Université de Montréal

Algorithmic authority in music creation: the beauty of losing control / De l'autorité algorithmique dans la création musicale : la beauté de la perte de contrôle

par

Tiago Bortoletto Vaz

Faculté de Musique

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maîtrise en musique (M. Mus)
option composition

Janvier 2023

Université de Montréal

Ce mémoire intitulé

Algorithmic authority in music creation: the beauty of losing control / De l'autorité algorithmique dans la création musicale : la beauté de la perte de contrôle

présenté par

Tiago Bortoletto Vaz

a été évalué par un jury composé des personnes suivantes :

Myriam Boucher

(président-rapporteur)

Pierre Michaud

(directeur de recherche)

Dominic Thibault

(membre du jury)

Résumé

De plus en plus de tâches humaines sont prises en charge par les algorithmes dans le domaine des arts. Avec les nouvelles techniques d'intelligence artificielle (IA) disponibles, qui reposent généralement sur le concept d'*autoapprentissage*, la frontière entre l'assistance informatique et la création algorithmique proprement dite s'estompe.

À titre de mise en contexte, je présente, à l'aide d'exemples récents, un aperçu de la manière dont les artistes utilisent des algorithmes sophistiqués dans leur travail - et comment cette nouvelle forme d'art piloté par l'IA pourrait différer des premières formes d'art générées par ordinateur. Ensuite, sur la base de théories récentes issues d'une société post-humaniste et la montée de ce que certains auteurs appellent le *dataisme*, j'aborde des questions liées à l'autonomie, à la collaboration, aux droits d'auteur et au contrôle de la composition sur la création sonore et musicale.

D'un point de vue pratique, je présente les logiciels libres et open source (FLOSS, de l'anglais *Free/Libre and Open Source Software*) qui ont pris une place constante dans mon processus de composition ; et je discute de la façon dont leur écosystème, principalement dans le domaine de l'IA, a façonné mon travail au fil des ans, tant d'un point de vue esthétique que pratique.

Une série de trois pièces mixtes produites dans le cadre de ce projet de recherche-création est ensuite analysée. Je présente les différentes dimensions dans lesquelles le concept d'*autorité algorithmique* a pris part à mon processus de composition, des approches techniques aux choix esthétiques. Enfin, je propose des stratégies de notation musicale, basées sur des standards ouverts, visant à assurer la lisibilité, et donc la pérennité de mon travail.

Mots-clés : musique mixte, composition assistée par ordinateur, IA, logiciel libre

Abstract

Algorithms have taken over an increasing number of human tasks in the realm of the arts. With newly available AI techniques, typically relying on the concept of *self-learning*, the line separating computer assistance from actual algorithmic creation is blurring.

To contextualize, through recent illustrative examples I elaborate an overview of ways in which artists are making use of sophisticated algorithms in their work – and how this new form of AI-driven art might differ from early computer-generated art. Then, based on recent theories concerning a post-humanist society and the rise of what some authors call *dataism*, I discuss issues related to autonomy, collaboration, authorship and compositional control over the creation of sound and music.

From a practical perspective, I present the *Free/Libre and Open Source Software* (FLOSS) that have been a consistent presence in my compositional process; and discuss how their ecosystem, mainly in the domain of AI, has shaped my work over the last decade from both aesthetic and practical standpoints.

A series of three mixed pieces produced as part of this research-creation project is then analyzed. I present the different dimensions in which the concept of *algorithmic authority* took part in my composition process, from technical approaches to aesthetic choices. Finally, I propose some strategies for music notation, based on open standards, aiming to assure the readability, and therefore the perpetuity of my work.

Keywords : mixed music, computer-assisted composition, AI, FLOSS

Contents

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
AGI	Artificial General Intelligence
ANN	Artificial Neural Networks
BCI	Brain-computer interface
DL	Deep Learning
DSP	Digital Signal Processing
EEG	Electroencephalogram
FLOSS	Free/Libre and Open Source Software
GPT-2	Generative Pre-trained Transformer 2
GUI	Graphical User Interface
HCI	Human-Computer Interaction
MIDI	Musical Instrument Digital Interface
ML	Machine Learning
NLP	Natural Language Processing
OSC	Open Sound Control
RNN	Recurrent Neural Network
TTS	Text-to-Speech
XML	Extensible Markup Language

Acknowledgments

I first thank my long-time partner Tássia who, besides her tireless moral support, has been a gifted mentor in programming and LaTeX for many years already.

I also thank everyone who participated in one way or another in this project: my supervisor Pierre Michaud for his open-mindedness and human approach throughout my master's program; Hippolyte Vendra for his support with recordings; Daniel Áñez for accepting the challenge of playing with me and my algorithms, keeping an incredible energy during the entire process.

Finally, I cannot help but thank all the *Free/Libre and Open Source Software* developers who, by dedicating countless hours of voluntary work, open the doors to this kind of experimental work. To name one, Olivier Belanger, who has for many years already developed one of the best audio engines currently available, Pyo.

Introduction

...then the answers, instead of coming from my likes and dislikes, come from chance operations, and that has the effect of opening me to possibilities that I hadn't considered.

John Cage [Montague, 1985]

Humans have a deep desire for control. Feeling that we are in control lets us believe that we can shape our work to our liking, or to what we think will be somewhat appreciated by others. Studies suggest that achievements that we feel are under our control boost our morale, therefore improving our well-being [Ryan and Deci, 2000]. This alone could be enough to explain one's desire for exercising personal autonomy on any aspect of one's life, yet evidence from many areas of research has supported the idea that our need for control might also be biologically motivated. [Leotti et al., 2010]

The sense of control also brings comfort because it creates the perception that we are not under someone else's control [Raghunathan, 2021], and this is closely related to our evolutionary survival mechanisms. Not surprisingly, this constant pursuit for self-determination is no different in the realm of the arts. Approaching the subjects of *autonomy* and *control* in a creative process, however, is a challenging task – after all, as well noted by researcher Alan Chamberlain, "what is perceived as autonomous, may be viewed as control in another setting, in the same way that one might see order and patterns in chaos, while another person may interpret the same phenomena as truly random and devoid of meaning" [Chamberlain, 2017].

Yet, such perceptual blurring opens up compelling creative possibilities in an era where the advancement of algorithms is redefining the idea of human autonomy over human decisions. Indeed, over the last decades, the very idea of *control* has been creatively confronted in the realm of music composition: *aleatory music*, *generative music*, *intuitive music*, *open form*, *stochastic music*, and *indeterminacy* are just a few examples of now-established approaches closely related to the subject.

Greatly inspired by these concepts, I elaborate in the next sections how my attempts to shift authority away from the composer have been at once revealing and disconcerting. By making use of the most recent *Machine Learning* (ML) techniques, I seek a sense of maximal liberation from the constant control we are used to exercising in artistic activity. In a back and forth exchange process between composer, interpreters and algorithms, I explore the *glitchy* side of AI in music, as well as making its weaknesses and downsides the theme of my pieces. The result leads me to a challenging state of equanimity in the appreciation of the generated musical outcome, which, in my view, has its own peculiar beauty in the midst of what appears to be a lack of order.

In chapter 1, to offer some context and thoughts that have influenced this work, I discuss how our understanding of AI has evolved over recent years, and how it has affected the way we produce and perceive some forms of art. On a more practical level, in chapter 2 I present, through objective examples, some FLOSS that have significantly shaped my work over the last decade. In chapter 3 I discuss the creation process of three mixed pieces conceived in the context of my masters project, followed by a review where my experiments towards finding a collaborative approach integrating composer, performers, listeners and algorithms have *perhaps* succeeded, and what could be improved upon.

Chapter 1

Artificial Intelligence and the arts

Man is losing his humanity and becoming a thing amongst the things he produces.

Aman Khanna

Our resistance to the idea that algorithms can autonomously create art is often based on the assumption that non-sentient creatures lack the necessary understanding of socio-historical contexts. This thought is well expressed in the article "Creativity is, and always will be, a human endeavor"[1] by philosopher Sean D. Kelly. Here, the author cites Schoenberg as a notable example of a creative innovator "not just because he managed to create a new way of composing music but because people could see in it a vision of what the world should be.". Going further, he claims that "...his innovation was not just to find a new algorithm for composing music; it was to find a way of thinking about what music is that allows it to speak to *what is needed now*". Such sophisticated features found in Schoenberg's work (such as a *vision of what the world should be* and *what is needed now*) are used by the author to reinforce his proposition, widely supported in both artistic and academic circles, that human creative achievement is in some way socially embedded, therefore will not succumb to advances in artificial intelligence.

[1]`https://www.technologyreview.com/2019/02/21/239489/a-philosopher-argues-that-an-ai-can-never-be-an-artist/` (accessed on Dec 01, 2022).

Another supposedly inherent human skill that is also regarded as indispensable to the creative process is the *social awareness*, or the ability to recognize the perspectives of others and apply this understanding to interactions with them. After all, how could a bunch of circuits be able to ingeniously manipulate shapes, lines and colors in order to produce sensations of volume, space and movement? Or, how could a thing incapable of perceiving and experiencing be able to compose a refined orchestral work provoking the deepest emotional responses from its listeners? How could a secret algorithm, which has never faced the consequences of oppression, climate change or police brutality, be apt to compose a cathartic, provocative and yet melodic hip-hop song? Indeed, such reluctance towards pure AI-driven art production has been advocated by notable thinkers over the last decades. The following excerpt from the book *L'Immatériel* illustrates the reasoning behind this sentiment:

> *Le sentiment de manquer, le besoin de se dépasser vers la satisfaction de ce manque sont constitutifs de la conscience vivante. L'intelligence se développe sur cette base et en tire l'impulsion première de vivre. La conception machinique de l'intelligence la présuppose comme étant toujours déjà là, programmée dans le cerveau, prête à être mobilisée. Mais l'intelligence n'est précisément pas un programme déjà écrit : elle n'existe que vivante comme capacité de se produire selon ses propres intentions ; et cette capacité de se faire manque, qui est au fondement de la capacité de créer, d'imaginer, de douter, de changer, bref de s'autodéterminer, n'est pas programmable dans un logiciel. Elle n'est pas programmable parce que le cerveau n'est pas un ensemble de programmes écrits et transcriptibles : il est l'organe vivant d'un corps vivant, un organe qui ne cesse de se programmer et de se reprogrammer lui- même.*
>
> [Gorz, 2010]

André Gorz was a social philosopher who worked closely with Sartre, being strongly influenced by existentialism and phenomenology. He believed that intelligence cannot exist without a body attached to it and his reasoning is closely related to the absence of consciousness of the machine. Later in the same book, he affirms:

> *Or, si l'intelligence doit exister, évoluer dans l'espace et le temps, être capable d'apprendre, de s'enrichir par l'expérience, elle a besoin d'un corps vivant. Mieux : elle a besoin de se donner, de créer son corps, de créer sa vie, à sa mesure. Pour créer l'intelligence artificielle, il faut donc créer la vie artificielle.*

Nonetheless, a point to ponder in Gorz's and many other bright minds' observations lies in the traditional characterization of AI, which has for a long time been pictured as "the science and engineering of making intelligent machines". This definition was proposed by John McCarthy, the scientist who coined the term AI itself in 1969 – and its resemblance to Gorz's views is clearly stated in the following conclusion, still from *L'Immatériel*:

> *L'ambition des pionniers de l'ia et de la VA [vie artificielle] se révélera autrement plus grande : il s'agit pour eux d'abolir la nature et le genre humain pour créer une « super-civilisation » robotique, un « au-delà de l'humanité » qui façonnera l'univers à son image et « transformera l'être humain en quelque chose de complètement différent ».*

Through McCarthy's perspective, Gorz's response to the AI and Artificial Life (AL) pioneers is precise and correct. However, though such assumptions were followed by respected writers such as physicist Stephen Hawking and philosopher Nick Bostrom, the dystopic scenario where AI rebels against humans and turns into something out of control has been radically reviewed by contemporary thinkers such as historian Yuval Harari and computer scientist Jaron Lanier (who is also a skilled musician). These younger authors will, unsurprisingly, rethink AI through a new viewpoint – as something something closer to "the study and design of intelligent agents" where an *intelligent agent* can be seen as a system that perceives its environment and takes actions that maximize its chances of success. [Russell and Norvig, 2009]

Yuval Harari is a historian and philosopher who became a reference in debates about the future of technology and its impacts in humankind. He elaborates a far less romanticized definition of AI, as presented in his article *Why Technology Favors Tyranny* [Harari, 2018]:

> *AI frightens many people because they don't trust it to remain obedient. Science fiction makes much of the possibility that computers or robots will develop consciousness — and shortly thereafter will try to kill all humans. But there is no particular reason to believe that AI will develop consciousness as it becomes more intelligent. We should instead fear AI because it will probably always obey its human masters, and never rebel. AI is a tool and a weapon unlike any other that human beings have developed; it will almost certainly allow the already powerful to consolidate their power further.*

According to him, the most powerful algorithms are currently controlled by a small *tech elite*, and have achieved great power over humankind mostly due to the generosity (or naiveness) of humans themselves, who deliberately provide such algorithms with the raw material they need in an unquestioning fashion. This phenomenon, often referred to as "data colonialism", is considered analogous to other processes of colonialism throughout history. However, this new format, rather than being based on exogenous domination, aims for the "reconfiguration of human life around the maximization of data collection for profit", therefore pushing humanity to "a large-scale social, economic and legal transformation based on the massive appropriation of social life through data extraction" [Couldry and Mejias, 2019].

This *Big Data revolution*, leading to an inevitable transformation of our social lives – including the way we produce and appreciate art – has inspired Harari to predict a new age in human history, dominated by an emerging ideology known as *dataism*[2]: an inflection point in human history at which the authority to make the most important decisions of our lives shifts from ourselves to algorithms – therefore marking the end of the current era of humanism. This concept, which envisions such a dramatic transformation in human behavior, along with its consequences for artistic creation, is what I try to explore in the pieces presented later in this work.

[2]The term was firstly used by David Brooks in his article "The Philosophy of Data" from 2013 available at `https://www.nytimes.com/2013/02/05/opinion/brooks-the-philosophy-of-data.html` (accessed on Dec 01, 2022).

1.1. Is AI art art?

The question of whether a computer can think is no more interesting than the question of whether a submarine can swim.

Edsger Dijkstra

Our transcendence adorns,
That society of the stars seem to be the secret.

PoemPortraits algorithm[3]

A few decades ago, algorithms were already capable of assisting artists in many domains, being mostly supported by complex, yet deterministic sets of rules. Such algorithms could easily play with geometry, serial music, tonal counterpoint or use natural language in order to produce interesting outputs. Not surprisingly, they have evolved, and now they are able to process a much greater amount and variety of data. Furthermore, algorithms became capable of *learning*.

As elaborated by Wang, "with the use of machine learning[4] and algorithmic techniques, programming compositional algorithms has been largely substituted by programming machines that extract patterns or rules from musical examples." [Wang et al., 2016]. That being said, let's still consider the fact that these "programming machines" lack the understanding of complex social contexts required to create music. But is *context* truly a requirement? Well, if we take a materialist view – and accept that mind and consciousness are merely byproducts of biochemistry – why would algorithms not be able to create music that provokes the emotional reactions some composers do? It has been widely revealed that algorithms are getting to the point of knowing – and manipulating – human behavior much better than humans (so human composers!) themselves. A Cambridge and Stanford University study from 2015 shows that by tracking 300 Facebook likes, an algorithm can predict your personality as well as your spouse [Youyou et al., 2015]. The conclusion of this and

[3]`https://artsexperiments.withgoogle.com/poemportraits` (accessed on April 26, 2022)

[4]Machine learning (ML) is a subfield of artificial intelligence, based on the idea that a machine can learn and imitate intelligent human behavior. An interesting article explaining how ML works is available at `https://deepai.org/machine-learning-glossary-and-terms/machine-learning` (accessed on Dec 20, 2022).

many other studies is that it takes very little for algorithms to know us better than we know ourselves, especially considering that we keep feeding them with data more sophisticated than *Likes.* They can also learn and predict our daily habits by consensually tracking our voice[5].

Apparently, we seem to enjoy providing algorithms with high-precision biometric data such as pulse rate, pressure, sweat composition, dilation of the pupils[6], or even real-time orgasm data[7]. Even more stunning, algorithms are able to measure our stress level by detecting micro pulses of blood in our face when we use a camera[8], eliminating the need for any kind of extra biometric gadgets. Thus, the end of the poker face.[9]

In this direction, Harari offers the following observation:

> *Art is often said to provide us with our ultimate (and uniquely human) sanctuary. [...] Yet it is hard to see why artistic creation will be safe from the algorithms. Why are we so sure computers will be unable to better us in the composition of music? According to the life sciences, art is not the product of some enchanted spirit or metaphysical soul, but rather of organic algorithms recognizing mathematical patterns. If so, there is no reason why non-organic algorithms couldn't master it.*
>
> *[Harari et al., 2018]*

A visually captivating example of (art?)work produced by "non-organic algorithms recognizing mathematical patterns" was "The Next Rembrandt", released in 2016 by Microsoft in collaboration with public and private institutions.

"The Next Rembrandt" is a painting resulting from the analysis of more than 300 works produced by Dutch painter Rembrandt between 1632 and 1642. This analysis was performed by a *deep learning* (DL) system capable of processing enough *features* to finally

[5] `https://www.washingtonpost.com/technology/2019/05/06/alexa-has-been-eavesdropping-you-this-whole-time/` (accessed on April 26, 2022)
[6] `https://www.npr.org/sections/13.7/2018/02/28/589477976/biometric-data-and-the-rise-of-digital-dictatorship` (accessed 26 April 2022)
[7] `https://www.forbes.com/sites/janetwburns/2016/01/20/wearables-may-soon-track-heartbreak-orgasms-in-real-time/` (accessed 26 April 2022)
[8] `https://www.wired.com/story/artificial-intelligence-yuval-noah-harari-tristan-harris/` (accessed 26 April 2022)
[9] `https://www.ted.com/talks/poppy_crum_technology_that_knows_what_you_re_feeling` (accessed 26 April 2022)

produce a 3D-printed portrait of a “Caucasian male with facial hair, between the ages of thirty and forty, wearing black clothes with a white collar and a hat, facing to the right.” mimicking Rembrandt’s style.[10]

The significant attention that this project received by the media, as well as by the AI and arts communities, may reveal a trend in the current AI discourse which author and artist Joanna Zylinska defines as a “curatorial fascination with what we may call ‘AI imitation work’, also known as ‘style transfer”’ [Zylinska, 2020].

