelligence (AI), such as neural networks and deep learning, is their black box nature, which makes it difficult to explain and justify their decisions.

Explainability is indeed a real issue, as we would like to be able to understand and explain what (and how) a deep learning system has learned from a corpus as well as why it ends up generating a given musical content.

4.5 Strategy

The strategy represents the way the architecture will process representations in order to generate the objective while matching desired requirements.

Examples are single-step feedforward, iterative feedforward, decoder feedforward, sampling and input manipulation. As they've already been covered in previous sections, they won't be approached here.

5. Methodology

From the point of view of the game, we intend to investigate the use of music generation in different game genres. The greatest interest is in games with a deeper narrative such as: RPG; action-adventure games; and simulation games. Still, there is also the possibility to analyze casual games and music games[16].

The most important issue of associating interactive storytelling with musical narrative must start from the pioneering experiments of the ICAD/VisionLab[17] group with cinematography and music (Soares de Lima et al. 2005) (Soares de Lima et al. 2017). Scene events are created by the plot generator and the scene type is automatically calculated by the system based on parameters of the narrative. In a first approximation, for example, a fear scene, the soundtrack must have fast times, dissonances and small variations in pitch. Another possibility is to work directly with the parameters of the narrative, such as: genre of the story; dramatic arc of the plot; type of player; player behavior; and emotions, personality and relationships of the characters. Finally, the experiences of the ICAD/VisionLab group with personality models, behavior models and forecasting techniques in the generation of interactive storylines should be analyzed as mechanisms to support the generation of music for games (Soares de Lima et al. 2018a) (Soares de Lima et al. 2018b).

Currently, there are several ways to approach the issue of generating music in an adaptive way to a narrative, via a set of control parameters (as, for example, in the system of Prechtl et al. 2014. An even more ambitious goal is a combined generation of narrative and music. In this project, we chose deep learning as the basis of learning and generation. The first advantage is to take advantage of the recent innovations in generative architectures and the means to control them. The advantage of this approach over a formulation based on Markov models is twofold: more precise style learning (greater conformity) and learning less prone to plagiarism. A large-order Markov model also has good corpus compliance, but there is a tendency to recopy sequences from the corpus and thus generate plagiarism.

---

[16] i.e. games that challenge the player to follow sequences of movement or to develop specific rhythms.

[17] A research & development lab on visualization, digital television/cinema, and games at PUC-Rio in Brazil.

To map parameters from narrative to generation control, we propose the use of a conditional architecture, also called conditioning. In practice, this strategy (Briot et al. 2019) means adding a specific entry to the neural network to parameterize the generation. An example of conditioning is an architecture for generating melody conditioned by text (lyrics) (Yu et al. 2021). This architecture, called conditional LSTM-GAN, combines a GAN[18] architecture, a recurring network architecture, in practice a LSTM[19], and a text conditioning.

This type of architecture is an interesting starting point for the project. The advantage of using a recurring network is that the generation is iterative[20] and, thus, can adapt dynamically to the dynamics of the game's narrative. The granularity of the generation (note, measure...) can be configured according to the type of narrative, depending on the minimal granularity of the interactive narrative. The melody (or melodies, or the melody and the chord) can be generated in real time and transformed into music audio from real-time sound synthesis, also controllable. An alternative architecture to GAN is a VRAE[21] (Fabius and R. van Amersfoort 2014), which offers additional control possibilities by manipulating the values of the latent variables of the autoencoder[22].

[18] Generative Adversarial Networks.
[19] Long Short-Term Memory.
[20] Note by note or measure after measure.
[21] Variational Recurrent Autoencoder
[22] They constitute a compressed and abstract representation - variational - of the examples of songs learned, the variations corresponding to the dimensions of variation among the examples, such as the texture, the duration of the notes, etc.

6. Performed Activities

After preliminary studies, it was decided that we should analyze tools that would serve as a basis for the development of the idealized system. In this context, it was concluded that the Magenta Project would be the most suitable framework for the situation due to several factors, among them, being open source and an extremely powerful tool for music generation.

6.1 The Magenta Framework

Magenta is built on top of Tensorflow, and like it, it has a great Python dominance regarding its use. However, for a first contact I chose to use Magenta.js, a version of the framework used to create music directly from the browsers[23], due to my greater ease with web development in relation to game development. Moreover, the Javascript version offered a higher delivery speed focused on materializing theoretical knowledge about music generation and the framework itself.

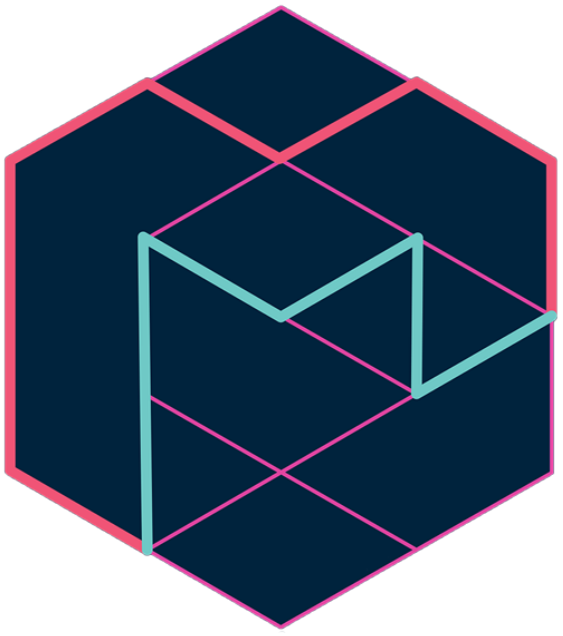

Figure 1. Magenta Project logo[24]

[23] Although it can also be used with Node.js.
[24] Source: https://magenta.tensorflow.org/assets/magenta-logo-bottom-text.png

6.2 First Efforts

So, inspired by the musical ambiance of sites like the one of the game Illuvium[25] I started the project implementation aiming to deliver a new experience for users in the web environment.

On a website, as in games and movies, the user has a journey to go, regardless of the type of conversion desired. However, unlike the film or game industry, the music resource is rarely used in the web context. Thus, the idea was, as in a game, to use automatic music generation, with Magenta.js, to produce musical content for the site, according to the user's usage.

For that, I set up a small MVP[26], hosted locally, where according to the mouse's scroll position on the site, parameters of the music generation would be changed, providing a unique and immersive experience for the user.

Given time limitations, only a single parameter was varied using several available models: the temperature of the model, that is, the randomness rate of the predictions. This implementation was made so that the temperature varied according to the position of the user's mouse scroll, thus, when entering the site, on a screen mostly composed of images, the music is more aggressive, however, when navigating to the lower sections of the site, composed mostly of text, it calms down and more closely follows the style of the samples that the model was trained on.

In this first moment, we used many of the main models available in Magenta for Javascript, such as MusicRNN[27] and MusicVAE[28], all of them already trained with pre-trained hosted checkpoints available in the Magenta documentation[29]. However, it is noteworthy mentioning the delay in loading the models in the browser, which can be observed as a possible problem for future implementations of the proposal in question.