When looking for AI techniques for my own music creation I faced a similar scenario: the available AI algorithms were indeed mostly built to learn and imitate styles, i.e. to artificially generate Debussy or Bach. Yet such algorithmic audacity can also be inspiring, as it has certainly influenced the work discussed in chapter 3.

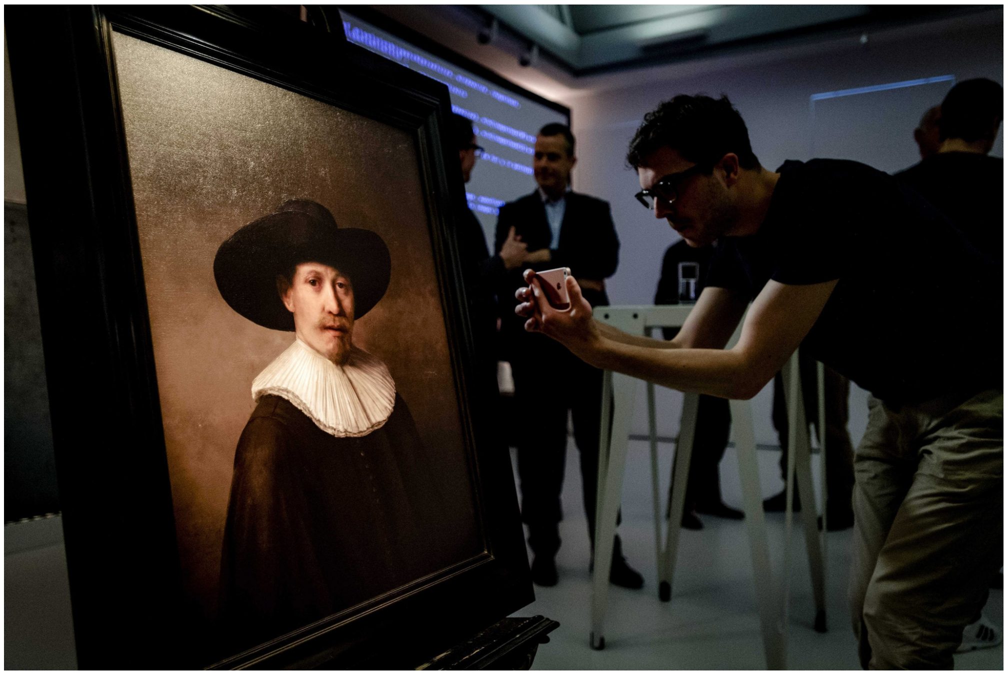


Figure 1.1. “The Next Rembrandt”, unveiled to media and enthusiasts in the Netherlands – photo from `https://nos.nl/artikel/2097253-the-next-rembrandt-3d-en-pixels` (accessed on Nov 30, 2022).

In a more nuanced position, Zylinksa argues that “...this supposedly scientific notion of perception, tied as it is to the expert idea of art, is precisely what tends to rile the general

[10]`https://www.nextrembrandt.com/` (accessed 26 April 2022)

public. Imitation art thus encourages populist sneering at experts, who may end up being 'taken in' by an artificially generated van Gogh or Bacon." [11].

Evidently, AI can also be used for producing non-imitative art. Distancing himself from this "style transfer" trend, Montreal-based artist Adam Basanta made use of similar deep learning techniques in the development of his *Landscape Past Future* (2019), a "re-thinking of landscape paintings through the lens of data-aggregation technologies."[12].

By extracting and processing photographic data from the permanent collections of major Museums, Basanta created new original canvases that "reveal both specific details of historically significant works as well overarching statistical features". Thus, unlike *The Next Rembrandt*, his work is devoid of the mimetic jargon too often present in today's forms of AI-driven art. Having worked as a software developer for this project, I admit to being influenced by his perspective on the application of AI to arts – in a hope that I am not *mimicking* his style too much in my musical creation.

[11]Zylinska's book "AI Art: Machine Visions and Warped Dreams" (2020) is an *open access book* licensed under *Creative Commons*. This book presents an enlightening critical view of AI applied to arts, including deep discussions on recent AI-driven works, which can also be individually consulted via its *(Art) Gallery of Links*. Its digital version is available at `http://openhumanitiespress.org/books/titles/ai-art/` (accessed 26 April 2022)

[12]`http://adambasanta.com/landscapepastfuture` (accessed 26 April 2022)

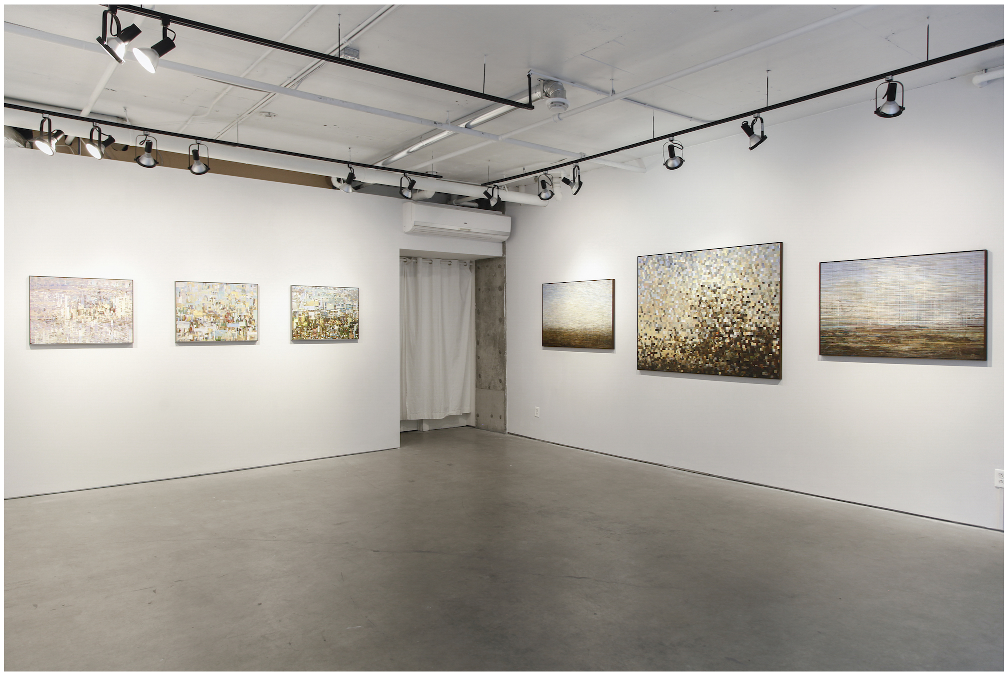


Figure 1.2. Computer-generated works in *Landscape Past Future*, Adam Basanta, 2019.

Back to Harari's position on the subject, he might agree with Zylinska that, regardless of considering such AI-based work as art, a huge impact in the world of arts production is to be expected:

> *Will all this result in great art? That depends on the definition of art. If beauty is indeed in the ears of the listener, and if the customer is always right, then biometric algorithms stand a chance of producing the best art in history. If art is about something deeper than human emotions, and should express a truth beyond our biochemical vibrations, biometric algorithms might not make very good artists. But nor do most humans. In order to enter the art market and displace many human composers and performers, algorithms won't have to begin by straightaway surpassing Tchaikovsky. It will be enough if they outperform Britney Spears.*
>
> *[Harari, 2019]*

It is inevitable that broader questions are raised once we face such a scenario: "What does it mean to be fascinated by realistic imitations?" [Jackson, 2017]. "Is there an ontological difference between early computer-generated art, net art and the more recent forms of AI-driven art?" [Zylinska, 2020]. "What is art *creation*? What do we mean by *art*? And, what do we mean by *machines* creating art?" [Coeckelbergh, 2017].

In order to address some of these questions (and certainly to raise new ones!), there have been many experiments using the *Turing Test*[13] (TT) in the field of AI-driven art. Paintings and music are often the content selected for evaluation, and tests are performed in many formats, from academic studies to popular competitions. Findings have been at once interesting, diverse and... contradictory. [Daniele et al., 2021] Such experiments are usually focused on the participants' ability to distinguish AI-driven art to art made by humans – actually the first condition to pass the TT. A second condition, as proposed by Boden [Boden, 2010], lies in the art to be considered as having as much aesthetic value as one produced by a human being.

A recent study, however, intended to push the Turing Test one step further, by asking participants how their reception of a piece of art changed once knowing its creator's identity: "Rather than asking participants (as in the original parlor game) whether the concealed figure is a man or a woman, it asks the fundamentally more important question of how their evaluation of the individual changes on the basis of that information." [Hong and Curran, 2019]

For this study, 288 participants were asked to evaluate visual artworks by rating over eight criteria: *originality*, *composition*, *personal style*, *expression*, *experimentation*, *degree of improvement*, *aesthetic value* and *successful communication of idea.* For the evaluation, four groups were created based on the identity of artists (AI or human). Through a rigorous method, this study concluded that the evaluation coming from participants who knew the creator's identity did not present a discrepancy with those who did not know. And also, that there is a clear difference between the blind evaluation of AI-driven art and human-created art, where human-created artworks were given a substantial higher rating in *composition*, *degree of expression* and *aesthetic value.* The authors then argue that "these variables can

[13] `https://plato.stanford.edu/entries/turing-test/` (accessed on April 26, 2022)

be seen as either strengths of human artists or alternatively the goals that AI art generators should fulfill to produce *worthy* art products."

Ultimately, are these (*composition*, *degree of expression* and *aesthetic value*) the missing features for machines to create "great art"? So that, beyond outperforming Britney Spears, algorithms can surpass the mastery of Tchaikovsky, as in Harari's observations? As suggested by [New Scientist, 2017.], perhaps we should flip the question: if it is in fact possible to break down the style of most original composers into algorithms, so perhaps human artists are more machine-like than we would like to think?

Chapter 2

FLOSS and the arts

The interaction between FLOSS philosophy and digital art should not be seen as yet another category of art, but as an added layer. It acts as another dimension on top of existing fields of digital art and enriches the way artists and collectives can work by adding another degree of freedom in the creative process.

[Mansoux and Valk, 2008]

FLOSS is a term commonly used to designate a software which is considered Free/Libre or Open Source, or rather, that anyone is freely licensed to use, copy, study, and change in any way. The source code of a FLOSS is openly shared so that people are encouraged to voluntarily learn and improve it. [Goossens et al., 1994]

Many factors have contributed to the popularization of Free/Libre and Open Source Software in the artist's practice. First, the degree of flexibility intrinsic to FLOSS enables endless possibilities for creators, avoiding the artistic process from being limited by the predefined features of a commercial tool. As elaborated by Lee [Lee, 2008], "such a degree of flexibility in software re-use and modification is non-existent and strictly prohibited with proprietary software". Moreover, due to its distribution at no charge and its frequent compatibility with outdated/modest hardware, FLOSS extends possibilities for those with limited access to modern and expensive equipment. Also, FLOSS communities usually depend on motivated volunteers dedicated to *internationalization* (preparing the software to be translatable) and *localization* (actual translations to specific languages), commonly known as i18n and l10n[14]; this taskforce is essential to making the software accessible in less privileged regions of the globe.

It is true that, for some artists, software is no more than a means for the production of art. For others, however, software is an essential part of their creative work, occupying a space far beyond pragmatism, at once *revealing* and *being itself the result of* the artist's aesthetical and ethical beliefs. Either way, a given software will significantly shape one's art. In this direction, Lee [Lee, 2008] goes further by affirming that "...for instance, given the same hardware, artists who employ commercially packaged programs and those who develop their own software would often have contrasting ideology and aesthetics". FLOSS communities are indeed frequently associated with issues related to "Technology Ethics" (or "Technoethics") such as digital rights, freedom of information, censorship, privacy, transparency, copyright, *e-waste*, and so forth. Thus, it is not unusual to identify these concerns in the work of artists who value and use FLOSS. An example that well illustrates such an affinity is the work of *afrofuturist* maker Onyx Ashanti, who is also a "musician, geek and open-source advocate".[15] Ashanti works with cheap micro-controllers and a homemade 3D printer built from recycled electronics that he uses to develop kinetic modular digital synthesizers. In one of the "Far Off Sounds episodes", called "Onyx Ashanti Programs Himself"[16], Onyx Ashanti energetically talks about his aesthetic choices:

> *This version, you know, it's bulky and I... because I tell you one thing: [what] I have noticed in the [inaudible] year wearing this and all that, is the number of people who want it to be so... want to be cute! They want to oh... "can you make that less bulky, less... you know... and I'm like: "neither you or I even know what it is yet... We don't know what it is! Why, WHY does it have to be cute NOW?"*

The "bulky" interface that he refuses to make cute is one of his wearable musical instruments, showed in Figure 2.1. It is not hard to spot some influence of the issues

[14]`https://plato.stanford.edu/entries/turing-test/` (accessed on April 26, 2022)
[15]`https://howlround.com/commons/onyx-ashanti` (accessed 26 April 2022)
[16]`https://www.youtube.com/watch?v=qdizFnpp2oI` (accessed 26 April 2022)

previously listed in Ashanti's work. Yet it is difficult to conceive such machinery being built over a layer of expensive hardware and commercial software.

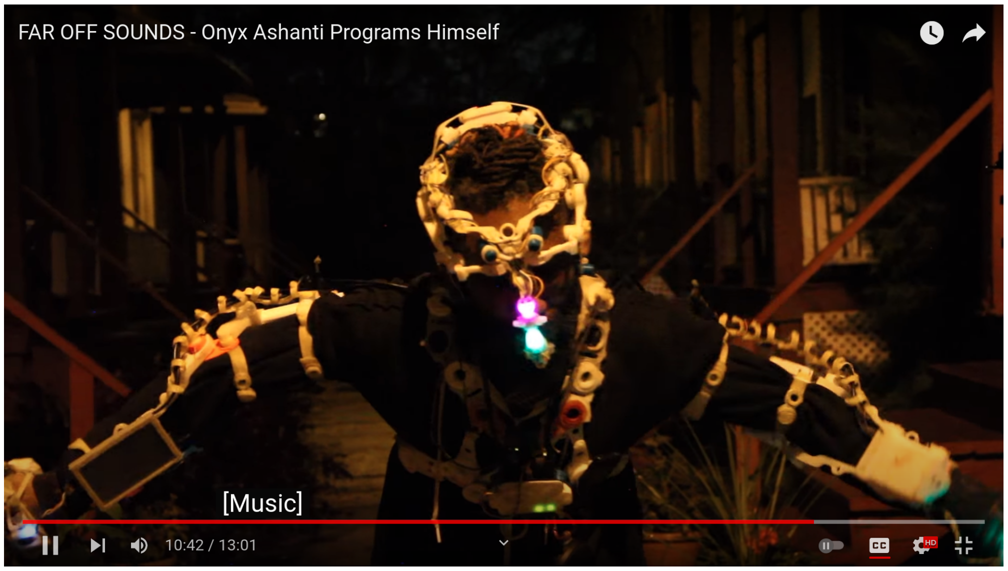


Figure 2.1. Artist Onyx Ashanti plays his wearable musical instrument built using FLOSS and open hardware.

Revealing a little of his creative process, Ashanti engages in a conversation with Miller Puckette (the author of *Max* and *Pure Data*), in which both emphasize the importance of FLOSS in the realm of the arts.[17]. In that conversation, published under the title "Write the Software and Let the World Have It", Puckette goes so far as to say, from a developer's perspective, that:

> *The way to spread a thing quickly and efficiently, and also to the widest possible population, was simply to make it open-source. That way, you don't have to do 90 percent of the work, which is wrapping the sucker up, and you can just do 10 percent of the work, which is the kernel. That appealed a hell of a lot to me – writing the software and letting the world have it, as opposed to trying to squeeze money out of anyone.*

Adding to this, Ashanti makes his point as a FLOSS user and artist, claiming that "people who would potentially be programmers have given away a lot of their autonomy to

[17]`https://howlround.com/write-software-and-let-world-have-it`

pre-designed [proprietary] systems and the idea of ease of use. It's not that the other thing [FLOSS] is harder, it's just that it's slightly more abstract.".

In the next section I present a plethora of free/libre and open-source software, then I briefly discuss how they have served me in creating music over the years. The main reason I decided to dedicate a full section to the subject is, above all, the fact that I strongly share the belief that "by understanding the tools artists use, one can gain greater insight into both their practice and artistic ideology" [Lee, 2008]; and also because, regrettably, at least in my academic circle, only very few colleagues are aware of the existence of FLOSS, let alone understand its ramifications in one's artistic practice.

2.1. An overview of available FLOSS for music creation

In 1957, the first computer program built for composing sounds was developed. It was called *Music I*. Engineer Max Matthews kept improving it up to his *Music V*. Meanwhile, a few composers were already on the track of creating computer-assisted musical works, such as "The Illiac Suit for String Quartet", by Lejaren Hiller. Since then, software technology has quickly evolved, becoming more and more accessible for domestic use. FLOSS like *CSound*, *Puredata*, and *SuperCollider* have provided musicians with a great range of possibilities and new creative approaches over the last decades. These and other classic tools remain in use nowadays, depending on a huge online community and being extensively documented.

In the next section, however, through an informal categorization I discuss the usage of more "modern" FLOSS in which I see the potential to become the new classic.

2.1.1. Operating system (OS)

Anyone who decides to move to a free OS nowadays will probably end up using a *GNU/Linux distribution*[18]. Distributions normally supports a great range of hardware and count on a helpful online community prompt to assist newcomers. Also, due to an enormous

[18]As defined in `https://www.howtogeek.com/132624/htg-explains-whats-a-linux-distro-and-how-are-they-different/` (accessed 26 April 2022).

number of tools available for GNU/Linux systems, many of the current major distributions feature a sophisticated and user-friendly way to manage software installation.

Currently, the most popular distributions are directly or indirectly derived from Debian GNU/Linux; a distribution itself first launched in 1993 and still highly active today, with over a thousand developers around the planet.[19] Some distributions derived from Debian, such as *Ubuntu Studio* and *AVLinux*, are designed for digital artists and offer certain advantages, for instance *realtime kernel*, lightweight desktop environments, pre-installed specialized software, and a complete *JACK*[20] audio setup.

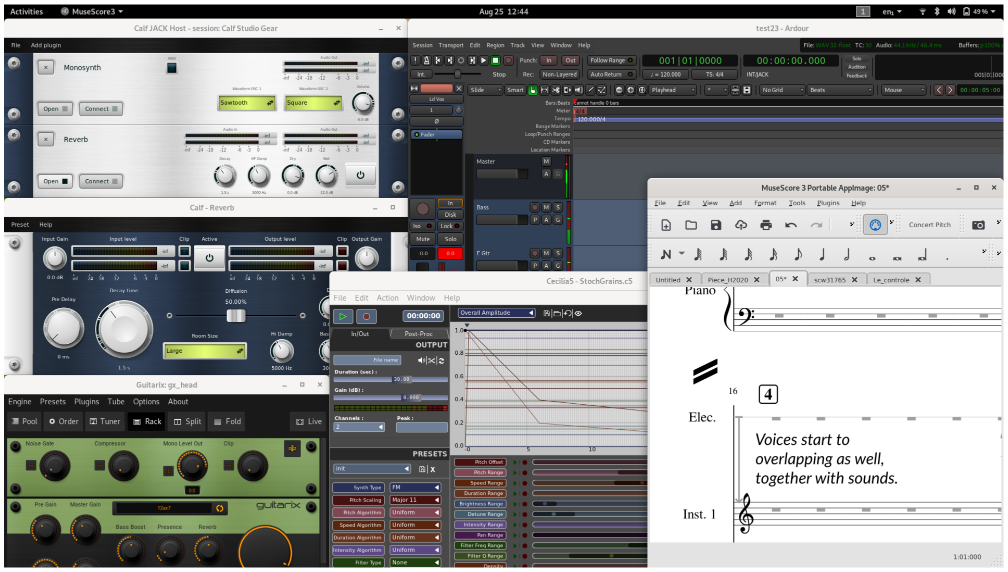

Calf JACK Host - session: Calf Studio Gear
Monosynth
Reverb
Calf - Reverb
Guitarix: gx_head
test23 - Ardour
MuseScore 3 Portable AppImage: 05*
Cecilia5 - StochGrains.c5
Piano
Elec.
Inst. 1
Voices start to overlapping as well, together with sounds.

Figure 2.2. A few tools for music creation running in a Debian GNU/Linux distribution: Calf Studio Gear, Guitarix, Ardour, Cecilia5 and Musescore.

Most of the use cases mentioned in the present work were performed in a Debian GNU/Linux system, which makes the instructions also compatible with trendy derivatives such as Ubuntu, Mint, Raspberry Pi OS (formerly Raspbian) and many others. My personal preference for Debian comes from a long time ago. I have been using and voluntarily contributing to Debian since 1999. In 2009 I officially became a developer and soon later I started packaging a variety of software related to music creation[21].

[19] `https://www.debian.org/` (accessed 26 April 2022)
[20] `https://jackaudio.org/` (accessed 26 April 2022)
[21] `https://qa.debian.org/developer.php?login=tiago` (accessed 26 April 2022)

2.1.2. Coding

A computer code can be seen as a form of notational language.[22] Coding has been extensively used in the realm of the arts to create generative systems, allowing artists to shift away from predetermined structures and to construct algorithms which "actively take part in the creative process with or without human intervention" [Lee, 2008]. Mainly focused on the impact of computer code for music creation, Magnusson states that "the linear timelines of staff notation or digital audio workstations seem to be slightly incongruous with code, as the former encourages deterministic and fixed approach to composition, whereas the latter opens up for indeterminacy, non-linearity and liveness" [Magnusson, 2014].

Naturally, terms such as *indeterminacy*, *non-linearity* and *liveness* are closely associated with the subject currently guiding my composition work. And to experiment with such possibilities I often produce code in a general-purpose programming language named *Python*. [Van Rossum and Drake Jr, 1995]

In an attempt to emphasize code readability and reuse, I tend to either program my own tools in pure Python or make use of third-party *code libraries* which sufficiently fit my needs. For instance, if I need to generate a random musical excerpt based on a collection of baroque corpus, or to randomly pick a scale among a set of options, I will mainly use *Music21* [Cuthbert, 2021]. Music21 not only offers a generous database of corpus, but also an enormous collection of high-level methods (or functions) to manipulate them. Furthermore, this library provides a great integration with standard digital music formats, such as *MusicXML*, *Humdrum*, *ABC*, *Musedata* and *MIDI4*. If I just need to do some calculation using musical primitives such as notes and duration, or perform operations such as transposition, retrograde, and inversion, I will certainly be well served by the *Pyknon* music library. Pyknon is a Python library provided as a resource for the (freely available) book "Music for Geeks and Nerds" by composer Pedro Kroger.[23]

[22]Curiously, as in music, a computer code is often performed by what we technically call *interpreters*.

[23]`https://www.pedrokroger.net/mfgan` (accessed 26 April 2022)

Not surprisingly, a Python code which uses either Music21 or Pyknon for specific tasks can be later integrated with other external libraries in order to meet the composer's needs. This means that, by coding in a single language, a composer could use Music21 to organize the core material of her piece, Pyknon to perform custom musical operations, *TensorFlow* to apply some AI to it, then *Pyo* to construct synthetic instruments or to create a time-based score to be live-performed.

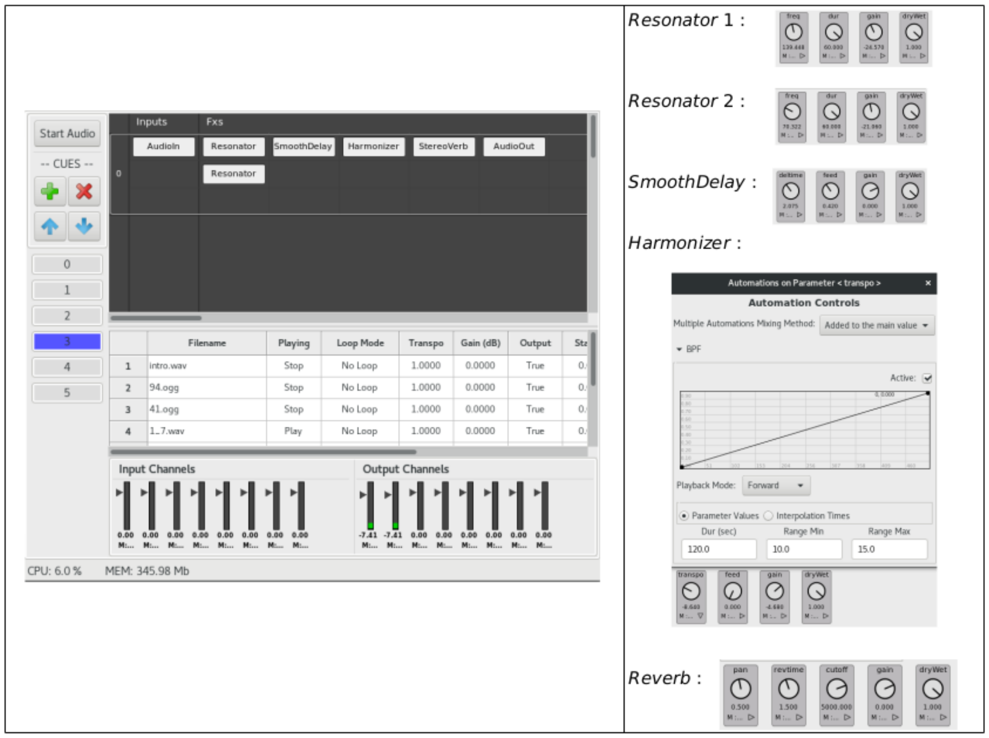
Many factors may influence one's decision when choosing a programming language for a project. A previous familiarity with Python certainly had a weight on my choice. However, I am more and more convinced that the ultimate benefit of using Python for music composition lies in the notable ecosystem of music-oriented libraries orbiting around it.

2.1.3. Live performance

A live performance, in the context of mixed/electronic music, usually involves low-latency signal processing, integrating and controlling external devices, and triggering events over time. Once again, the Python programming language happens to be a versatile option for achieving this goal.

For pieces that require a precise management of events I tend to use a Python-based FLOSS called Q-Live[24]. Q-Live is a software designed and developed at the Faculté de musique of the Université de Montréal which aims to address two issues commonly present in mixed music works: technological obsolescence (mostly on the software side); and the considerable amount of time required for the composer to master the basics of computer programming [Michaud et al., 2015]. Therefore, Q-Live provides users with a friendly graphical interface, while still remaining highly powerful and flexible through its modular architecture.

[24]Q-Live's source code is available at `https://github.com/belangeo/qlive` (accessed 26 April 2022)



	Filename	Playing	Loop Mode	Transpo	Gain (dB)	Output	Sta
1	intro.wav	Stop	No Loop	1.0000	0.0000	True	0.
2	94.ogg	Stop	No Loop	1.0000	0.0000	True	0.
3	41.ogg	Stop	No Loop	1.0000	0.0000	True	0.
4	1_7.wav	Play	No Loop	1.0000	0.0000	True	0.

Automations on Parameter < transpo >

Automation Controls

Multiple Automations Mixing Method: Added to the main value

BPF

Active: ✓

Playback Mode: Forward

[✓ Parameter Values] Interpolation Times

Dur (sec)	Range Min	Range Max
120.0	10.0	15.0

Figure 2.3. An illustration of Q-Live, a cue-based software to help the creation of mixed music.

Q-Live offers a series of built-in live effects which can be combined in chains and in tracks along with any external recorded material. The event management feature in Q-Live is provided through the concept of cues, which are supposed to be activated/deactivated in a time-based sequence during the performance. Each cue keeps information about all parameters previously set, being able to also trigger other events. Cues are numbered and can be directly managed via the computer hosting the software or through external MIDI devices.

With Q-Live being released as a FLOSS, anyone with some understanding of Python can add new audio objects to it. The excerpt in Figure 2.4 shows the basic code needed for a standard reverb effect using the Pyo *Freeverb* class, followed by a custom synth object built over a chain of Phasors and a fourth-order lowpass filter.

```
class FxFreeverb(BaseAudioObject):
    def __init__(self, chnls, ctrls, values, interps):
        BaseAudioObject.__init__(self, chnls, ctrls, values, interps)
        self.reverb = Freeverb(self.input, self.size.sig(), self.damp.sig(), 1, mul=self.
    gain.sig())
        self.process = Interp(self.input, self.reverb, self.dryWet.sig())
        self.output = Sig(self.process)

class MSTB303In(MidiSynth):
    def __init__(self, chnls, ctrls, values, interps):
        MidiSynth.__init__(self, chnls, ctrls, values, interps)
        self.factor = MToT(self.transpo.sig() + 60)
        self.phase = Phasor(self.pitch * self.factor)
        self.phase2 = Phasor(self.pitch * self.factor * 1.005)
        self.square = Sig((self.phase < self.duty.sig()) * 2 - 1, mul=self.envelope)
        self.square2 = Sig((self.phase2 < self.duty.sig()) * 2 - 1, mul=self.envelope)
        self.stereo = Mix([self.square.mix(1), self.square2.mix(1)], voices=2)
        self.process = MoogLP(self.stereo, self.freq.sig(), self.res.sig())
        self.output = Sig(self.process, mul=self.gain.sig())
```

Figure 2.4. Implementation of an audio object in Q-Live.

On the development side, there has been an ongoing effort to improve the file storage process, in order to simplify as much as possible the reconstitution of a composition which makes use of Q-Live. Currently, the software is able to save all media files in a platform-independent format, along with text files containing the audio objects (effects, algorithms) and a description of the events triggered (including parameters, automations and descriptions).

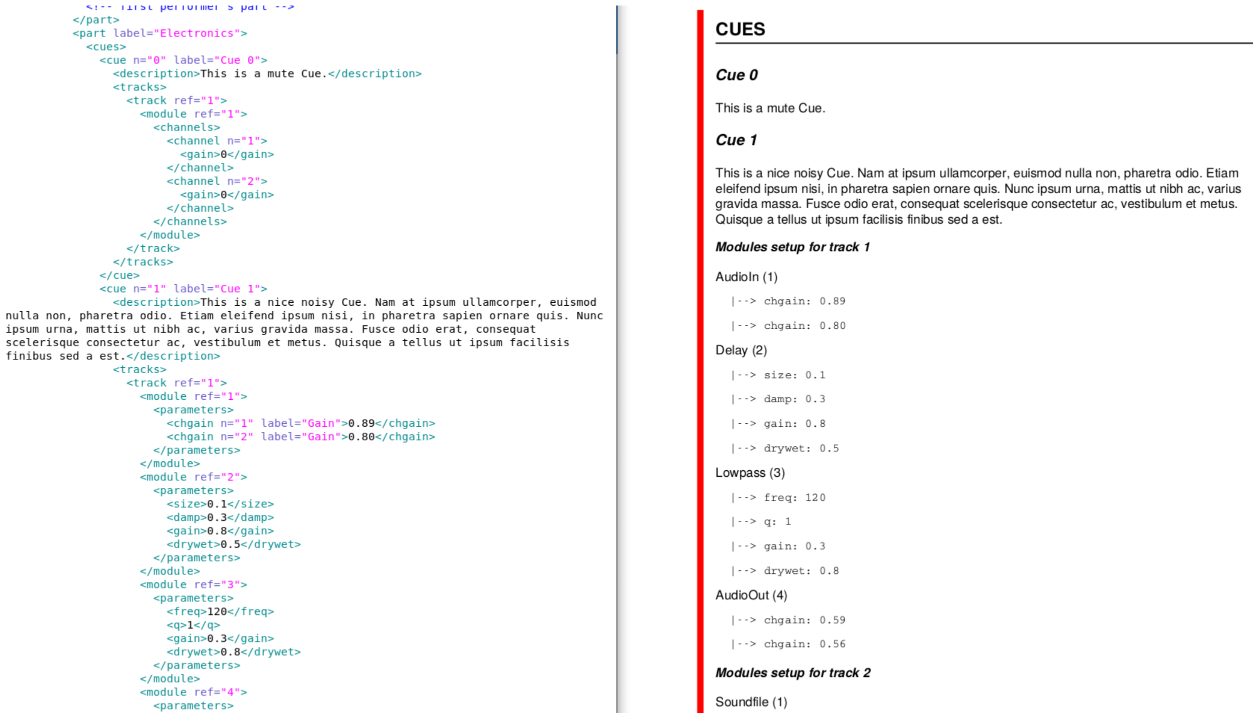
Being myself part of the Q-Live development team, I have contributed with some code for integrating it with a machine-readable structure proposed by *The Music Encoding Initiative* (MEI):

> *The Music Encoding Initiative [...] brings together specialists from various music research communities, including technologists, librarians, historians, and theorists in a common effort to discuss and define best practices for representing a broad range of musical documents and structures. The results of these discussions are then formalized into the MEI schema, a core set of rules for recording physical and intellectual characteristics of music notation*

documents expressed as an eXtensible Markup Language (XML) schema.

[MEI, 2021]

An example of a MEI structured file generated by Q-Live is illustrated in Figure 2.5 (left), followed by the actual output to be used in a score of a mixed piece (right).



```
        <!-- first performer's part -->
      </part>
      <part label="Electronics">
        <cues>
          <cue n="0" label="Cue 0">
            <description>This is a mute Cue.</description>
            <tracks>
              <track ref="1">
                <module ref="1">
                  <channels>
                    <channel n="1">
                      <gain>0</gain>
                    </channel>
                    <channel n="2">
                      <gain>0</gain>
                    </channel>
                  </channels>
                </module>
              </track>
            </tracks>
          </cue>
          <cue n="1" label="Cue 1">
            <description>This is a nice noisy Cue. Nam at ipsum ullamcorper, euismod
nulla non, pharetra odio. Etiam eleifend ipsum nisi, in pharetra sapien ornare quis. Nunc
ipsum urna, mattis ut nibh ac, varius gravida massa. Fusce odio erat, consequat
scelerisque consectetur ac, vestibulum et metus. Quisque a tellus ut ipsum facilisis
finibus sed a est.</description>
            <tracks>
              <track ref="1">
                <module ref="1">
                  <parameters>
                    <chgain n="1" label="Gain">0.89</chgain>
                    <chgain n="2" label="Gain">0.80</chgain>
                  </parameters>
                </module>
                <module ref="2">
                  <parameters>
                    <size>0.1</size>
                    <damp>0.3</damp>
                    <gain>0.8</gain>
                    <drywet>0.5</drywet>
                  </parameters>
                </module>
                <module ref="3">
                  <parameters>
                    <freq>120</freq>
                    <q>1</q>
                    <gain>0.3</gain>
                    <drywet>0.8</drywet>
                  </parameters>
                </module>
                <module ref="4">
                  <parameters>
```

CUES

Cue 0

This is a mute Cue.