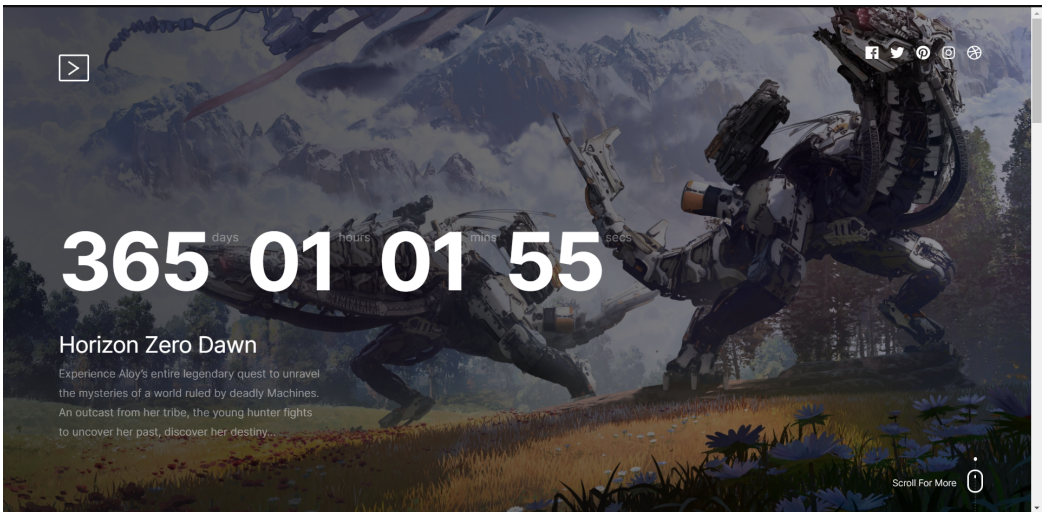

Figure 2. The home view of the developed MVP site

[25] Found at: https://illuvium.io/
[26] Minimum Viable Product
[27] Found at: https://www.npmjs.com/package/@magenta/music#musicrnn
[28] Found at: https://www.npmjs.com/package/@magenta/music#musicvae
[29] Found at: https://www.npmjs.com/package/@magenta/music#model-checkpoints

6.3 The System Development

Afterwards, with a better definition of the necessary requirements for the automatic music generation system for games, I was able to use the knowledge obtained in the experience above, to effectively start the implementation in the context of games.

To this end, the maximum possible abstraction of the context of the game itself was admitted, since all elements, except the musical creation, will be covered in another project in the future.

So, I thought of an implementation that followed this paradigm and was as far as possible decoupled from the creation of the game itself. That said, inspired by the Docker architecture[30] of server and client, I decided to try to implement a similar one: a server responsible for music creation, and several clients, responsible for using the generated music in the most diverse environments, being game engines like Unity and Unreal, or even games created directly in the browser.

Thus, in addition to keeping the project agnostic about the use of generated music, the project's scalability is also greatly increased, centralizing all the logic and computational cost of AI music creation in specialized machines and environments.

In this project, however, the idea is to focus only on server development, that is, on the challenge of the musical generation. For that, I decided to break this challenge into smaller and more efficient problems. That is, instead of training a single model capable of producing all types of musical content, I planned the implementation of several specialized models, responsible for generating specific content for each musical type, all of which should be controlled by a large capable state machine to choose which network will be active in a given request, according to the parameters provided.

In continuity, to generalize the environment of the games, and encompass a wide range of possible types of games, the first network I chose to explore was the one responsible for generating the ambient music of the game. So I started my research on efficient models for this particular task where I found the Music Transformer (Anna Huang et al. 2018), an attention-based neural network that can generate music with improved long-term coherence.

---

[30] Explained at: https://docs.docker.com/get-started/overview/#docker-architecture

To build the server, Typescript was used with Nest.js, an architectural framework built on top of Node.js. Unfortunately, Majenta.js still doesn't support Music Transformer, so it was necessary to migrate to the main version of Magenta in Python. At first, to reduce setup time, we used a container already made and available by Dan Jeffries, Chief Technology Evangelist at Pachyderm, in a work in partnership with Project Magenta (Jeffries 2020). At this first moment there was no need to run away from this container as no architectural changes to the network were required, however, in a future moment this was necessary.

The model itself already had a pre-trained checkpoint, where it was trained with ambient music. Since it already provides incredible results, the checkpoint was used in the server implementation, although, if new training would be necessary, there was already an entire pipeline available with Pachyderm[31], a platform that enables you to run your data science experiments in a reliable and reproducible way linking together a bunch of loosely coupled frameworks into a smoothly scaling AI generating machine. This approach has several advantages, but a particularly attractive one, given the short time and limited resources of this project, is the easy use of computing resources in the cloud to carry out the training in the desired time. Thus, this same pipeline was used to train any other musical styles needed in the rest of the implementation.

Therefore, to use this same configuration in the Nest.js environment, I developed an endpoint on the server so that every request made on it, a new container is executed, responsible for generating a new song based on the parameters received, at first, only the duration of the song. To take full advantage of the great musical coherence of Music Transformer, durations longer than 2 minutes have always been used, guaranteeing better results.

```
[Nest] 19960  - 26/11/2021 23:41:13     LOG [NestFactory] Starting Nest application...
[Nest] 19960  - 26/11/2021 23:41:13     LOG [InstanceLoader] DiscoveryModule dependencies initialized +33ms
[Nest] 19960  - 26/11/2021 23:41:13     LOG [InstanceLoader] AppModule dependencies initialized +1ms
[Nest] 19960  - 26/11/2021 23:41:13     LOG [InstanceLoader] MusicGenModule dependencies initialized +0ms
[Nest] 19960  - 26/11/2021 23:41:13     LOG [InstanceLoader] ScheduleModule dependencies initialized +1ms
[Nest] 19960  - 26/11/2021 23:41:13     LOG [RoutesResolver] AppController {/}: +206ms
[Nest] 19960  - 26/11/2021 23:41:13     LOG [RouterExplorer] Mapped {/, GET} route +3ms
[Nest] 19960  - 26/11/2021 23:41:13     LOG [RoutesResolver] MusicGenController {/music-gen}: +0ms
[Nest] 19960  - 26/11/2021 23:41:13     LOG [RouterExplorer] Mapped {/music-gen, POST} route +1ms
[Nest] 19960  - 26/11/2021 23:41:13     LOG [NestApplication] Nest application successfully started +12ms
-       AI Music Generator 🎵
-       Resource Unlocked 🔓
-       Generating Your Musics...
-       Done ✔
-       Check Your Musics At /assets/outputs
-       Total Time Taken: 277s

[Nest] 19960  - 26/11/2021 23:46:16     LOG [HTTP] POST /music-gen 201 1681 - PostmanRuntime/7.26.8 ::1
```

Figure 3. The view from the console when the server receives a request and generates it's output

[31] Pachyderm website: https://www.pachyderm.com/

After testing the music generation a few times and verifying that everything was working correctly, two new challenges were addressed, one, the time needed to generate a new song, and the other, the system output through MIDI files.