Cue 1

This is a nice noisy Cue. Nam at ipsum ullamcorper, euismod nulla non, pharetra odio. Etiam eleifend ipsum nisi, in pharetra sapien ornare quis. Nunc ipsum urna, mattis ut nibh ac, varius gravida massa. Fusce odio erat, consequat scelerisque consectetur ac, vestibulum et metus. Quisque a tellus ut ipsum facilisis finibus sed a est.

Modules setup for track 1

AudioIn (1)

```
  |--> chgain: 0.89
  |--> chgain: 0.80
```

Delay (2)

```
  |--> size: 0.1
  |--> damp: 0.3
  |--> gain: 0.8
  |--> drywet: 0.5
```

Lowpass (3)

```
  |--> freq: 120
  |--> q: 1
  |--> gain: 0.3
  |--> drywet: 0.8
```

AudioOut (4)

```
  |--> chgain: 0.59
  |--> chgain: 0.56
```

Modules setup for track 2

Soundfile (1)

Figure 2.5. MEI output generated by Q-Live.

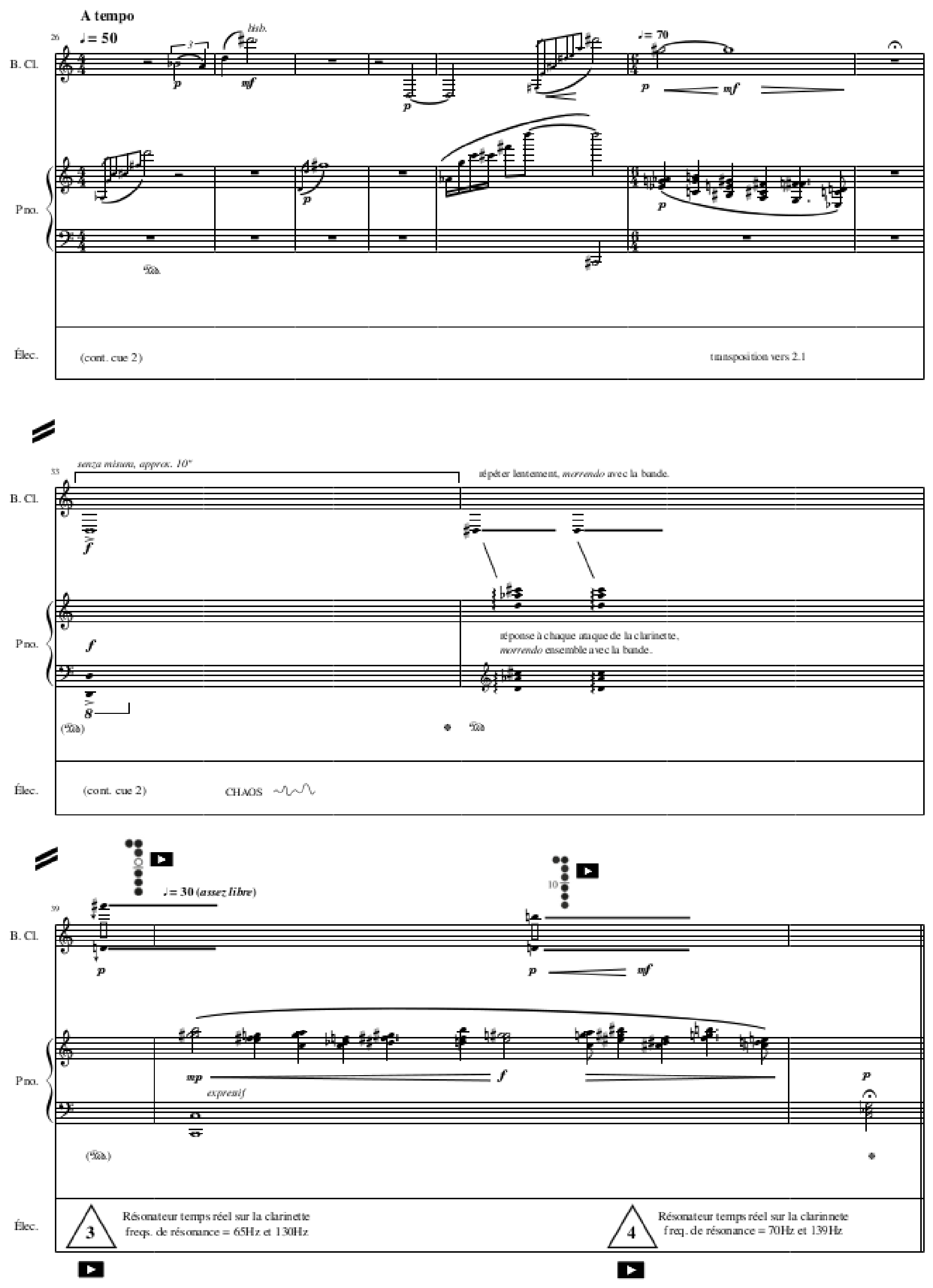

A tempo
♩= 50
bisb.
♩= 70
B. Cl.
Pno.
Élec.
(cont. cue 2)
transposition vers 2.1
senza misura, approx. 10"
répéter lentement, morrendo avec la bande.
réponse à chaque attaque de la clarinette, morrendo ensemble avec la bande.
CHAOS
♩= 30 (assez libre)
expressif
Résonateur temps réel sur la clarinette freqs. de résonance = 65Hz et 130Hz
Résonateur temps réel sur la clarinnete freq. de résonance = 70Hz et 139Hz
3
4

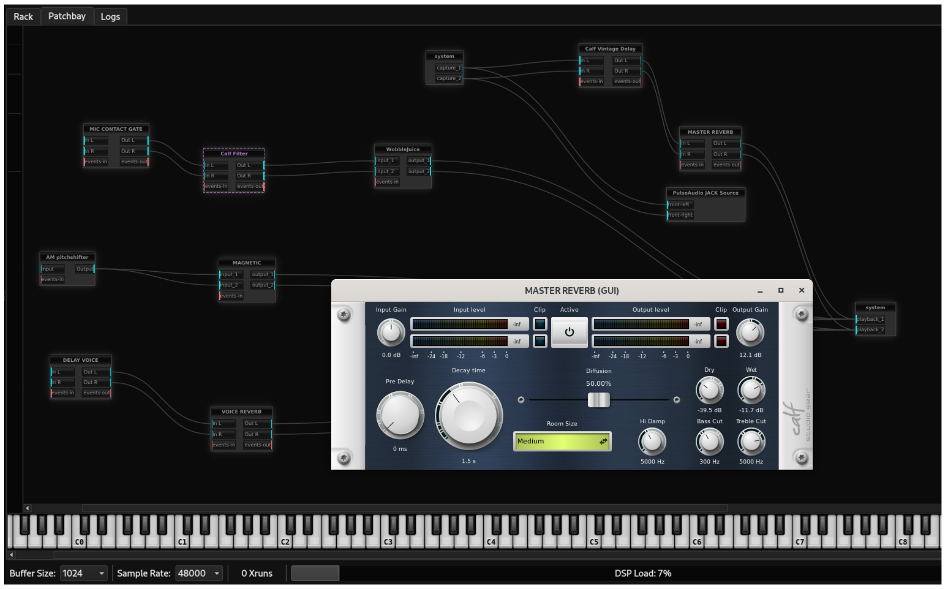
Figure 2.6. Excerpt from *Trois miniatures* (2017), Q-Live cues noted in the score.

I used to make exclusive use of Q-Live to control cues and effects in my past compositions, together with *Musescore* notation software[25] and *Inkscape*[26] to produce my scores. An example of how I usually notate Q-Lives cues in my scores can be seen in Figure 2.6,

[25]`https://musescore.org/en` (accessed 26 April 2022)
[26]`https://inkscape.org/` (accessed 26 April 2022)

which presents an extract of *Trois miniatures*, a piece that I composed in 2017 and was first performed in 2018 by the *Ensemble de Musique Contemporaine de l'UdeM*. In this excerpt we can see the indication of when the live electronics performer should trigger *cue 3*. The parameters for *cue 3* are illustrated in Figure 2.3, which is also a fragment from the *Trois miniatures* score.

Another piece of software serving a similar purpose, and that I've been exploring lately, is *Carla*. Carla is a "fully-featured modular audio plugin host, with support for many audio drivers and plugin formats. It has some nice features like transport control, automation of parameters via MIDI CC and remote control over OSC".[27] Carla certainly lacks some of the cue-related features found in Q-Live. Nonetheless, for my recent work – in which I approach music in a much more free form – I felt that tweaking with unfamiliar/obscure open-source effects would better match my intentions than controlling cues in a precise manner. Unlike Q-Live, Carla supports a range of standards for audio plugins, such LADSPA, DSSI, LV2, VST2 and VST3. Also, I highly appreciate the excellence of the audio effects provided by the *Calf Studio Gear*[28] project. Such effects were also made available as LV2 plugins by Calf developers, so that I could easily *host* them in my Carla setup, as illustrated in *Figure 2.7*.



Input Gain 0.0 dB
Output Gain 12.1 dB
Pre Delay 0 ms
Decay time 1.5 s
Diffusion 50.00%
Room Size Medium
Dry -39.5 dB
Wet -11.7 dB
Hi Damp 5000 Hz
Bass Cut 300 Hz
Treble Cut 5000 Hz

Buffer Size: 1024 Sample Rate: 48000 0 Xruns DSP Load: 7%

Figure 2.7. Carla *patchbay* view, hosting audio LV2 plugins such as Calf Reverb.

[27]`https://kx.studio/Applications:Carla` (accessed 26 April 2022)
[28]`http://calf-studio-gear.org/` (accessed 26 April 2022)

That being said, although I continue to use Q-Live for more deterministic pieces, for the creation project presented in chapter 3 I decided to adopt *Carla* instead.

2.1.4. Sound design and DSP

The existing sound generators and signal processors are a vast resource. In most scenarios, a composer will likely be well served by these *audio plugins* without the need to programme audio objects by themself. In some cases, however, coding either in a visual language such as Pure Data or in traditional scripting languages such Python, might open a new level of possibilities that would be hard to explore through pre-programmed tools. Evidently, FLOSS plugins can be modified to the composer's taste; however, the time it takes to understand a third-party code to the point where you are ready to improve it can easily become a limiting factor. A *library*, on the other hand, can offer the composer the best of both worlds. In the Python world, one of the most complete and advanced libraries in this regard is *Pyo*.

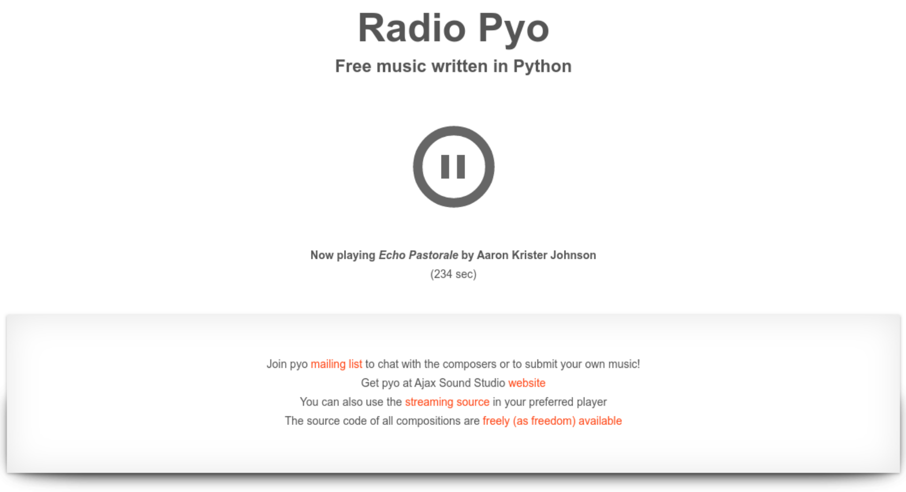
Pyo was created by Olivier Belanger and has been used for many years as a pedagogical tool in his classes at the Faculté de musique of the Université de Montréal. An online community around Pyo has grown over the years, becoming a helpful space for discussion, mostly through a public mailing list.[29]

Pyo is so complete that it can be used as a standalone solution for creating music. It provides an extensive set of signal generators, effects, filters, triggers, random generators, and much more. Users can build a rich setup of synthetic instruments, while at the same time benefiting from event sequencing classes provided for the production of time-based scores.

Composers from many countries and from different backgrounds are already using Pyo to make music. Some of their pieces can be listened to throught a dedicated online radio called *Radio Pyo*.[30] I am currently the person behind Radio Pyo, which is built over a set of tools deployed in a web server which processes and streams audio from Python scripts

[29]The Pyo mailing list is public and available at `https://groups.google.com/g/pyo-discuss` (accessed 26 April 2022)
[30]`http://radiopyo.acaia.ca` (accessed 26 April 2022)

submitted by composers. Music written in Pyo often happens to be a *generative product*, meaning that it will never sound the same way twice. Pieces from Radio Pyo are at once software and music. Released under a free license, they could be seen as a kind of *FLOSS music*, available for anyone who would like to read, learn or remix. Listeners can connect to the streaming source using their preferred player, or embed it in their websites. Source code (or scores?) of those pieces playing on Radio Pyo are available in a public git repository.[31].


Radio Pyo
Free music written in Python

Now playing *Echo Pastorale* by Aaron Krister Johnson
(234 sec)

Join pyo mailing list to chat with the composers or to submit your own music!
Get pyo at Ajax Sound Studio website
You can also use the streaming source in your preferred player
The source code of all compositions are freely (as freedom) available

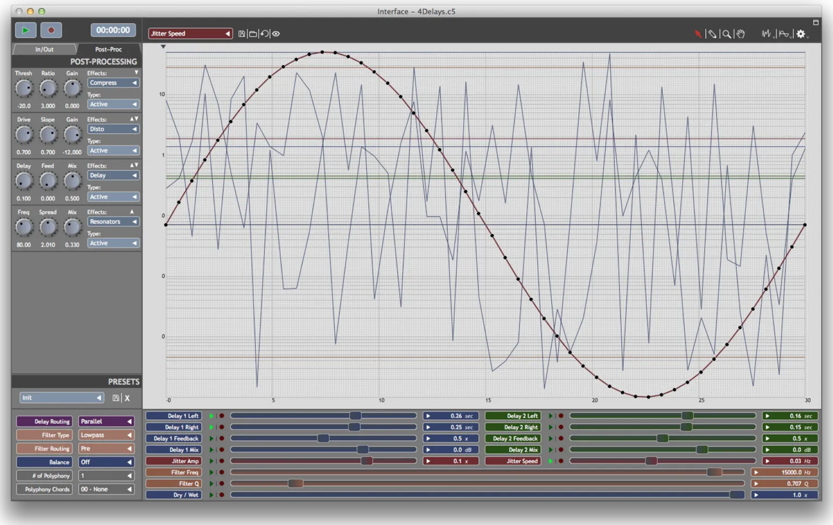
Figure 2.8. RadioPyo, free music written in Python, at `http://radiopyo.acaia.ca/`.

Pyo also offers its own *Open Sound Control* (OSC) and MIDI classes, making it easy to integrate digital instruments, time sequencing and external devices.

Although being a convenient library for sound design purposes, Pyo requires some degree of programming competence which composers may lack at first. Fortunately, Pyo developer Olivier Belanger has released two notable FLOSS offering intuitive and greatly flexible GUIs: *Cecilia5* (initially developed by composer Jean Piché in the 90's) and *Soundgrain*.

[31]`https://github.com/tiagovaz/radiopyo` (accessed 26 April 2022)

Cecilia is an audio signal processing environment aimed at sound designers which mangles sound in ways unheard of. It lets you create your own GUI using a simple syntax. Cecilia comes with many original built-in modules and presets for sound effects and synthesis.[32]



Thresh	Ratio	Gain	Effects:
-20.0	3.000	0.000	Compress
			Type: Active

Drive	Slope	Gain	Effects:
0.700	0.700	-12.000	Disto
			Type: Active

Delay	Feed	Mix	Effects:
0.100	0.000	0.500	Delay
			Type: Active

Freq	Spread	Mix	Effects:
80.00	2.010	0.330	Resonators
			Type: Active

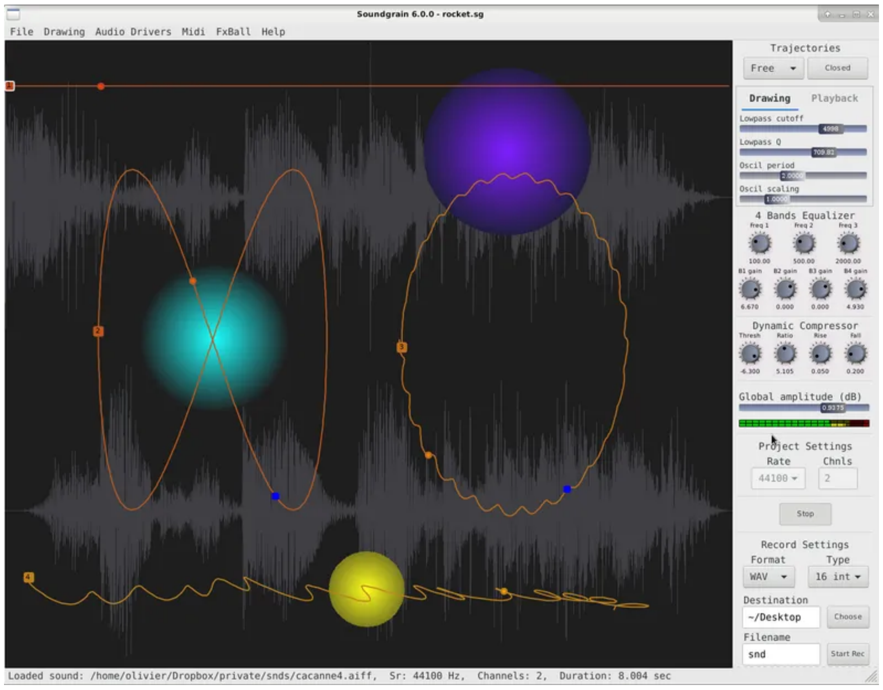
Delay Routing	Parallel
Filter Type	Lowpass
Filter Routing	Pre
Balance	Off
# of Polyphony	1
Polyphony Chords	00 - None

Delay 1 Left	0.26 sec	Delay 2 Left	0.16 sec
Delay 1 Right	0.25 sec	Delay 2 Right	0.15 sec
Delay 1 Feedback	0.5 x	Delay 2 Feedback	0.5 x
Delay 1 Mix	0.0 dB	Delay 2 Mix	0.0 dB
Jitter Amp	0.1 x	Jitter Speed	0.03 Hz
Filter Freq	15000.0 Hz		
Filter Q	0.707 Q		
Dry / Wet	1.0 x		

Figure 2.9. Cecilia, an audio signal processing environment.

Compared to Cecilia, Soundgrain is a simpler tool focused on the creation of new sounds through granular synthesis, by drawing and editing trajectories in an interesting graphical interface.[33] Both software use Pyo as their audio engine.

[32]`http://ajaxsoundstudio.com/software/cecilia/` (accessed on 26 April 2022).
[33]`http://ajaxsoundstudio.com/software/soundgrain/` (accessed 26 April 2022)



Lowpass cutoff 4998
Lowpass Q 709.82
Oscil period 2.0000
Oscil scaling 1.0000

4 Bands Equalizer
Freq 1 100.00, Freq 2 500.00, Freq 3 2000.00
B1 gain 6.670, B2 gain 0.000, B3 gain 0.000, B4 gain 4.930

Dynamic Compressor
Thresh -6.300, Ratio 5.105, Rise 0.050, Fall 0.200

Global amplitude (dB) 0.9375

Project Settings
Rate 44100, Chnls 2

Record Settings
Format WAV, Type 16 int
Destination ~/Desktop
Filename snd

```
Loaded sound: /home/olivier/Dropbox/private/snds/cacanne4.aiff,  Sr: 44100 Hz,  Channels: 2,  Duration: 8.004 sec
```

Figure 2.10. Soundgrain, granular synthesis tool.

2.1.5. Mapping and integration

Another feature of software as a medium that is often unnecessarily blocked, is the freedom to mix and match different applications without bumping into incompatibility issues, making it possible to use system and software as a creative toolbox in which all parts can be used together and in different configurations.

[Van Nort and Castagné, 2007]

Mixed pieces and electronic installations are commonly built over a range of input data from different sources, which are to be processed and transformed in order to generate the desired output. Generally speaking, this is what we call *mapping*. In this context, "the mapping stage is responsible for filling the ontological gap between the gestures performed by the user (or more precisely the gesture signals), and the parameters of the sound processes." [Van Nort and Castagné, 2007]. Mapping can become an extremely complex process, and often plays a significant role in the artist's work.

Technically speaking, mapping implies communication between different devices, either *real* or *virtual*, such as MIDI controllers, amplified instruments, a wide variety of sensors, stage lighting, digital synthesizers, and so on. Not only might this process require advanced programming skills, but also a deep understanding of the communication protocols involved.

Adopting open standards is a practice that is a great help during the implementation of mapping strategies. Nowadays, artists will most likely use *Open Sound Control* (OSC) under a wireless setup (for instance, the *Éponge* from Montreal-based composer Martin Marier [Marier, 2017]). Others will prefer the *Bluetooth Low Energy* protocol, as seen in the *Augmented Harp* [Sullivan et al., 2018]. Both technologies have been natively supported by major music production software and therefore have become the *de facto* standards for communication between multimedia devices.

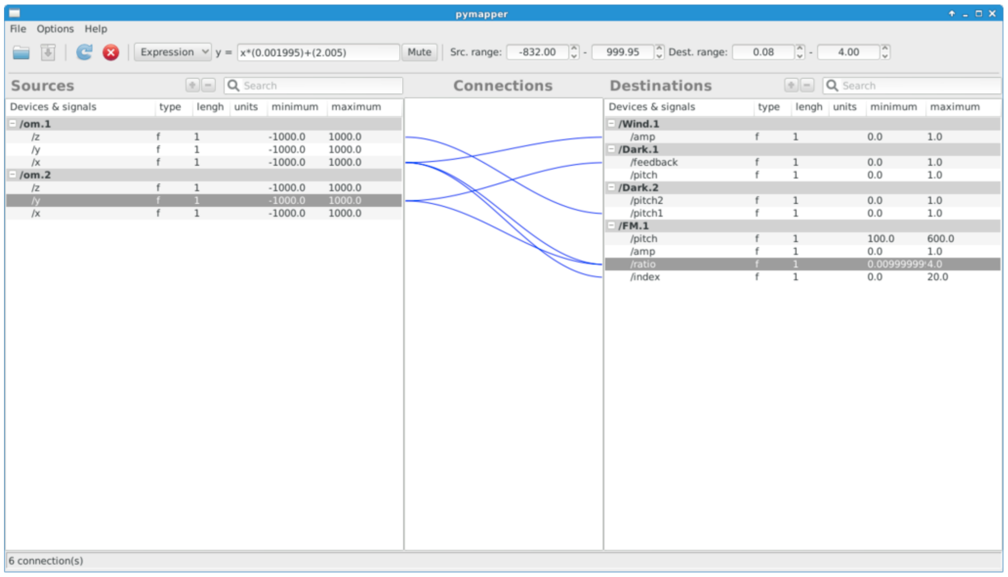
Certainly, one could use a programming language such as Python to develop an entire mapping code. This can easily become an arduous task, though, as mapping usually requires a time consuming tweaking of parameters and connections. I would argue that adding some layers on top of OSC can make this task easier. In this sense, two projects have recently caught my attention: *libmapper* and *Chataigne*.

libmapper: a library for connecting things

> *libmapper is an open-source, cross-platform software library for declaring data signals on a shared network and enabling arbitrary connections to be made between them. libmapper creates a distributed mapping system/network, with no central points of failure, the potential for tight collaboration and easy parallelization of media synthesis.*[34]

By using libmapper artists can dynamically experiment with many different mappings without touching any code whatsoever. A few user interfaces are available nowadays. At the time I started using libmapper, only one UI was fully functional – however, it was developed over Max/MSP, which does not run on GNU/Linux systems. Since libmapper is a FLOSS project with a documented API, I was able to develop a cross-platform UI called

[34]`http://libmapper.org/` (accessed 26 April 2022)



Expression y = x*(0.001995)+(2.005) Src. range: -832.00 - 999.95 Dest. range: 0.08 - 4.00

Sources

Devices & signals	type	lengh	units	minimum	maximum
/om.1					
/z	f	1		-1000.0	1000.0
/y	f	1		-1000.0	1000.0
/x	f	1		-1000.0	1000.0
/om.2					
/z	f	1		-1000.0	1000.0
/y	f	1		-1000.0	1000.0
/x	f	1		-1000.0	1000.0

Destinations

Devices & signals	type	lengh	units	minimum	maximum
/Wind.1					
/amp	f	1		0.0	1.0
/Dark.1					
/feedback	f	1		0.0	1.0
/pitch	f	1		0.0	1.0
/Dark.2					
/pitch2	f	1		0.0	1.0
/pitch1	f	1		0.0	1.0
/FM.1					
/pitch	f	1		100.0	600.0
/amp	f	1		0.0	1.0
/ratio	f	1		0.00999999	4.0
/index	f	1		0.0	20.0

6 connection(s)

Figure 2.11. PyMapperGui, a graphical interface for libmapper. Two accelerometers controlling sound parameters over a network.

PyMapperGUI[35]. Currently, a web UI named Webmapper is actively being developed and seems to support the most recent libmapper versions.[36] Libmapper is currently in a stable release and has been successfully used as the core mapping environment for the T-Stick digital music instrument [Nieva et al., 2018].

Chataigne: artist-friendly modular machine for art and technology

> *Chataigne is a free, open-source software made with one goal in mind : create a common tool for artists, technicians and developers who wish to use technology and synchronize software for shows, interactive installations or prototyping. It aims to be as simple as possible for basic interactions, but can be easily extended to create complex interactions.*
>
> *[Kuperberg, 2021]*

Chataigne supports an extensive range of hardware, software, and protocols, and can be easily expanded. It can work as a state machine, "which means you can create different states, each one being a group of rules that you can activate or deactivate as you wish". Users

[35]`https://github.com/tiagovaz/pyMapperGUI` (accessed 26 April 2022)
[36]`https://github.com/libmapper/webmapper` (accessed 26 April 2022)

can also create transitions between states, allowing all kinds of automation and autonomous interactions between those states.

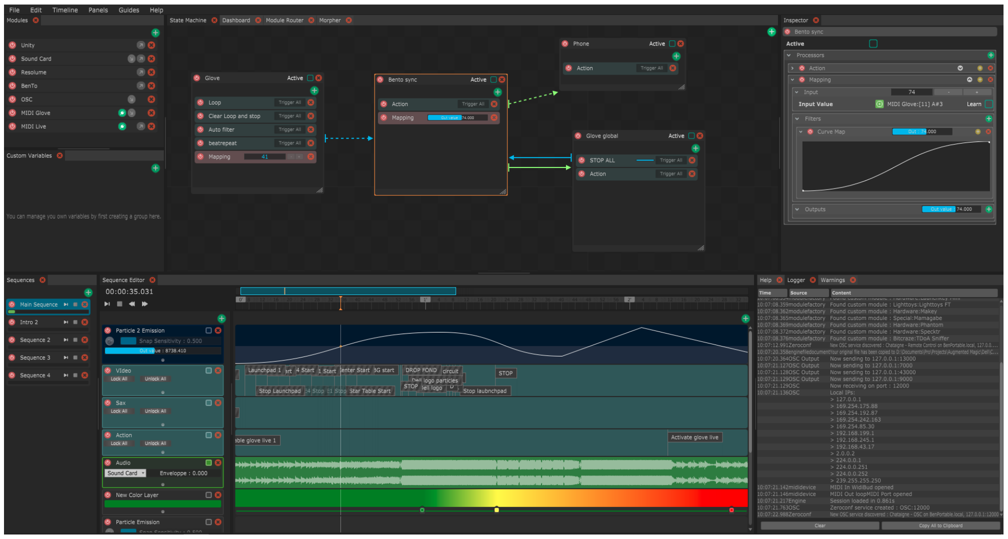


Figure 2.12. Chataigne: artist-friendly modular machine for art and technology.

Chataigne states can be seen as analogous to the previously mentioned Q-Live cues, although *states* are able to control a wider set of parameters from different devices on a network, rather than being limited to its own audio objects. Another core concept of Chataigne is its capacity to work as a time machine, which, as documented, "will let you create timeline-based sequences and create triggers and parameter animations over time"[37]. Chataigne can be easily integrated with libmapper, Pyo, Q-Live, Carla and so forth, thus allowing artists to benefit from the best of what each tool may offer.

[37]Taken from "The Amazing Chataigne Documentation" available at `https://bkuperberg.gitbook.io/chataigne-docs/the-time-machine-sequences/introduction-to-the-time-machine` (accessed on Nov 30, 2022).

2.2. Further comments

NASA cannot rely on software unless they have control over every line of code in it, and although musicians and artists don't usually shoot billions of dollars into space, they do need control over the tools they work with.

[de Valk, 2009]

This quotation by artist and researcher Marloes de Valk is particularly interesting here because it talks about *control.* It may seem contradictory to the title of this thesis, but in reality, allowing yourself to lose control, or delegating it to another entity (be it a musician, or a software), involves *trust.*

For instance, when I work with a performer in a collaborative setting, I expect a certain *openness* from her. I expect her to agree, disagree, argue, be open to changing her mind, but not to come with a *End-User License Agreement* (EULA) violation notice whenever I ask her to play her instrument in a weird way that she has never played before.

I also believe that she would not appreciate having to wait for the next "bug fixes" from the manufacturer of her instrument in order for it to be ready to play at the next concert. Nor does she want to wait for an approval from their legal department before she can replace a defective component. On the contrary, she certainly wants to deeply understand how her tool works, and to be able to choose the luthier of her preference to repair and adjust it to her liking, at the time she needs it. Or rather, she might want to have some fun repairing, modifying, improving or *augmenting* her instrument by herself, following her musical aspirations.

I also tend to agree with de Valk when she says that "there is a need for tools that match an artist's needs, instead of tools that match what an industry determines an artist needs.". Actually, it would be naive, or dishonest, to declare that in the domain of digital arts, all proprietary software is absolutely limiting. This is not true, as there are also closed source tools with a myriad of parameters and settings available. However, deprived of the source code, there will always be a boundary somewhere. And the moment the artist needs to cross this boundary to achieve an artistic goal, it will be very frustrating.

For example, in the pieces discussed in chapter 3, I present a situation in which I needed a feature that was supposed to be unavailable in a given Python library. I managed to reach its developer in a public mailing-list, and he pointed me to the source code where I could find what I was looking for, and what modifications I should perform to achieve my goal.

In another case, I was able to use an older version of an *EEG* device thanks to an old available library, not developed by the manufacturer, but by a third-party FLOSS developer who managed to learn the device's communication protocol. This library was deprecated, so again, thanks to its *openness* I was able to work it out by myself and discard the need for a newer device. Otherwise, not only would the company determine that I needed a new device, but also that the software to use would be that which they had a monopoly on developing.

These apparently minor benefits of using FLOSS end up making a huge difference in my creative process. Perhaps a negative point of all this is precisely the inherent fascination present in the DIY culture, which can often seduce us to the point of depriving us of making art in order to create all the components that are supposed to serve our artistic creation.[38]

[38]This discussion is well covered in the second part of the cited article by Marloes de Valk.

Chapter 3

Creation project: *Detaching*

Detaching is a series of three mixed pieces which explores the scenario of a new post-human ideology referred as *dataism*. This exploration, although being mostly thematic, embraces a concrete element of this philosophy: the disproportionate use of self-learning algorithms to accomplish tasks which were, not long ago, solely the domain of humans.

The first of three pieces, *Control (taking) / Control (losing)*, was conceived as an interactive installation. It is structured in 2 parts and does not require a dedicated human performer. This algorithm-oriented experiment presents an uncensored cooperation between a machine and a human brain. *Control (taking) / Control (losing)* makes extensive use of AI for both musical and textual output. A non-invasive *brain-computer interface* (BCI) is used to translate brainwaves into sound.

The second piece, *You're thinking I'm a good person*, was written for two performers, electronics, and *Disklavier*. It uses texts generated by AI and modern *Text-to-Speech* (TTS) techniques in order to directly affect the performance, intending to subvert the sense of a well-known collection of empathic statements.

In the third piece of the series, the performer (a human pianist) engages in a troubled mix of fighting and cooperation with the machine (a prepared *Disklavier* controlled by algorithms) in order to keep control of his music. This piece is structured in 3 scenes and depends on human improvisation skills, both in music and in life, that, as the performance proposes, have become increasingly scarce.

In the next lines, before delving into the pieces, I describe how and why each of the afore mentioned elements (AI, TTS, Disklavier, BCI etc) has served this experiment.

Worth noting that, given the restrictions related to the COVID-19 pandemic, the three pieces presented here could not be live-performed for the general public, and were therefore recorded in studio. The available media files are listed in Appendix A.

3.1. Disklavier

The three pieces of this series make use of a player piano. The one I had access to for the recording sessions was a *Yamaha Disklavier*, which is today the most widespread brand of self-playing pianos. Such instruments were originally based on a pneumatic mechanism which read music from perforated paper or metallic rolls. Currently, a player piano by Yamaha is equipped with electronic sensors for recording and electromechanical solenoids for the playback feature. The musical information is manipulated throughout a Standard MIDI File (SMF). Some models, such as *Mark IV*, are built on an embedded Linux and run on a FLOSS-based system, maintained by an independent programmer.[39]

Modern self-playing pianos have been widely used in music education and research.[40] Not surprisingly, composers have also explored the artistic possibilities of such instruments.

Given that the most notable feature of a Disklavier lies in its capacity to play "impossible music" (or, music unplayable by a human pianist), they are, in the words of Disklavier composer Hans Tammen, "often used to present the 'superhuman' capabilities of these machines". Tammen recorded in 2015 the album "The Choking Disklavier", in which the piano, "remotely controlled by the composer/performer, [...] produces constantly rumbled and

[39]`http://dkvbrowser.sourceforge.net/` (accessed on 26 April, 2022)

[40]At least by those able to afford one. Although a second-hand old model Disklavier can be found for less than $30,000 USD, a newer version might easily be prohibitive for some. As of April 2022, a DC3X ENPRO is priced around $90,000 USD. The cost can go further: a special model released in 2000 had a retail price of $333,000 USD (`https://www.cbc.ca/news/science/yamaha-shows-off-333-000-piano-of-the-future-1.218381`, accessed on 26 April, 2022)

crackled noises [...]".[41] Other creative uses of a Disklavier have been documented, for instance in "The Editing and Arrangement of Conlon Nancarrow's Studies for Disklavier and Synthesizers" [Willey, 2014], and the experience of an improviser "playing *on* and *with* the system at the same time" [Dahlstedt, 2014].

3.1.1. The role of the player piano in *Detaching*

Technically speaking, a player piano is an *automaton*. In other words, "a moving mechanical device made in imitation of a human being [...], that performs a function according to a predetermined set of coded instructions, especially one capable of a range of programmed responses to different circumstances" [Hobson, 2001]. As such, the player piano fits remarkably well the subject I happened to explore in my creation. My approach towards such a machine, though similar to Tammen's (given that I too emphasize the richness of noise produced by its internal mechanisms), is mostly focused on the level of autonomy granted to the algorithms controlling it.