To solve the first problem, two options were evaluated, one the use of high-performance cloud computing, which ended up being discarded, at least for the time being, due to the lack of time to explore and evaluate the costs and qualities of possible existing services, and the second, which was implemented, using CUDA[32], the parallel computing platform and programming model developed by NVIDIA for general computing on GPUs[33]. With this decision, the time taken to generate one music was reduced by approximately 9 times, which, even though it can still be optimized, has already solved the problem in that first moment.

One of the problems to do this in the local environment, was the fact that the containers in the local machine were runned with Docker Desktop, from within the WSL, which, with it's current windows 10 version, didn’t have the support for GPU usage. So, to solve that, the container option had to be discarded and a submodule of the project, in python, was created. This submodule was implemented with the exactly same environment of the previous container but now, running directly from the local machine and, therefore, with access to the GPUs.

For the second challenge, the effectiveness of the separation of the system into client and server was further corroborated. And this because, as it was possible to consider the client as a black box, having its implementation free for the environment that best suits it, the MIDI format became perfect.

Unlike regular audio files like MP3s or WAVs, MIDI files don't contain actual audio data and are therefore much smaller in size. They instead explain what notes are played, when they're played, and how long or loud each note should be. Files in this format are basically instructions that explain how the sound should be produced once attached to a playback device or loaded into a particular software program that knows how to interpret the data.

This makes MIDI files perfect for sharing musical information between similar applications and for transferring over low-bandwidth internet connections. The small size also allows for storing on small devices like floppy disks, a

[32] Compute Unified Device Architecture
[33] Graphical Processing Units

common practice in early PC games. Furthermore, they allow even more customization of the client according to their needs, without having to change the flow of music generation in any way.

Thus, to check the quality of the system outputs and simulate a client using the server, a music creation software, Ableton Live[34], was used. This choice was made mainly by the quality and quantity of instruments provided by this tool, including great instruments for ambient music, such as Seashore Pad and Pitched Ambience.

6.4 The Musical Creation

After that, I started to effectively experience the musical creation itself. However, when using only one instrument to perform the generated songs, the result was poor, without the various complements that are present in a good musical composition. So, I started to generate songs from different samples and put them to play simultaneously with different instruments. The observed result is that one can easily change the final result of the composition by just turning on, turning off or changing any of the songs being played at that moment.

Therefore, when analyzing the performance of different results of musical generation, at the same time, and with different instrumentation, the idea of composing the system not only by state machines with specific musical styles, but also by layers that could dynamically change the music generation as needed came up.

With this idea in mind, 4 layers were defined to be implemented[35] with the first one being the most constant and the last one the most aggressive, and the intermediate layers being of gradual transition between the former ones. So, if the game was in a calm environment, only the first layers would be active, or if it was following a dangerous situation, only the last ones would be active, and so on.

[34] Ableton Live website: https://www.ableton.com/
[35] Although this value could be easily changed if needed

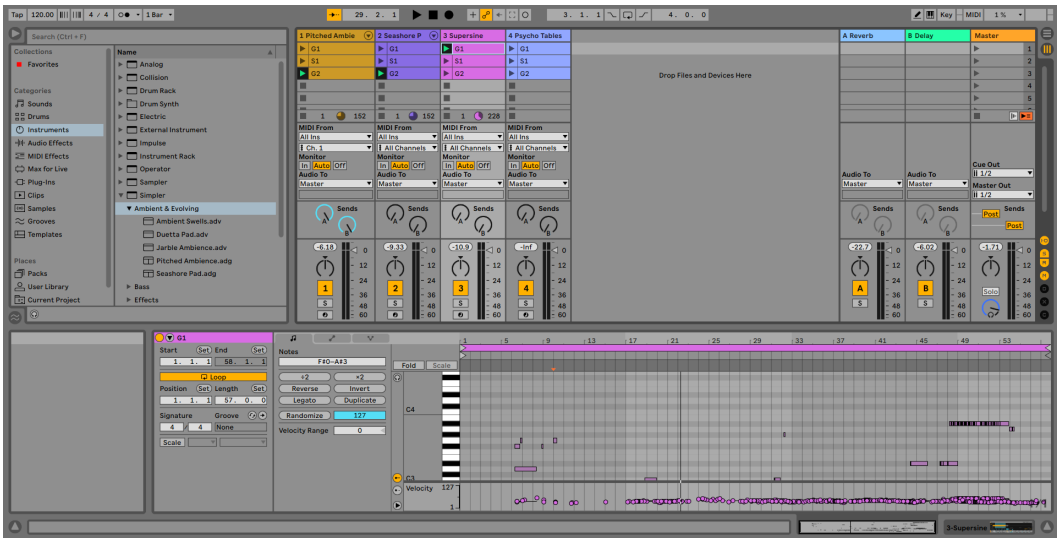

Figure 4. The visualization of the four established layers at the Ableton Live's software

6.5 First Results

In conclusion, it was possible to generate, in a very short space of time, very interesting results, proving the initial premise that this model and this architecture would work very well in the context of the proposed music creation.

7. The System

The server side of the system was built in Typescript with NestJS. It handles all backend requests and logic like data storage and schedules. For music generation, a submodule built in Python with Magenta / Tensorflow was used. With this configuration, the main idea is that the request is handled by the server and then, if necessary, the sub-module is activated to generate the music.

7.1 The Proposed System

The first proposal to the system came up as the following scheme:

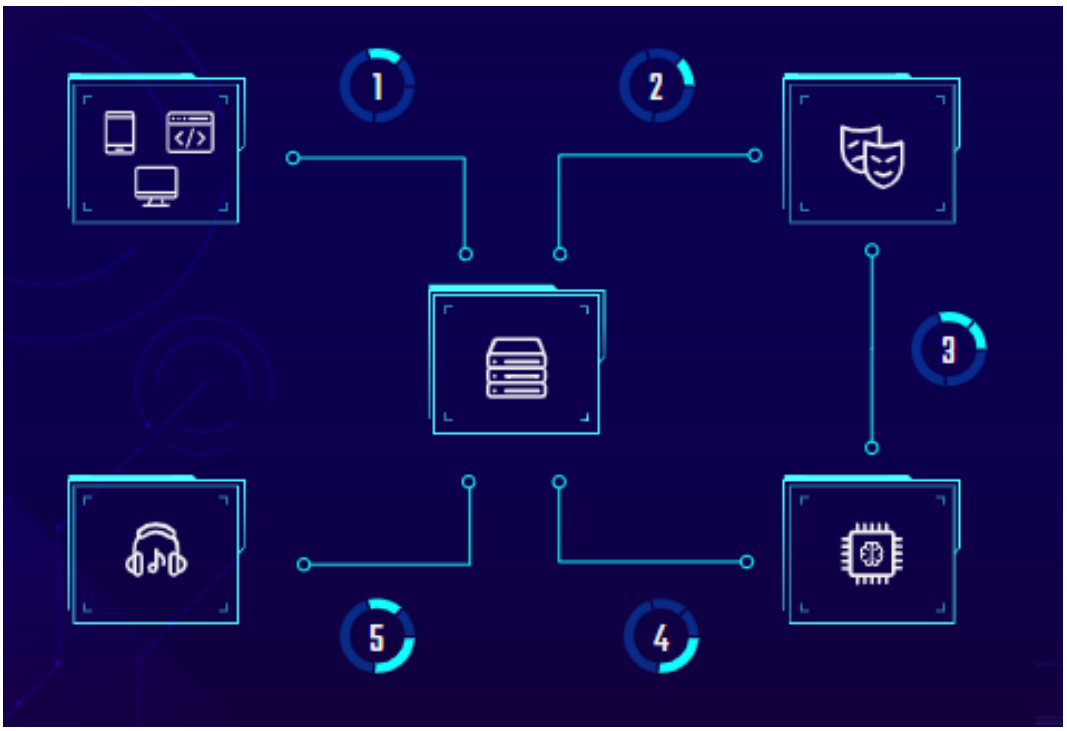

Figure 5. The schema of the proposed system

1. User's client requests a music.
2. The server maps the user feeling through the provided parameters.
3. It uses the mapped feeling to start the music generation.
4. After the music is generated, it attaches metadata such as instruments
5. It delivers the request response with the music to the final user.

7.2 Mapping Emotions

With the base structure of the system defined, as approached in the last section, the need to evaluate parameters capable of integrating the game's plot and it's emotions with the music generation, arised.

To address that, some options were studied, like the use of game analytics to map the player's and game's emotion. But, for this first approach, we wanted something simple, easily replicable to a lot of different projects, and mostly agnostic from the game itself.

So, it was decided to ignore the huge amount of data that could be used in the map of emotions, and focus only in just two dimensions: arousal and valence.

7.3 The Arousal Valence Model

In this model, an emotion can be mapped using two parameters, the intensity of the feeling, the arousal, and the kind of feeling (e.g. if it's positive, negative, neutral etc.), the valence.

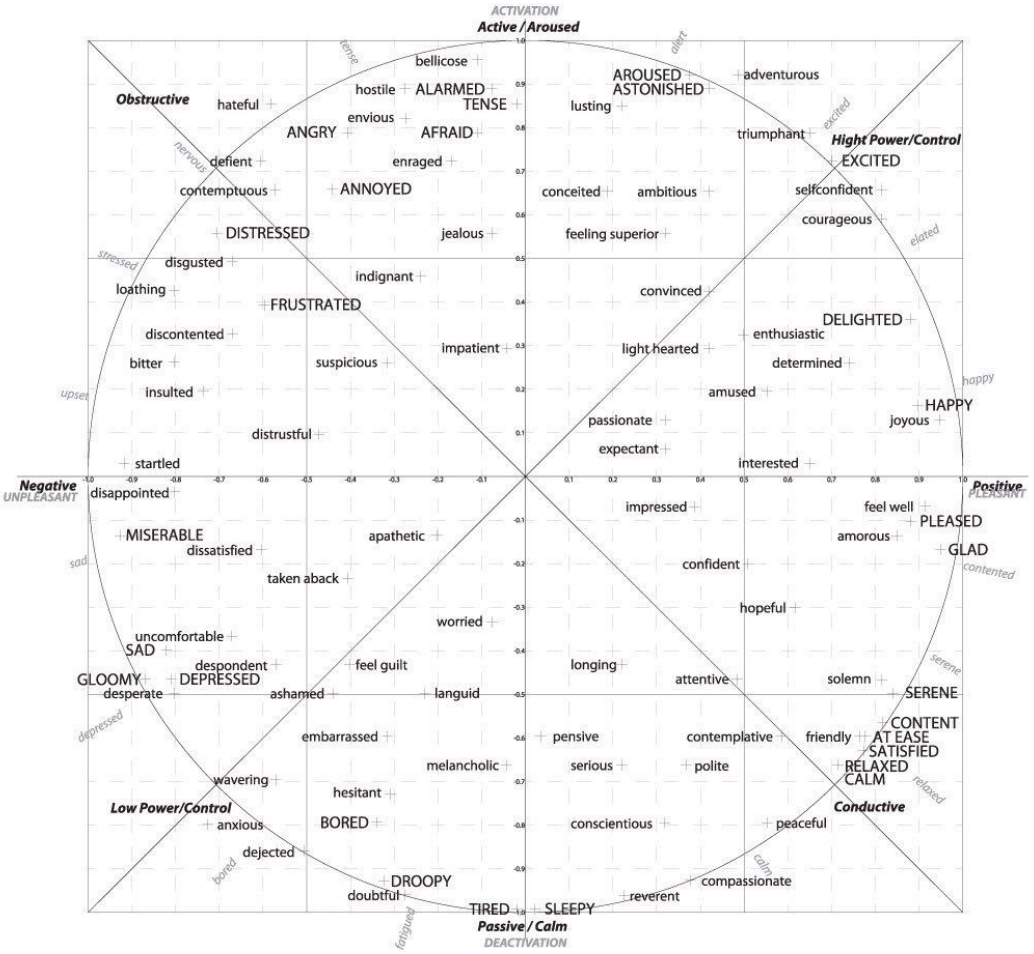

Figure 6. The arousal valence model[36]

For this first system, there was no need to use many feelings, since we

[36] Source: https://i.pinimg.com/originals/68/71/3a/68713ad7906a141a51ab1f604ae70137.jpg

could easily decrease the complexity by establishing just a few and projecting an implementation able to scale up for the future.

That said, to this project were used nine feelings, shown by the grid below:

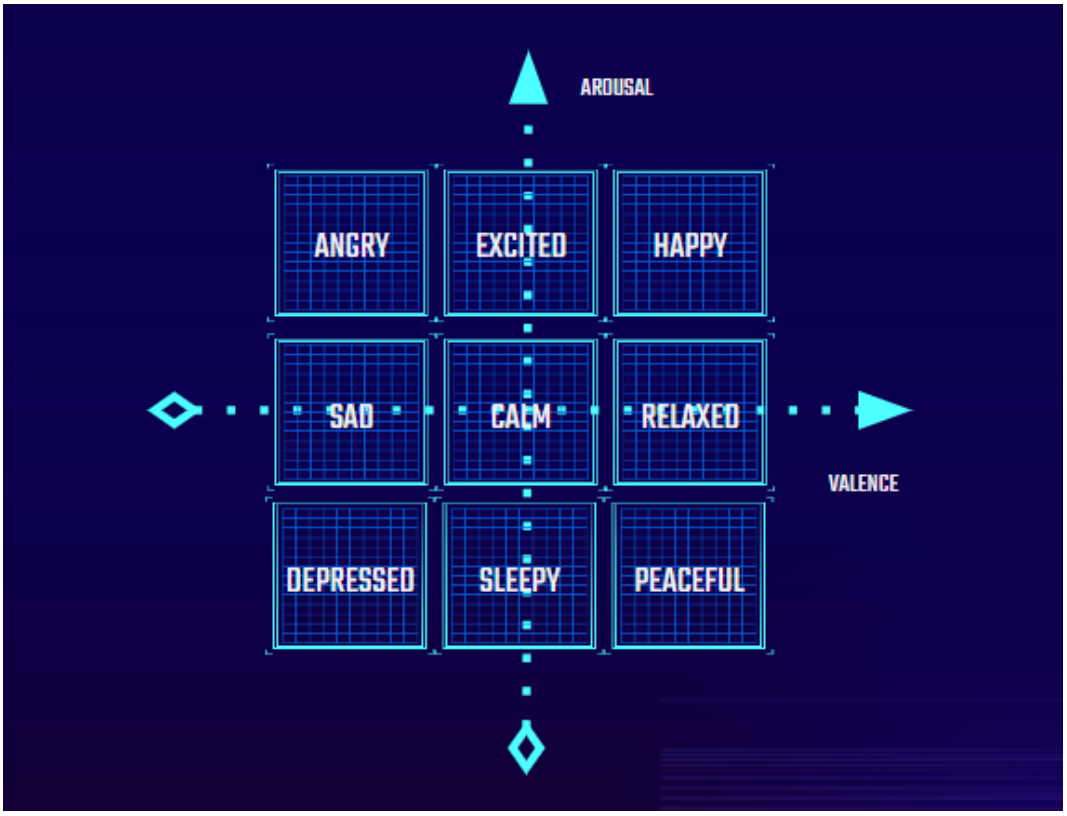

Figure 7. The developed arousal valence scheme, using a map with nine feelings

7.4 Generating The Player's Music

To the generation of the music itself, four models were evaluated: MusicVAE[37], MusicRNN[38], GANSynth[39] and MusicTransformer. As explained in Huang et al. 2018, because of its deeper long-term structure the MusicTransformer was the chosen one.

7.4.1 MusicTransformer

Briefly, the transformer is an attention-based neural network that can generate music with improved long-term coherence.

---

[37] MusicVAE is a variational autoencoder made up of an Encoder and Decoder, along with a data converter for converting between Tensor and note sequence objects for input and output.
[38] A MusicRNN is an LSTM-based language model for musical notes.
[39] GANSynth is a method for generating high-fidelity audio with Generative Adversarial Networks (GANs).

7.4.2 Attention

As excellently elucidated by Chris Nicholson in his introduction to attention (Nicholson), attention takes two sentences, turns them into a matrix where the words of one sentence form the columns, and the words of another sentence form the rows, and then it makes matches, identifying relevant context.

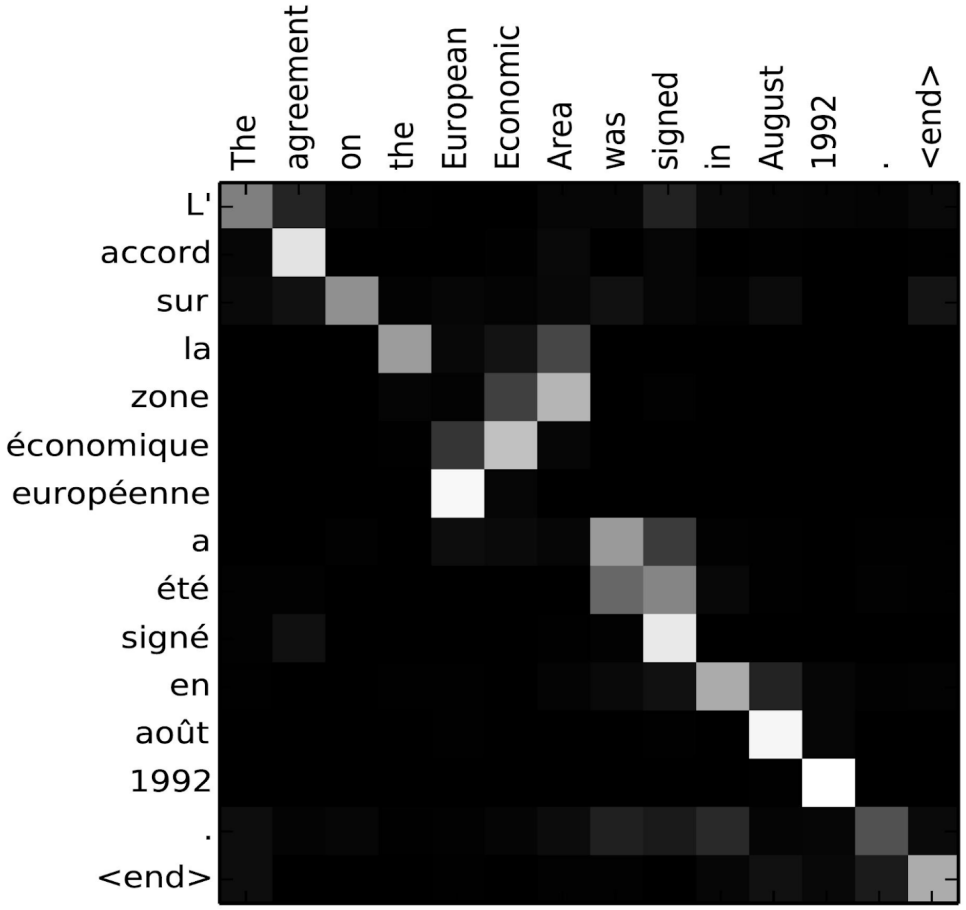

Figure 8. Attention example[40]

7.4.3 Self Attention

But the amazing thing about attention is that you don't need to lay out two different sentences, you can lay out the same sentence, to learn about it's important words, rather than the difference between two translations, this is called "self attention" as explained in Jeffries 2020.

[40] Source: https://wiki.pathmind.com/images/wiki/attention_translation_grid.png