In the next sections I present the algorithms that controls the Disklavier in my work, and how they interact with the audience in *Control (taking) / Control (losing)*, with a duo of performers in *You're thinking I'm a good person*, and finally with a pianist in *You can't un-neighbor your neighbor*.

3.2. A few words about AI terminology

Understanding and classifying the technical elements of an AI system is not straightforward. Given the fast and ongoing evolution in this domain, assimilating its many layers of concepts and acronyms (such as NLP, ML, DL, ANN, RNN, AGI and so forth) can be overwhelming. The correct use of these terms is a challenge by itself, even for experts. Thus, in the next lines I attempt to explain a few concepts which I see as necessary (and sufficient) to understand the creation process that will be discussed later in this thesis.

[41] `https://tammen.org/CD-Choking-Disklavier` (accessed 26 April 2022)

Firstly, we can safely consider *Artificial Intelligence* as the broad term relating to this discussion. Although at this point in history AI no longer needs an introduction, it is nonetheless important to keep in mind that there are many different forms of *learning* that can be applied to it. One of these forms is based on *learning through experience*; and one of the possible methods to achieve this is using the *machine learning* (ML) method.

In summary, ML algorithms build models by processing sample data, also known as *training data*, in order to make predictions. From this point of view, ML is considered a subset of AI. However, ML is itself a broad subject and can make use of many different approaches. Currently, the dominant approach used in ML is called *deep learning* (DL), which is based on *artificial neural networks* (ANNs).

For instance, one of the models that I trained is based on a *long short-term memory* (LSTM) algorithm. LSTM is used to extend its Recurrent Neural Network (RNN) algorithm, which is a special type of ANN. ANN is the core component of its DL algorithm, which is a subset of the broader concept of ML. Ultimately, in this work a particular ML method was used to make a computer perform a task that is usually carried out by humans (composing music) – all these concepts considered, the result can (perhaps) be called AI.

In addition to this theoretical terminology, there is also a range of terms related to the implementation of those algorithms, which includes programming languages, libraries, frameworks, datasets, etc. Just as one should not expect a composer to master all the technical bits behind the scenes (which I myself do not), for the purpose of appreciating my creation process, it might be enough to keep the following in mind:

- The tools that I developed for my pieces were built using Python as a programming language, including many of its libraries, with emphasis on Pyo.
- In my pieces I used ML techniques to generate music (MIDI data) and text.
- For text generation, I *fine-tuned* a pre-trained model from *OpenAI* called *Generative Pre-trained Transformer 2* (GPT-2).

- For music generation, I *trained* a few different models using the *Performance RNN* algorithms from the *Magenta* project.
- Both GPT-2 and Performance-RNN make use of a Python library called *Tensorflow*, an "end-to-end open source platform for machine learning"[42], therefore are easily integrated into my own Python code.

3.3. Composing with Magenta

Magenta is distributed as an open source Python library, powered by TensorFlow. This library includes utilities for manipulating source data (primarily music and images), using this data to train machine learning models, and finally generating new content from these models.

[Waite et al., 2016]

My first question when facing such a complex system for generating musical sequences was: what kind of *musical elements* can I make use from this intricate framework? Ultimately, these models generate nothing more than MIDI data. Therefore, tone color was not a trivial choice. Also, most of the available models have been developed with traditional melody/harmony generation in mind – and I was certainly not looking for structured melodies or harmonies for my pieces. Furthermore, Magenta models do not understand musical form. To make things harder, I quickly realized that silence is virtually absent from the generated pieces. Finally, I found that texture and expressivity were the only elements that, once enriched by some piano preparation, I could seriously consider for the kind of music I was willing to create.

Currently, there are eighteen models available in Magenta's git repository.[43] For the present work I tested a few of them, such as *ImprovRNN*, *MelodyRNN* and *PolyphonyRNN*. As their names suggest, they are supposed to generate melodies (*MelodyRNN*), improvise melodies over a series of chords (*ImprovRNN*) and produce polyphonic music, mostly focused

[42]`https://www.tensorflow.org/` (accessed 26 April 2022)

[43]Models for creative work, such as painting, music score and audio synthesis: `https://github.com/magenta/magenta/tree/main/magenta/models` (accessed 26 April 2022)

on counterpoint (*PolyphonyRNN*). Unhappy with those (they did not provide the level of texture and expressivity that I was looking for), I decided to try out a model called *PerformanceRNN*, a "LSTM-based recurrent neural network designed to model polyphonic music with expressive timing and dynamics".[44] Fortunately, with this model I was able to mimic gestural expressions in a much more interesting way than the others could offer.

PerformanceRNN is capable of generating polyphonic performances with more natural timing and dynamics compared to other Magenta models. It produces a standard MIDI file output, replacing explicit durations with *note-on* and *note-off* events, as explained by its developers: "In order to support expressive timing, the model controls the clock with time-shift events that move forward at increments of 10 ms, up to 1 second. All note-on and note-off events happen at the current time as determined by all previous time shifts in the event sequence" [Simon and Oore, 2017].

This means that, to take advantage of the full potential of this model, I would need a MIDI dataset in which note timings were based on human performance, rather than being extracted from a score. After some research, I found an article introducing the *GiantMIDI-Piano*, a "large-scale MIDI dataset for classical piano music". [Kong et al., 2020]. This dataset contains 10,854 unique piano pieces by 2,789 composers. It has been transcribed using a high-resolution piano transcription system and is considered the largest piano MIDI dataset produced to date [Kong et al., 2020.]. Although not publicly available, the dataset was kindly provided by its authors through a simple email request. I then started training the *PerformanceRNN* model using different subsets of the *GiantMIDI-Piano MIDI dataset.*

Training a ***PerformanceRNN*** model

The following steps summarize how someone with little knowledge of machine learning concepts and techniques can still train their own Magenta model and generate interesting MIDI material for their pieces.

[44]`https://magenta.tensorflow.org/performance-rnn` (accessed 26 April 2022)

PerformanceRNN provides the composer with command line tools (or *scripts*) in order to simplify the basic tasks. For further flexibility, programming proficient composers might want to make direct use of the Python library instead. General instructions can be found at the Magenta developers' repository [Simon and Oore, 2017]. My custom code is available in a public git repository [Vaz, 2021].

For one of the pieces to be discussed later, I used a subset of 147 works by J.S. Bach. In addition, to generate AI sequences supposed to sound like "glitchy Bach", I was hoping to produce musical material presenting lower dynamics and pitch range. My intention was to create contrasting sequences to those generated from most of the available subsets, where original performances come from more modern music featuring full piano power.

The first step for training such a model is to convert the collection of MIDI files into *NoteSequences* which, as the name implies, are just sequences of notes taken from the original musical material. *NoteSequences* provide a serialized structured data in binary form, so that they become smaller and faster to be processed than a regular XML format. These NoteSequences are stored in a single *TFrecord* file, the one used by *Tensorflow* – the open source library performing the machine learning tasks behind Magenta.

The command bellow will process as input the MIDI files located in `bach_midi_dataset` directory in order to produce the TFrecord file `note_sequences_bach.tfrecord`:

```
$ convert_dir_to_note_sequences \
        --input_dir=./bach_midi_dataset \
        --output_file=./note_sequences_bach.tfrecord \
        --recursive
```

The next step is to extract sequences from NoteSequences and save them as *SequenceExamples*. At this point, two collections of SequenceExamples will be created. These are used to be fed into the model for training and evaluation. They contain a sequence of inputs and labels that represent, in this instance, the piano track. By default, 90% of the sequences are generated for training and 10% for evaluation. This ratio can be redefined by adjusting the *eval_ratio* parameter. Similarly to NoteSequences, SequenceExamples are also stored in TFrecord files (named `training_performances.tfrecord` and `eval_performances.tfrecord`).

The lines to generate SequenceExamples are the following:

```
$ performance_rnn_create_dataset \
        --config=multiconditioned_performance_with_dynamics \
        --input=note_sequences_bach.tfrecord \
        --output_dir=./training/bach \
        --eval_ratio=0.10
```

The training/evaluation process uses the previously generated SequenceExamples files as input. This step can take days to reach a good level of accuracy, depending on the available computer resources. For instance, using an old Nvidia GTX 1070ti GPU, it took me about 72 hours to complete a training job using 147 MIDI files with accuracy > 0.71 and loss < 0.79. These are just a few of many metrics that can be monitored during the process, using for instance the *TensorBoard* tool.[45]

Training can be performed through the following command, which will execute a 3000-step task:

```
$ performance_rnn_train \
        --config=multiconditioned_performance_with_dynamics \
        --run_dir=./checkpoints/bach/logdir/run1 \
        --sequence_example_file=./training/bach/training_performances.tfrecord
        --num_training_steps=3000
```

After training a model, one is ready to generate musical material. At this point the algorithm becomes the core creator of musical sequences (from now on I will refer to such sequences as *performances-rnn*, or *performance-rnn* in its singular form). Nevertheless, we are still able to partially influence the results. For instance, given that this model was trained in a *multi-conditional* configuration (as set by the `-config=multiconditioned_performance_with_dynamics`), it can generate

[45]`https://www.tensorflow.org/tensorboard/` (accessed 26 April 2022)

performances-rnn conditioned to a desired note density and pitch class distribution. Furthermore, *PerformanceRNN* models can receive a list of pitches, a melody, or a MIDI file as a "priming track". In other words, we can provide our AI system with a musical phrase and tell it to continue the performance; then we ask the system to limit the generated notes to a set of X pitches, where notes are played Y times per second, for a given duration Z.

Another interesting input parameter supported by this model is called "temperature", which determines the level of "creativity" permitted to the machine. The higher the temperature, the more a performance-rnn sounds "random" – and so increase the chances to be surprised. For instance, the following command generates three (line 4) 2-minute performances-rnn (line 5), where notes are played, on average, 5 times per second (line 6), mostly limited to the G major scale (line 7), with a level of "creativity" of 0.7 (line 8):

```
1 $ python3 custom_generator.py \
2         --output_dir=/tmp/ \
3         --config=multiconditioned_performance_with_dynamics \
4         --num_outputs=3 \
5         --num_steps=12000 \
6         --notes_per_second=5 \
7         --pitch_class_histogram=’[1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1]’ \
8         --temperature=0.7 \
9         --run_dir=./checkpoints/bach/logdir/run1
```

The above command also specifies the previously generated checkpoint dir as input (line 9). This contains the necessary data for generating performances (checkpoint, metagraph, and metadata about the model). However, we might want to replace such complex structure with a *bundle file*, which is a convenient way of combining the model's data into a single file for greater portability and simplicity. As an example, the command bellow creates the bundle file `performance_rnn_bach.mag`. This file can be called using the `-bundle_file` parameter rather than `-run_dir` the next time we want to generate a performance-rnn.

```
$ python3 custom_generator.py \
        --config=multiconditioned_performance_with_dynamics \
        --run_dir=./checkpoints/bach/logdir/run1 \
        --bundle_file=./performance_rnn_bach.mag \
        --save_generator_bundle
```

By following these steps as described, we can generate performances-rnn that last as long as we want, by setting the `num_steps` parameter. An important constraint is that, as already mentioned, Magenta models will not learn musical form. There is no beginning, no preparations, no ending. Motifs are not developed. Sections, variations and modulations are nonexistent. Such free-form structure might still be musically expressive and end up being an inspiring source of musical material for the creation process. It is however extremely limiting in the case where we want, to a great extent, to avoid human manipulation of the final musical product – which was my goal for this project.

To get around this limitation, I looked for an option to modify a few parameters *during* the performance-rnn generation process, rather than *before* it. By asking the model developer about such a feature, I realized that it is already there, though undocumented.[46] This means that, surprisingly, *PerformanceRNN* allows density and pitch changes over time. This hidden feature allowed me to give some form to the generated music, which was especially useful in the first piece of the series, *Control (taking) / Control (losing)*.

As a final and more comprehensive example, the command bellow generates 3 performances-rnn.

```
$ python3 custom_generator.py \
--output_dir=/tmp/ \
--config=multiconditioned_performance_with_dynamics \
--num_outputs=3 \
--num_steps=30000 \
--notes_per_second='[1, 3, 3, 5, 1]' \
--primer_melody='[60, -2, 60, -2, 67, -2, 67, -2]'
--pitch_class_histogram='[[0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0], \
                          [0, 0, 0, 0, 0, 1, 0, 3, 0, 1, 0, 0], \
                          [0, 1, 0, 1, 0, 1, 0, 2, 0, 1, 0, 1], \
                          [0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0], \
                          [0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0]]' \
--temperature=1.2 \
--bundle_file=./performance_rnn_debussy.mag
```

[46] `https://groups.google.com/a/tensorflow.org/g/magenta-discuss/c/aEhG_auKVPE/m/ENpVvuP4DQAJ` (accessed 26 April 2022)

3.4. AI-driven text production with GPT-2

The first two pieces of *Detaching* make use of an AI-based text-generation model called *Generative Pre-trained Transformer 2* (GPT-2), which was fine-tuned for the purpose of providing textual material matching the respective themes of the pieces.

GPT-2 is a language model composed of 1.5 billion parameters, trained on a dataset of more than 8 million web pages. "GPT-2 uses deep learning to translate text, answer questions, summarize passages, and generate human-readable text output on a level that is often indistinguishable from that of humans. It is a general-purpose learner [...]" [Radford et al., 2019.].

Just like *PerformanceRNN*, GPT-2 is a deep learning model, though implementing a *Transformer* architecture rather than recurrence/convolution-based models (such as the previously mentioned RNN).

Unlike Performance RNN, GPT-2 did not require me to perform a training from scratch. *OpenAI*, the company behind GPT-2, has released four pre-trained models to date: the "small" 124M parameter model, the "medium" 355M model, the "large" 774M model and finally the "extra-large" 1558M model. They use an amount of 500MB, 1.5GB, 3GB and 12GB of disk, respectively. These are much larger than the Magenta's *PerformanceRNN* models that I trained to produce MIDI data. The initial GPT-2 training was performed by OpenAI itself using a huge corpus based on linked *Reddit*[47] posts in the English language that had received at least 3 *upvotes* before December 2017.

Thus, we are not required to *train* a GPT-2 model from our own dataset in order to generate text with a near-perfect grammatical syntax and punctuation. But we can benefit from the possibility of *finetuning* those existing models on custom datasets[48]. This allows us to generate style-oriented or thematic output such as poetry, song lyrics, news, novels, etc. Such a feature is made possible thanks to a *fork* of original GPT-2 implementation by

[47]`https://reddit.com` (accessed 26 April 2022)

[48]Although I prefer to use the term *finetuning*, some people may refer to this process as *retraining* or even – certainly inaccurate – *training*.

an independent FLOSS developer.[49] Additionally, on the top of this fork, a library called *gpt-2-simple*[50] makes the whole process simpler, delivering extra features, such as the ability to provide a prefix to specify how you want your text to start (a feature similar to the *priming track* from *PerformanceRNN*, and particularly used in *You're thinking I'm a good person*). Again, the complicated machine learning code behind all these layers is provided by the Tensorflow project.

Among the three pre-trained GPT-2 models available, the "small" 124M and the "medium" 355M models can be easily finetuned in a consumer machine equipped with a powerful GPU. The larger ones require GPU power which is usually not available in personal computers. Looking for faster results, I performed this step using a remote computer instance by *Google Colab*[51] instead. The free versions of *colab* instances offer either a *Nvidia T4 GPU* or a *Nvidia K80 GPU*, either one being faster than anything a composer could normally afford for such an experiment.

The full-version programs that I coded to generate the texts in my pieces are available in a public git repository [Vaz, 2021]. They are extensive and composed of several auxiliary functions less important for the context of this text. The code below, however, built from excerpts of my main code, summarize the steps from finetuning to generating text using the "medium" 355M GPT-2 model through the *gpt-2-simple* Python library.

[49]`https://github.com/nshepperd/gpt-2` (accessed 26 April 2022)
[50]`https://github.com/minimaxir/gpt-2-simple` (accessed 26 April 2022)
[51]`https://colab.research.google.com/` (accessed 26 April 2022)

```python
import gpt_2_simple as gpt2

FINETUNE = False
GENERATE = True

if FINETUNE:
    # Download a light model in case there is none
    model_name = "124M"
    if not os.path.isdir(os.path.join("models", model_name)):
        gpt2.download_gpt2(model_name=model_name)

    # Sample text for finetuning
    file_name = "source/Ten-Arguments-For-Deleting-Your-Social-Media-Accounts-Right-Now-by-Jaron-Lanier.txt"

    # Start gpt2 session
    sess = gpt2.start_tf_sess()

    # Finetune the model
    gpt2.finetune(sess, dataset=file_name, restore_from='fresh', \
              print_every=10, sample_every=200, save_every=500, \
              model_name=model_name, steps=3000)

if GENERATE:
    # (Re)start gpt2 session
    sess = gpt2.start_tf_sess()

    # GPT2 generator
    gpt2.load_gpt2(sess, run_name=vars.GPT2_RUN_NAME, checkpoint_dir=vars.GPT2_CHECKPOINT_DIR)

    # Setting random parameters
    GPT2_TEMPERATURE = random.uniform(1, 1)
    GPT2_TOP_K = random.randint(0, 0)
    GPT2_TOP_P = random.uniform(0.9, 0.9)

    # Setting a prefix
    prefix="Our destiny is "

    output_text = gpt2.generate(sess,
                                run_name=vars.GPT2_RUN_NAME,
                                return_as_list=True,
                                length=vars.GPT2_LENGTH,
                                temperature=GPT2_TEMPERATURE,
                                nsamples=vars.GPT2_N_SAMPLES,
                                batch_size=vars.GPT2_BATCH_SIZE,
                                prefix=prefix,
                                include_prefix=vars.GPT2_INCLUDE_PREFIX,
                                top_k=GPT2_TOP_K,
                                top_p=GPT2_TOP_P,
                                )[0]
```

Figure 3.1. Excerpts of text generation with GPT-2.

3.5. The generative aspect of *Detaching*

But the day came, when I, myself, invented confusion! (I could not get a copyright on it, because somebody else had invented it a few thousand years earlier.)

Herbert Brün [Brün, 1961]

Before delving into the analysis of the pieces, it is worth giving some thought to how this work relates to the notion of *generative music*, a term that has gained great popularity among contemporary composers over the last years.

It might be that most of the time I dedicate to my composition is devoted to designing systems to make music. Such systems comprise a series of principles and procedures that will be later adjusted for my musical intentions. Once the whole setup is loaded, the system runs by itself. This process is not new in music creation, as we can find exactly the same concept being explored back in 1965, in a mechanical phasing system built for the piece "It's Gonna Rain" by Steve Reich. It is also seen in "Music for Airports" (1978) by Brian Eno – who coined the term *generative music* in the 90s.

These two pieces are known for using generative *methods*, yet they do not retain their generative nature over time. Or, whenever performed, they sound the same. Nonetheless, generative music may also retain its generative nature in a performance, so that we could see them as a *generative product.* [Parviainen, 2021] Two notable examples of this concept are present in "Available Forms" (1961) by Earle Brown and "In C" (1964), by Terry Riley, both featuring a form of generative social system where the composer trust the spontaneity and the awareness of performers to the extent of making each performance sounding completely different from any other.

For most of my creation I made use of these two approaches mentioned above, by designing generative methods which will often produce a generative product. Randomness – a convenient and effective generative technique – is also regularly used in the three pieces of *Detaching*, being it produced from algorithms or from human improvisation; generated in a more or in a less constrained form.

3.6. Analysis of *Control (taking) / Control (losing)*

Control (taking) / Control (losing) is a mixed work presented as an interactive installation. It is structured in two sections. In this piece, I was in particular seeking to explore a generative system from two different perspectives. For the first section, *Control (taking)*, I adopted a generative method to obtain a musical result which does not retain its generative nature. In other words, it does not present any substantial variation with each new performance – or, every time you listen to this piece, you hear (almost) the same result. The "almost" here is due to the fact that *Control (taking)* only finishes when a human brain, connected through a *Brain-Computer Interface* (BCI), reaches a certain threshold of "attention".

For the second section, *Control (losing)*, I wanted the algorithms to be previously fed by brainwaves, so that the system generates the musical material required for the piece's execution. Like in *In C* (as discussed in section 3.5), this section takes on a new form every time it is performed. However, unlike *In C*, the path it takes is not a result of human will, but comes from disorderly brainwaves and a sophisticated neural network controlled by algorithms.

The generative system applied to both sections is based on Magenta *PerformanceRNN* models. The BCI adopted for the recordings was the *MindWave Mobile 2*, an affordable EEG (electroencephalograph) headset from NeuroSky. *Control (losing)* also makes use of a deep neural network system for text generation, the Generative Pre-trained Transformer 2 (GPT-2), from OpenAI.

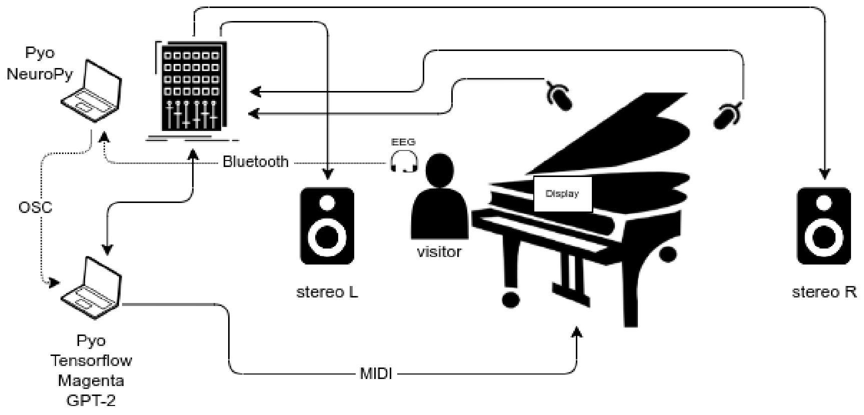


Figure 3.2. Stage organization of *Control (taking) / Control (losing)*.

3.6.1. Technical notes

In *Control (taking) / Control (losing)* I wanted to use biological signals to create a generative system which we have little control over. At first I considered monitoring vital signals such as body temperature, heart rate and blood pressure. A few sensors and a *single-board computer* (SBC) were used in the very beginning of this creation, in a mixed music seminary offered by the Faculté de musique of the Université de Montréal. At that time I was also able to access a BCI, the *MindWave Mobile 2*, produced by NeuroSky.[52]

MindWave Mobile 2 is a lightweight EEG headset able to detect brainwaves and send the collected data to another system. Connection is possible via bluetooth, which makes it easier to communicate with standard computer devices, single board computers, and microcontroller units.

[52]`https://store.neurosky.com/pages/mindwave` (accessed 26 April 2022)

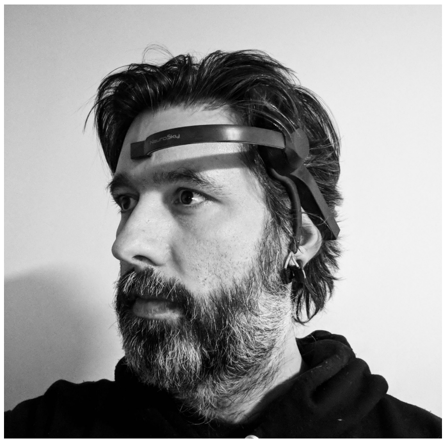


Figure 3.3. Wearing the MindWave Mobile 2, an EEG headset from NeuroSky.

The device can report five types of brainwave from different parts of the brain: Gamma waves (oscillating from 24 Hz to 40 Hz, related to states of high attention and concentration), Beta waves (between 12 Hz and 30 Hz, associated with normal waking consciousness), Alpha waves (8 Hz to 12 Hz, relaxed but conscious), Theta waves (lower frequency, between 4 Hz and 7Hz, associated with dreaming) and Delta waves (high amplitude, low frequency waves, associated with a state of dreamless sleep cycles). In addition to the 12-bit raw-brainwaves, MindWave Mobile 2 also calculates levels of *Attention* and *Meditation* using a NeuroSky proprietary *eSense* meter, in a 0-100 range. [Morshad et al., 2020]

Integrating MindWave Mobile 2 into my system was a bit of a challenge. NeuroSky only provides closed source and stand-alone software for this product. Nonetheless, I was able to find a third-party Python library distributed under a free license, called NeuroPy[53]. However, the available code is deprecated, which required me to perform a few changes in order to run it over the most recent Python releases. The modified working version of this library is distributed in the same git repository as the source code and scores of my work.

Once I was able to get data from the device, I used Pyo's OSC features in order to convert, scale and transfer the processed data to my own piece of software, as illustrated in the following code:

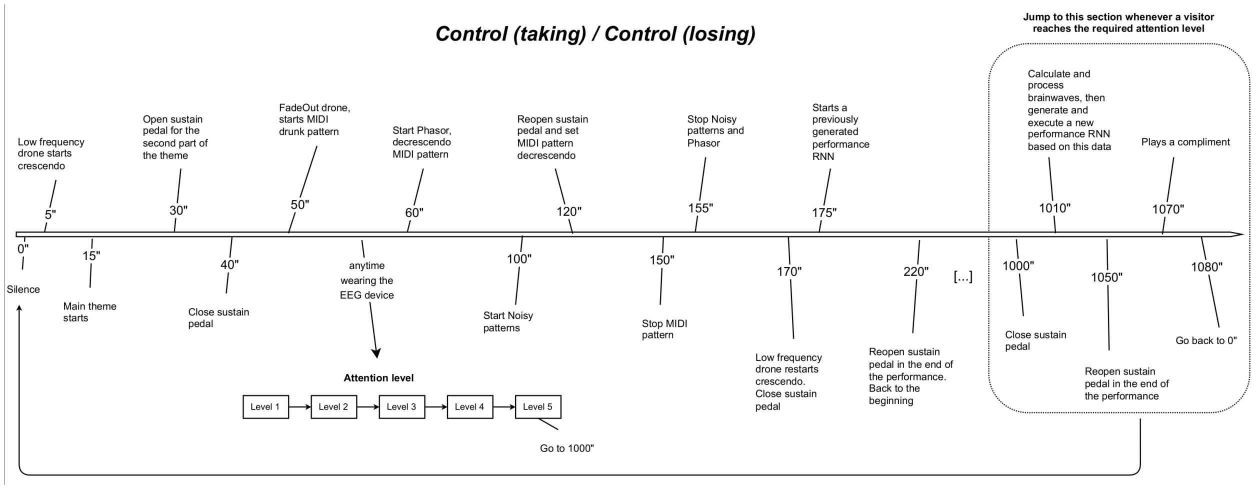
[53]`https://pythonhosted.org/NeuroPy/` (accessed 26 April 2022)

```python
from pyo import *
from NeuroPy import NeuroPy
import time

# localhost for testing purpose
OSC_RECEIVER = '127.0.0.1'

# Pyo initial setup
s = Server(audio='jack').boot()

# EEG initial setup
bw = NeuroPy("/dev/rfcomm0")
bw.start()

# Convert to audio stream
bw_input = SigTo([bw.attention, bw.meditation, bw.poorSignal])

# Send OSC data to receiver
bw_stream = OscSend(bw_input, port=10001, address=['/attention', '/meditation', '/poorSignal'], host=OSC_RECEIVER)

# Start Pyo audio server
s.start()

# Main loop
while True:
# Update data values
bw_input.setValue([bw.attention, bw.meditation, bw.poorSignal])
time.sleep(0.01)
```

Figure 3.4. NeuroPy and Pyo integration.

3.6.2. Installation workflow

In section 1, the Disklavier plays a *quasi*-deterministic algorithm-generated music. It starts with a recurrent theme extracted by a selected *performance-rnn*. Visitors are encouraged to sit on the piano bench, wear the EEG device and read the “music sheet” which is placed on the music desk. The music sheet here is a series of live-generated texts by another AI system, projected in a digital display. In the case where nobody wears the device, the system will complete its cycle of musical events. Such events include variations of simple patterns, a recurring theme, sounds generated via synthesis, and short periods of silence. Also, at a given moment, the system randomly chooses one of the previously generated *performances-rnn*. Thus, each cycle will present visitors with something new, while retaining its form. A graphic notation that represents this cycle is presented in Figure 3.5.



Control (taking) / Control (losing)
Jump to this section whenever a visitor reaches the required attention level
Low frequency drone starts crescendo
5"
Open sustain pedal for the second part of the theme
30"
FadeOut drone, starts MIDI drunk pattern
50"
Start Phasor, decrescendo MIDI pattern
60"
Reopen sustain pedal and set MIDI pattern decrescendo
120"
Stop Noisy patterns and Phasor
155"
Starts a previously generated performance RNN
175"
Calculate and process brainwaves, then generate and execute a new performance RNN based on this data
1010"
Plays a compliment
1070"
0"
Silence
15"
Main theme starts
40"
Close sustain pedal
anytime wearing the EEG device
Attention level
Level 1
Level 2
Level 3
Level 4
Level 5
Go to 1000"
100"
Start Noisy patterns
150"
Stop MIDI pattern
170"
Low frequency drone restarts crescendo. Close sustain pedal
220"
Reopen sustain pedal in the end of the performance. Back to the beginning
[...]
1000"
Close sustain pedal
1050"
Reopen sustain pedal in the end of the performance
1080"
Go back to 0"

Figure 3.5. Graphic timeline of *Control (taking) / Control (losing).*

Anytime in the piece, once the first brainwaves are processed, the music being played is immediately affected: brainwaves now take control of a few musical parameters. Additionally, a TTS-generated voice asks the visitor to keep their attention at a high level, and as a result a new piece will be created and performed to her. The attention level must be consistently kept for a little while. At this point, a few stress factors may arise, making focusing harder. One of them comes from the AI-driven texts themselves, which, by lacking a clear direction (due to their imperfect AI creator) can easily undermine the visitor's attention.

In the case that the visitor gives up, the system is reset and the Disklavier starts over the piece from the beginning. In the case that the visitor succeeds, the system congratulates them and starts generating a new music based on the collected data. Technically speaking, for about 60 seconds, the algorithm collects their brainwaves data and transfers the average of attention level to a *PerformanceRNN* generator. There is an audible feedback during the feeding period, reflecting the visitor's levels of attention in real-time. This element should, as intended, disturb their pursuit.

Once a performance-rnn is generated, the visitor is asked to leave the piano and to listen to their *brain music* right away. When the music ends, the system returns to its initial state (section 1) and is ready to host the next visitor. Each new AI-driven performance is added to the queue, becoming one of the available performances to be randomly chosen in section 1.

3.6.3. Recording

To date, a recording using Disklavier has not been possible for this piece. However, I was able to make an audio recording in conjunction with a video at the time of using the EEG device. The piano samples used do not accurately represent the expected musical result, since for this piece I prepare the piano, making it sound more like a percussion ensemble.

The recorded audio, video, and each Python script used in this piece are described in table Table 3.1.

File	**Description**
`AUDIO.wav`	Home-recorded audio demonstrating one cycle of the piece.
`VIDEO.mkv`	Home-recorded video from the same demonstration above, highlighting the moment when I wear the EEG device and reach the attention level required.
`GPT2_daemon.py`	This script calls GPT-2 algorithms in a loop and generates new lines of text to be displayed.
`display.py`	This script displays in fullscreen the texts being generated by “`GPT2_daemon.py`.
`Control_EEG_daemon.py`	This script connects to the EEG device. It collects the brain-waves data and sends them to the main program via OSC, over a network. The IP address of the OSC receiver should be set at the beginning of the code.
`Control_part1.py`	This script opens a Pyo GUI which gives you the option to start the piece. It is the main code for this piece, and will call other scripts such as `custom_generator.py` which uses Magenta to generate the new performance-rnn files.
`carla_patch.carxp`	A XML file containing the configuration of audio objects and MIDI assignments. This file is to be used by Carla audio plugin host software.

Table 3.1. *Control (taking) / Control (losing)*: description of recordings and script files

All the resources needed to run the piece are available in the project's git repository[54], including the fine-tuned GPT-2 model for text generation and the trained Magenta models for generating the performances-rnn in MIDI format.

3.7. Analysis of *You're thinking I'm a good person*

There will be an inevitable apathy alternating with short periods of resistance, leading the actors towards mostly robotic movements – although still trying to find some human sense (and control) within their micro-universe of gestures and sounds [...]

excerpt from the score

You're thinking I'm a good person is a 15-minute piece for 2 instruments, a Disklavier and live electronics. The theme of this piece is related to *social awareness*, more specifically to its most well-known feature: *empathy*. Here, two opposing currents are confronted against each other by algorithms in order to inspire the composition and the performance itself.

A set of 44 empathetic statements[55] which have been extensively publicized on self-improvement websites is given as input to a modern TTS engine provided by *Google's Wavenet Text*.

[54] `https://github.com/tiagovaz/detaching`

[55] *44 Empathy Statements That Will Make you the Greatest Listener* at `https://www.couples-thrive.com/blog/couples-therapy-fort-lauderdale/44-empathy-statements-that-will-make-you-the-greatest-listener/` (accessed 26 April 2022)

Figure 3.6. A recorded performance of *You're thinking I'm a good person.*

The audio output of each statement is randomly picked by an algorithm to be played during a few parts of the piece, seeking to encourage unplanned reactions from the performers. Such reactions – which are musical reactions – require some promptitude and improvisation skills from the musicians.

The same set of 44 empathetic statements was also used to fine-tune a GPT-2 model, so that a few algorithm-elaborated paragraphs are played (also using the TTS engine) during the performance. Some of the generated paragraphs are also directly used in the recording, as illustrated in Figure 3.10.

By fine-tuning the GPT-2 model with texts related to *empathy*, my goal was to generate semantically correct, yet confusing/contradictory paragraphs as output. For this purpose I used a diverse collection of sources, including supportive content as well critical and provocative material, such as the entire book "Against Empathy: The Case for Rational Compassion" from psychologist and writer Paul Bloom.

The 44 empathetic statements and the gpt2-generated texts are listed in the score (see Appendix C). The stage is organized as illustrated in Figure 3.7.

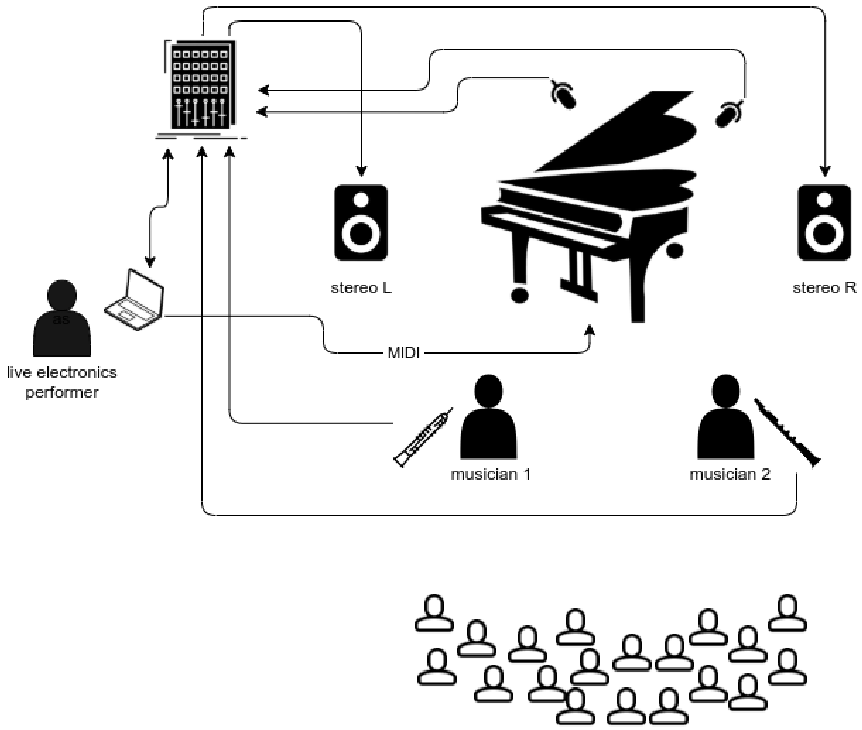


Figure 3.7. Stage organization of *You're thinking I'm a good person.*

You're thinking I'm a good person is structured in 3 scenes. Technically, each scene is controlled by a computer code to be sequentially triggered by the live electronics performer[56], as discussed in the next sections.