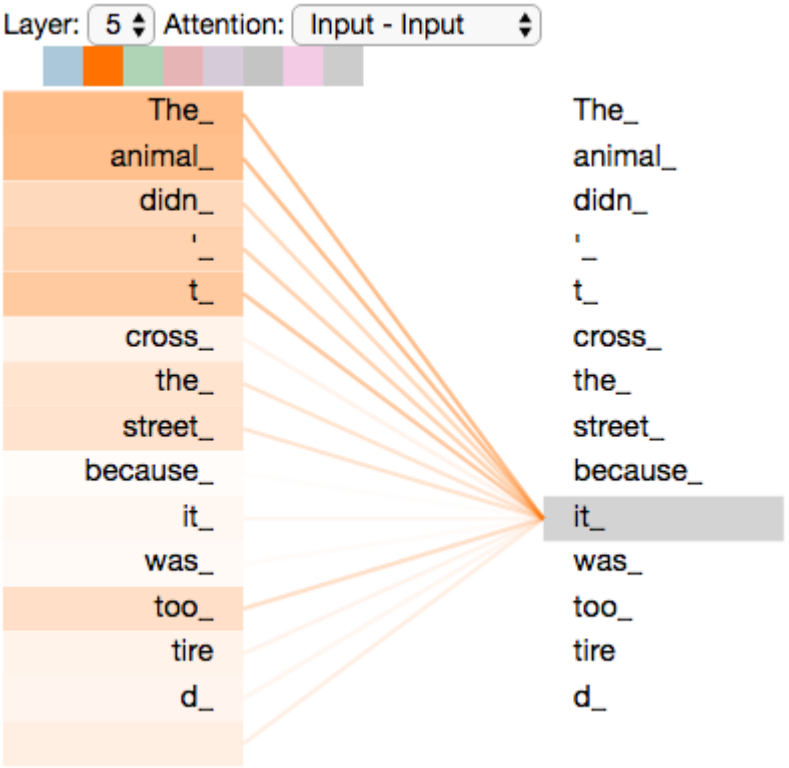

Figure 9. Self attention example[41]

7.4.4 Performance

In short, with self attention, Transformers can encode each word with how important it is to every other word. That means they don't just know about the few words that came before it, like RNNs, while knowing nothing about the words to come. Transformers know about every word in the sentence, forwards and backwards. That means they excel at both small and larger clusters of information and how they relate to each other. Thus, they are much better at developing a long term memory about what it's learned.

7.4.5 Relative Attention

With that in mind, it was developed the Music Transformer, a transformer with relative attention, a small modification in the original model, but which was responsible for enabling its use in the musical context, as shown in Huang et al. 2018.

While the original Transformer captures self-reference through attention, it relies on absolute timing signals and thus has a hard time keeping track of regularity that is based on relative distances, event orderings, and periodicity. Using relative attention, which explicitly modulates attention based on how far apart two tokens are, the model is able to focus more on relational features.

---

[41] Source: http://jalammar.github.io/images/t/transformer_self-attention_visualization.png

Relative self-attention also allows the model to generalize beyond the length of the training examples, which is not possible with the original Transformer model.

7.4.6 The Strategy System

One of the challenges encountered in developing the project was to efficiently implement a system capable of coherently generating different musical genres and types.

For this, as explained before, a state machine pattern was devised, but as with this first system transitions have not yet been explored, a strategy pattern was implemented. In it, each strategy has a specialized model, already trained, to generate a specific type of music. Three models were used, one trained with happy music, one with neutral music and the other with sad music.

src > music-gen > music-generator > strategies > TS angry.strategy.ts > ...

```
import { INSTRUMENTS, MODELS } from '../../../common/constants';
import { IFeelingMap } from '..';
import {
  ActiveLayer,
  InactiveLayer,
  ILayer,
  IStrategy,
  BaseStrategy,
} from './strategy.interface';

/**
 * Concrete Strategies implement the algorithm while following the base Strategy
 * interface. The interface makes them interchangeable in the Context.
 */
export class AngryStrategy extends BaseStrategy implements IStrategy {
  readonly model = MODELS.SAD;

  get layers(): ILayer[] {
    return [
      new ActiveLayer(INSTRUMENTS.SUNRISE_WAVES), // break line
      new ActiveLayer(INSTRUMENTS.DARK_ONE),
      new ActiveLayer(INSTRUMENTS.AIR_OF_DREAD),
    ];
  }

  isInRange({ arousal, valence }: IFeelingMap) {
    return (
      arousal >= 75 && // break line
      arousal <= 100 &&
      valence >= 0 &&
      valence <= 50
    );
  }
}
```

Figure 10. Code of the angry strategy, an example of implementation to illustrate the content of a strategy

With this implementation, the system is able to support as many strategies as needed, simply implementing the *isInRange* method to determine whether a strategy is active or not, in addition to having to choose the model it

will use and the active layers for this strategy, a topic that will be covered in the next section.

7.4.7 The Layering System

As already shown before, a layering system had been proposed, but now, it was integrated with the arousal valence model, meaning that the arousal would now be directly connected to the layering system, resulting in the final scheme integrating the strategies, the layers and the arousal valence model.

Active Layers / Music Generation Model

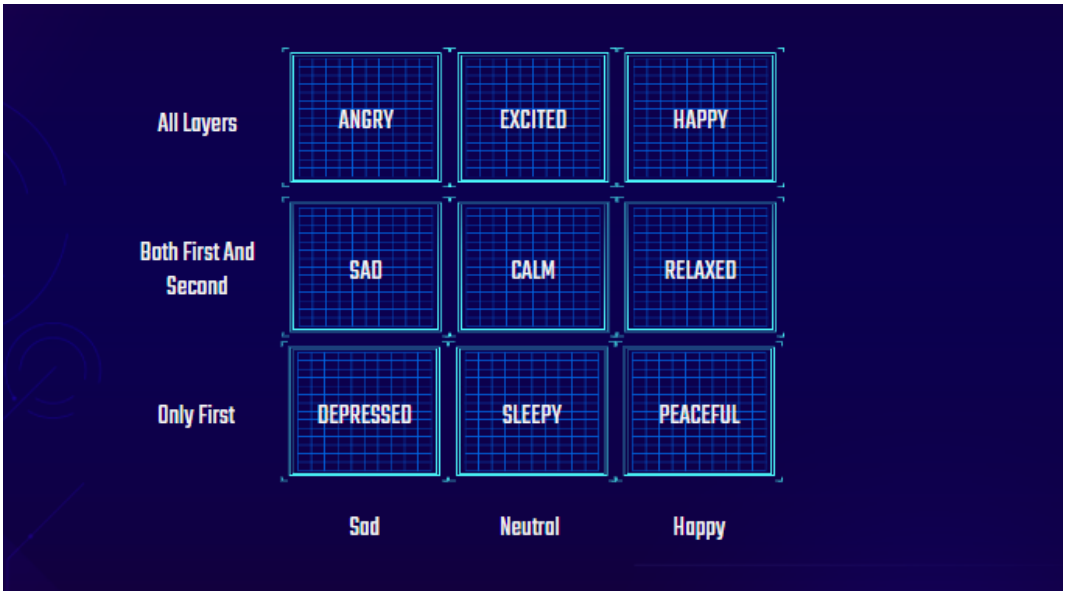

Figure 11. The final scheme, integrating strategies, layers and the arousal valence model

With this, the system is able to change the music output just by changing a layer, or its instrument, or even its duration. To further optimize the song generation process, the system has always saved at least one song of each strategy in memory, generating only a new one, if the current one has already been heard.

7.5 The Implemented System

After these considerations, the implemented system ended up a little different from the proposed one, embracing all the changes mentioned above.

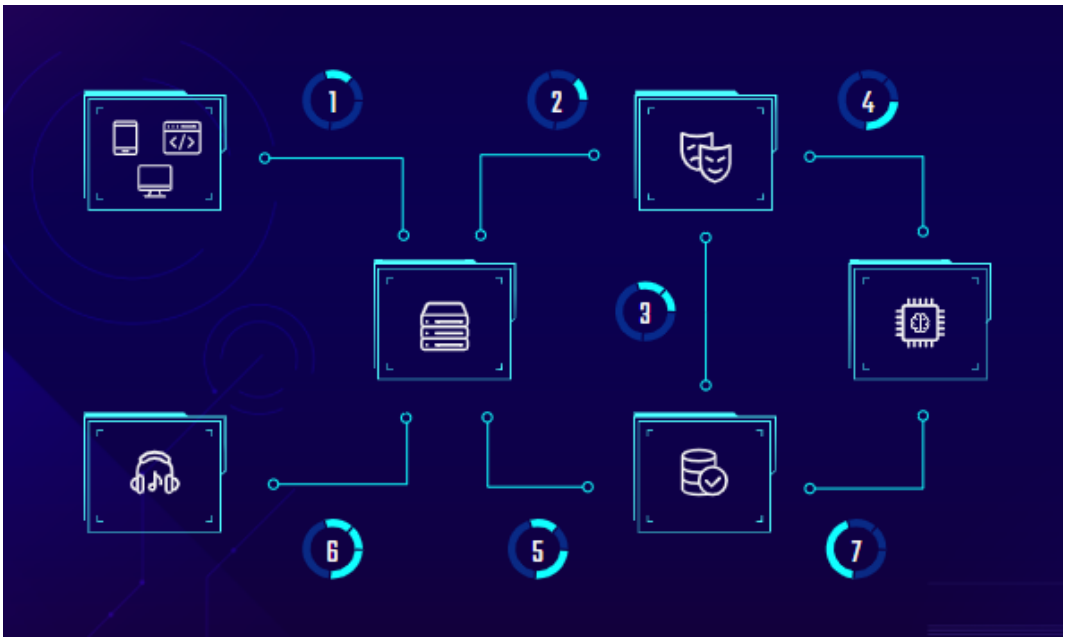

Figure 12. The implemented system scheme

1. User's client requests a music.
2. The server maps the user feeling through the arousal valence parameters.
3. It fetches, from memory, a song correspondent to the mapped feeling.
4. If no music is being generated, with the music transformer, it starts the generation of the most used layers.
5. After the music is fetched, it attaches metadata, such as layers configuration.
6. Delivers the request response with the music to the final user.
7. With the generated music refreshes the memory.

8. Final Considerations

Thus, in this project, we were able to start a research on music generation with deep learning for games, using very interesting techniques such as the arousal valence model, for emotion mapping, and the layer system, to optimize the process of generating music.

Furthermore, this study also led to the implementation of a proof-of-concept system, using a very interesting and promising model, which is the Music Transformer.

Finally, in a very short time, we got a working system capable of generating unique and controlled music for games, including some very good samples, using parameters to integrate it into the game's storyline, which was the main goal of the project.

8.1 Future Works

As the project implementation time was short, here we address some possible themes that would integrate very well into the current research and system presented in this report, but which were not addressed here due to lack of time.

8.1.1 Improved System Input

For this first approach, only the arousal valence model was used to map the player's emotions. But as explained before, there is a lot of data that could be used to improve this process, for instance, all the data retrieved from a game analytics system, e.g., age, gender, in-game events, etc.

Another option would be to increase the number of dimensions used in system input. At the moment, only arousal and valence are being considered, resulting in a 2D mapping, but it would be very interesting to have more dimensions, making it possible to map 3D structures or even nD spaces with many dimensions.

Thus, a very interesting work that could be studied in the future would be the improvement of input systems, either by integration with game analytics data or by increasing dimensions in the emotion mapping model.

8.1.2 Use Of Image Recognition

Another really nice option for optimizing the feeling map could be the use of image recognition to predict players' emotions, or at least assist in the mapping process.

8.1.3 Multiple Input Sources

At this time the system only receives one source of arousal and valence parameters, but there may be cases where several sources can be much more efficient, for example you may have many parameters coming from the game

map, player, plot etc. that you could compute and use to improve the system's emotion mapping.

Another thing to assess in this approach is the possibility of negative sources, for example, there may be cases where you don't want to intensify the feeling the player is having, that is, if the player is sad, you may want him to have an happier experience, or at least a less sad one, then you could simply use a negative weight for the source of players' emotions.

8.1.4 Multiple Levels State Machine

As proposed before, a good feature would be the implementation of a state machine pattern, tracking the transitions of the player's emotions. With this, the system would be able to use the music transformer to interpolate between the currently playing music and the next one, generating transitions for the generated music.

Even more, with the implemented state machine, you can implement multi-level states, adding new abstraction levels to the models, for example, you could have a macro level for the game theme, a medium level for the music genre and a micro level for a more granular analysis of the player's own feeling.

8.1.5 Scale Up The System

Another important thing to be studied, would be ways to scale up the system. The idea of this topic is to attempt to build an architecture capable of generating music for thousands of players at same time, for a MMORPG as an example.

In that context the server would have to be deployed in a cloud environment, but it would need a deep analysis of what technology should be used to do so, since there are a lot of infrastructure technologies such as dockerized options as AWS ECS[42], FaaS[43] options as Google Cloud Functions[44] etc.

Another important thing to be studied would be ways to scale the system. The idea of this topic is to try to build an architecture capable of generating music for thousands of players at the same time, for an MMORPG for example.

In this context, the server would have to be deployed in a cloud environment, but an in-depth analysis of what technology should be used, as there are many infrastructure technologies such as dockerized options like AWS ECS, FaaS options like Google Cloud Functions etc. that could be used in this process.

---

[42] Amazon Elastic Container Service - Is a fully managed container orchestration service that makes it easy for you to deploy, manage, and scale containerized applications. Found at: https://aws.amazon.com/ecs/?nc1=h_ls

[43] Function as a Service - Is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.
Source: https://en.wikipedia.org/wiki/Function_as_a_service

[44] Are scalable pay-as-you-go functions as a service (FaaS) to run your code with zero server management. Found at: https://cloud.google.com/functions

8.1.6 Analyze The MusicTransformer

Perhaps the most interesting option of all would be the attempt to improve the musical transformer model. One possibility is to analyze other variations of the vanilla transformer model, such as the FastTransformer (Vyas et al. 2020) and try to apply their modifications to the MusicTransformer. Another possibility would be the analysis of other components in the transformer, such as analyzing the control of the music generation process by changing the number of latent variables and dimensions in its encoders and decoders (Briot 2020).

9. References

Alammar, Jay. “The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.” *Jay Alammar Github*, 27 June 2018, http://jalammar.github.io/illustrated-transformer/. Accessed 2021.

Alammar, Jay. “Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention).” *Jay Alammar Github*, 9 May 2018, https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/. Accessed 2021.

Anna Huang, Cheng-Zhi, et al. “Music Transformer.” 2018. *arXiv:1809.04281*, https://arxiv.org/abs/1809.04281.

Briot, Jean-Pierre. “Compress to Create.” *MusMat - Brazilian Journal of Music and Mathematics*, 2020. *hal.sorbonne-universite*, https://hal.sorbonne-universite.fr/hal-02567390v3. Accessed 2021.

Briot, Jean-Pierre. *Deep Learning for Music Generation Course*. 2019, UNIRIO, Rio de Janeiro, Rio de Janeiro, Brazil.

Briot, Jean-Pierre, et al. “Deep Learning Techniques for Music Generation -- A Survey.” 2019. *arXiv:1709.01620*, https://arxiv.org/abs/1709.01620.

Briot, Jean-Pierre, and François Pachet. “Music Generation by Deep Learning - Challenges and Directions.” *Special Issue on Deep learning for music and audio, Neural Computing & Applications, Springer Nature*, 2018. *arXiv:1712.04371*, https://arxiv.org/abs/1712.04371.

Cabral, Giordano, et al. “Automatic x traditional descriptor extraction: The case of chord recognition.” *ISMIR 2005 - 6th International Conference on Music Information Retrieval*, 2005, pp. 444-449. *hal-01416436, version 1*, https://hal.archives-ouvertes.fr/hal-01416436.

Clarke, Arthur Charles. *Profiles of the Future (Second Edition)*. Pan Books, 1973.

Doersch, Carl. “Tutorial on Variational Autoencoders.” 2016. *arXiv:1606.05908*, https://arxiv.org/abs/1606.05908.

Engels, Steve, et al. “Automatic Real-Time Music Generation for Games.” *In Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 11, 2015, pp. 220–222. *AAAI*, https://ojs.aaai.org/index.php/AIIDE/article/view/12775.

Fabius, Otto, and Joost R. van Amersfoort. “Variational Recurrent Auto-Encoders.” 2014. *arXiv:1412.6581*, https://arxiv.org/abs/1412.6581.

Hadjeres, Gaëtan, et al. “DeepBach: a Steerable Model for Bach Chorales Generation.” *Proceedings of the 34th International Conference on Machine Learning, PMLR 70:1362-1371, 2017*, 2017. *arXiv:1612.01010*, https://arxiv.org/abs/1612.01010.

Huang, Anna, et al. “Music Transformer: Generating Music with Long-Term Structure.” *Magenta - Tensorflow.org*, 13 December 2018, https://magenta.tensorflow.org/music-transformer. Accessed 2021.

Jeffries, Daniel. “The Musician in the Machine.” *Magenta - Tensorflow.org*, 7 August 2020, https://magenta.tensorflow.org/musician-in-the-machine. Accessed 2021.

J. Goodfellow, Ian, et al. “Generative Adversarial Networks.” 2014. *arXiv:1406.2661*, https://arxiv.org/abs/1406.2661.

Nicholson, Chris. “A Beginner's Guide to Attention Mechanisms and Memory Networks.” *The Artificial Intelligence Wiki | Pathmind*, https://wiki.pathmind.com/attention-mechanism-memory-network. Accessed 2021.

Pachet, François. *Informatique musicale: du signal au signe musical*. Edited by François Pachet and Jean-Pierre Briot, Hermès Science Publ., 2004.

Prechtl, Anthony, et al. “Algorithmic Music As Intelligent Game Music.” *AISB 2014 - 50th Annual Convention of the AISB*, 2014.

https://www.researchgate.net/publication/287895305_Algorithmic_music_as_intelligent_game_music/citation/download, https://www.researchgate.net/publication/287895305_Algorithmic_music_as_intelligent_game_music.

Soares de Lima, Edirlei, et al. “Personality and Preference Modeling for Adaptive Storytelling.” *Brazilian Symposium on Computer Games and Digital Entertainment*, 2018a, pp. 187-197. *Portal SBGames*, http://www.sbgames.org/sbgames2018/files/papers/ComputacaoFull/185143.pdf.

Soares de Lima, Edirlei, et al. “Player Behavior Modeling for Interactive Storytelling in Games.” *Entertainment Computing*, vol. 28, 2018b, pp. 32-48. *ICAD PUC-Rio*, http://www.icad.puc-rio.br/~logtell/papers/Edirlei_SBGames_2016.pdf.

Soares de Lima, Edirlei, et al. “Video-based Interactive Storytelling Using Real-time Video Compositing Techniques.” *Multimedia tools and Applications*, vol. 77, 2017, pp. 2333-2357. *ICAD PUC-Rio*, http://www.icad.puc-rio.br/wp-content/uploads/2017/12/Edirlei_2017_Video_based_rendering.pdf.

Soares de Lima, Edirlei Everson, et al. “Director of Photography and Music Director for Interactive Storytelling.” *ISMIR 2005 - 6th International Conference on Music Information Retrieval*, 2005, pp. 129-137. *IEEE*, https://ieeexplore.ieee.org/document/5772280.

Stuart, Keith. “'Mozart would have made video game music': composer Eímear Noone on a winning art form.” *The Guardian*, 22 October 2019, https://www.theguardian.com/games/2019/oct/22/mozart-video-game-music-composer-eimear-noone. Accessed 26 November 2021.

Vaswani, Ashish, et al. “Attention Is All You Need.” 2017. *arXiv:1706.03762*, https://arxiv.org/abs/1706.03762.

Vyas, Apoorv, et al. “Fast Transformers with Clustered Attention.” 2020. *arXiv:2007.04825*, https://arxiv.org/abs/2007.04825.

Yu, Yi, et al. “Conditional LSTM-GAN for Melody Generation from Lyrics.” *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 17, 2021, pp. 1–20. *arXiv:1908.05551*, https://arxiv.org/abs/1908.05551.

Appendix A: Schedules

1. Planned

Activities	*Deadlines*									
	Mar 2021	*Apr 2021*	*May 2021*	*June 2021*	*July 2021*	*Aug 2021*	*Sept 2021*	*Oct 2021*	*Nov 2021*	*Dec 2021*
Project I										
Form Delivery										
Theme Study										
Objectives Alignment										
Proposal Delivery										
Proposal Response										
Align Requirements and Plan R&D										
Create the First Project Proposals										
Delivery of the Final Report										
Project II										
Build First "Proof of Concept" Prototypes and Experiment										
Preparation and Execution of Tests										
Perform the Analysis of Results										
Final Project Delivery II										
Final Project Stands II										

Table 1: The planned schedule

2. Current

Activities	Deadlines									
	Mar 2021	Apr 2021	May 2021	June 2021	July 2021	Aug 2021	Sept 2021	Oct 2021	Nov 2021	Dec 2021
Project I										
Form Delivery										
Theme Study										
Objectives Alignment										
Proposal Delivery										
Proposal Response										
Align Requirements and Plan R&D										
Create the First Project Proposals										
Delivery of the Final Report										
Project II										
Build First "Proof of Concept" Prototypes and Experiment										
Preparation and Execution of Tests										
Perform the Analysis of Results										
Final Project Delivery II										
Final Project Stands II										

Table 2: The current implemented schedule