3.7.1. Scene 1

Performers are guided to stand side-by-side in front of the Disklavier. They remain quiet; no reaction is expected. A synthetic voice introduces the piece by proclaiming the *gpt2-generated text 1*, as follows:

> *Empathy works by increasing the activation of your empathic awareness in others, which literally increases the effort of positive empathy work on their part. Thus, if you activate empathy near others, empathically speaking, this will help them feel more closely and empathically attached to you. It shouldn't be surprising then, that empathy interactions also have a direct relationship to empathy intensity.*

[56]The *live-electronics performer* role in all pieces mentioned in this work was played by myself.

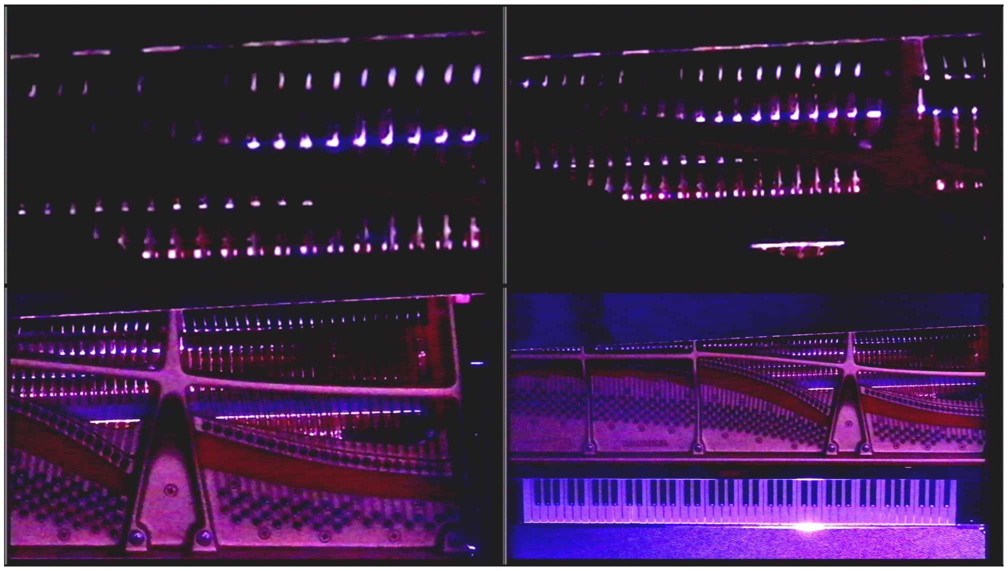


Figure 3.8. "You're thinking I'm a good person", Disklavier plays live algorithmic music in scene 1.

Right after, the Disklavier starts playing a musical progression generated by an algorithm performing in real-time. The only human reaction expected here comes from the live electronics performer, who manipulates the effects by following a *crescendo*, as described in the score. Figure 3.9 illustrates the actual notation reflecting this scene.

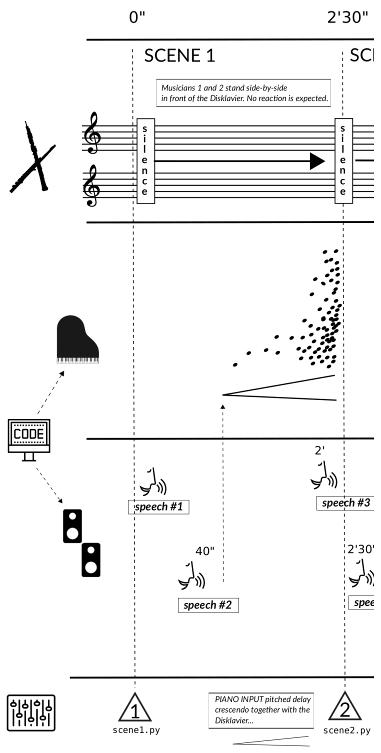

0"
2'30"
SCENE 1
SC
Musicians 1 and 2 stand side-by-side in front of the Disklavier. No reaction is expected.
silence
silence
CODE
2'
speech #1
speech #3
40"
2'30"
speech #2
spee
PIANO INPUT pitched delay crescendo together with the Disklavier...
1
scene1.py
2
scene2.py

Figure 3.9. Graphic notation of Scene 1 in *You're thinking I'm a good person.*

3.7.2. Scene 2

After the first scene has ended, and given some silence, *gpt2-generated text 2* is recited, followed by a reprise of *gpt2-generated text 1*, repeatedly. This time, however, the first continuous sound arises and the voice is suddenly interrupted by the piano. From now on, each time the piano plays a new note, it triggers a synthetic continuous sound of a slightly different frequency, leading to noticeable beatings. Such sounding experience produces a cloudy, yet contrasting effect, intended to – metaphorically – present the idea of a blurry line between the natural and artificial worlds.

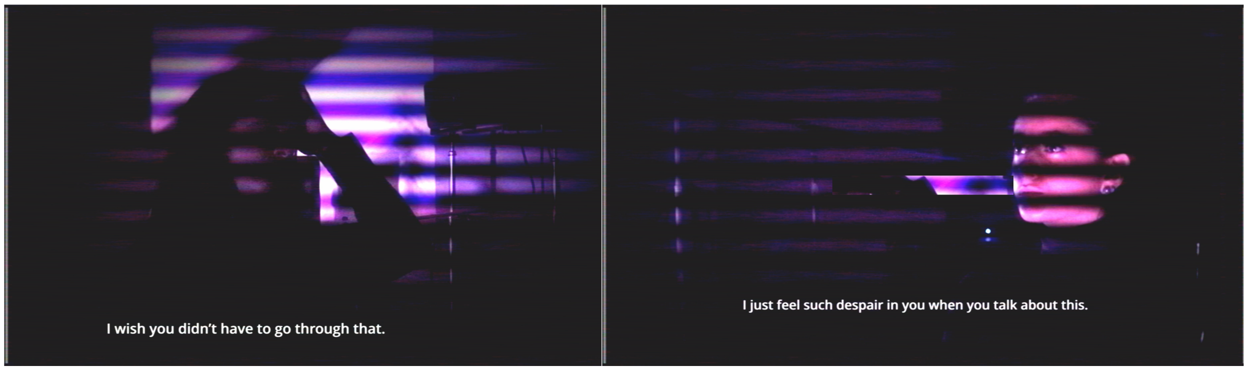

I wish you didn't have to go through that.
I just feel such despair in you when you talk about this.

Figure 3.10. *You're thinking I'm a good person*, randomly picked empathic statements rule the performance.

During this entire scene, performers are asked to musically react to the piano (or, the Disklavier controlled by algorithms), as well as to the voiced statements. Once this scene reaches its second half, all three performers have very little control of what lies ahead. The oral statements are randomly picked overtime by the algorithm, building new contexts, telling new stories, and therefore encouraging new reactions and original interpretations whenever a new performance takes place. The musicians' unpredictable reactions to this game are a substantial part of the piece.

Scene 2 ends with *gpt2-generated text 3* over the empathic statement 34 as a prefix input to the algorithm, intended to motivate an infuriated reaction. The graphic score for this scene is illustrated in Figure 3.11.

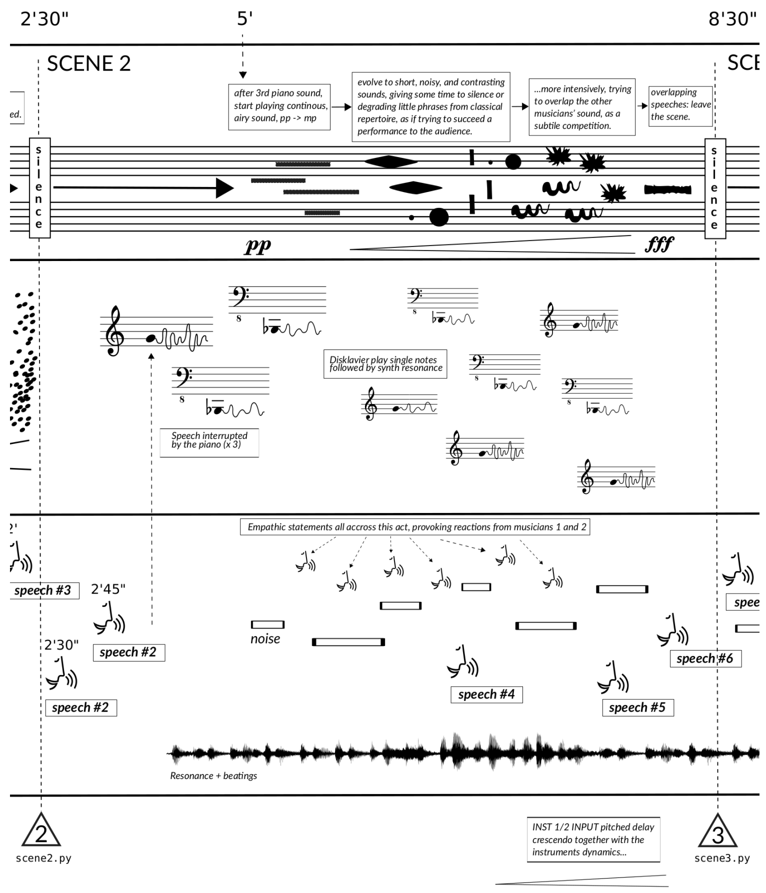

2'30"
5'
8'30"
SCENE 2
after 3rd piano sound, start playing continous, airy sound, pp -> mp
evolve to short, noisy, and contrasting sounds, giving some time to silence or degrading little phrases from classical repertoire, as if trying to succeed a performance to the audience.
...more intensively, trying to overlap the other musicians' sound, as a subtile competition.
overlapping speeches: leave the scene.
silence
pp
fff
Disklavier play single notes followed by synth resonance
Speech interrupted by the piano (x 3)
Empathic statements all accross this act, provoking reactions from musicians 1 and 2
speech #3
2'45"
speech #2
2'30"
speech #2
noise
speech #4
speech #5
speech #6
Resonance + beatings
INST 1/2 INPUT pitched delay crescendo together with the instruments dynamics...
scene2.py
scene3.py

Figure 3.11. Graphic notation of Scene 2 in *You're thinking I'm a good person.*

3.7.3. Scene 3

In Scene 3, for the first time musicians establish a dialogue. So far, they have been reacting to the sounds generated by the piano, and to the voiced statements. Now they face each other and play a short piece together. This time the piano plays aside, totally ignored, left on the second plan.

However, the potential human interaction is now disturbed by the music sheet placed in between. Performers apathetically focus on their score rather than on the produced sound. A set of effects are triggered and intensified, which make performers get lost in their music. The effects were programmed to break tone and rhythm. The musicians then realize they can just stop performing; but still the music continues playing on its own for a few minutes.

A final speech is then triggered, reciting the last gpt2-generated text, again using statement 34 as input:

> *I see. Let me summarize: What you're thinking here is that I'm not so bad. No. You're thinking I'm a shit-stain. You're thinking I'm a terrible person. You're thinking I'm a creep. You're thinking I'm a monster. You're thinking I'm a fucking lunatic. You're thinking I'm a dumbass. No. You're thinking I'm a good person. You're thinking I'm a good person. You're thinking I'm a good person.*

The graphic notation for Scene 3 is illustrated in Figure 3.12.

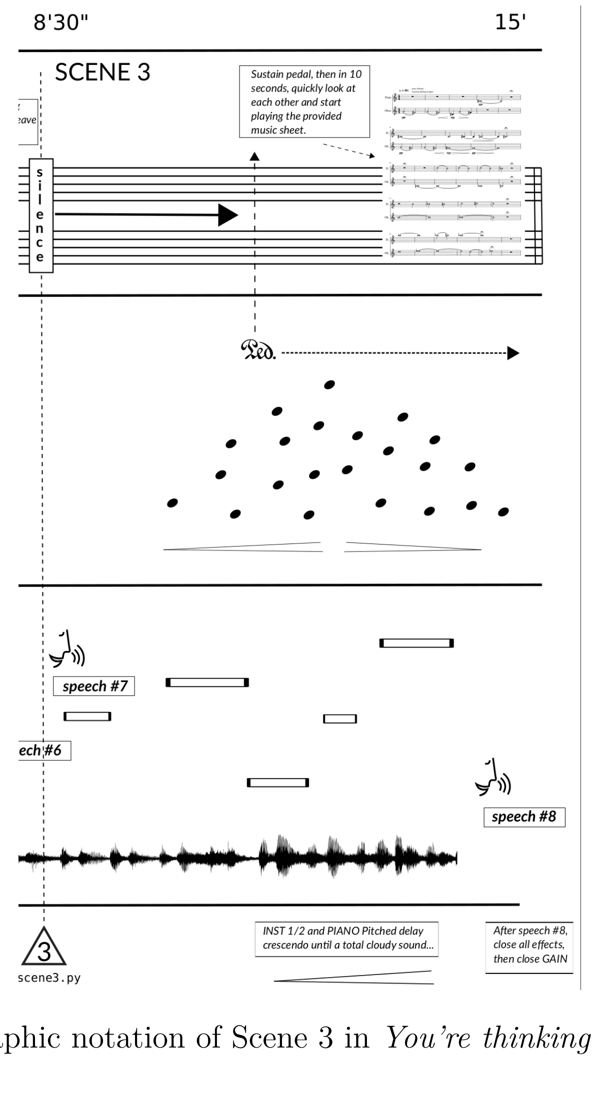

8'30"
15'
SCENE 3
Sustain pedal, then in 10 seconds, quickly look at each other and start playing the provided music sheet.
silence
speech #7
speech #6
speech #8
INST 1/2 and PIANO Pitched delay crescendo until a total cloudy sound...
After speech #8, close all effects, then close GAIN
3
scene3.py

Figure 3.12. Graphic notation of Scene 3 in *You're thinking I'm a good person.*

3.7.4. Rehearsals and recording

A performance of "You're thinking I'm a good person" was recorded in studio, and a video has been produced aiming to visually reflect the artistic intentions. This process happened during a residence with the *Ensemble de musique contemporaine* (EMC) at the Faculté de musique of the Université de Montréal between October and December 2020. Due to the pandemic situation at that time, a live performance with an audience was not an option.

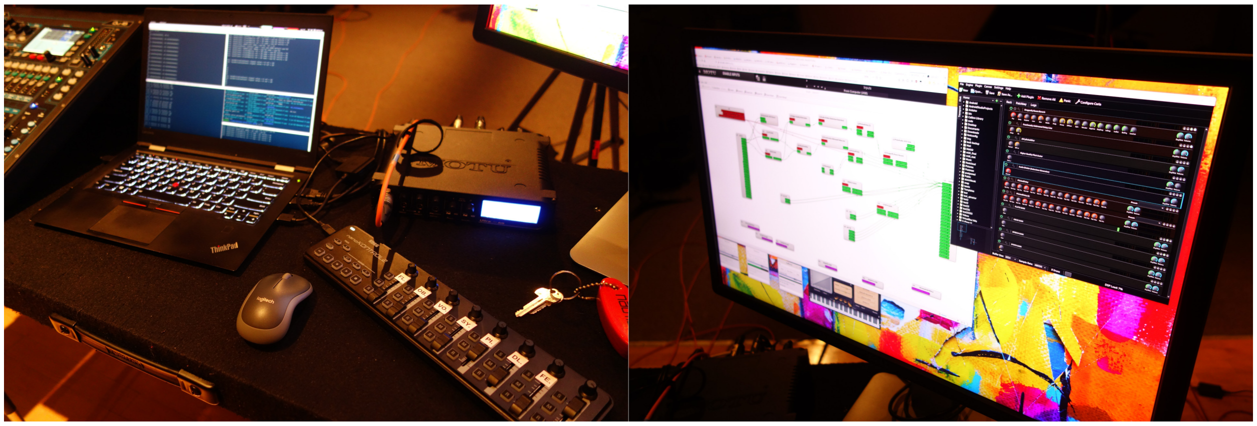


Figure 3.13. Live electronics setup using Carla, Jack, Pyo and a MIDI controller in "You're thinking I'm a good person".

Two skilled musicians were appointed to work with me on this piece: Louise Gagnac (flute) and Mélissa Tremblay (oboe). I was not willing to have control over the instrumentation. In fact, a few days preceding the first meeting I still did not know which instruments would be available, as I intentionally asked in the residence application. At the time I had just requested some sounding qualities and an extra motivation on the part of the musicians to join a collaborative composition process:

> *This piece is highly flexible in terms of instrumentation, as it is strongly based on a continuous sound approach together with some theatrical performance. [...] Instruments for this piece should be capable of producing a continuous sound in a great range with very little attack (ex.: using techniques such as circular breathing, continuous bowing, or sustained by digital effects). [...] Performers*

will be encouraged to share responsibilities and collaborate with the composer in terms of musical ideas, notation, scenography etc. The piece has many improvised sections. Performers with some interest in free improvisation will certainly be a better match to this experience.

In the end, working with a flute/oboe duo for this piece was an enlightening experience, mostly due to the fact that the musicians were mainly classically trained and had little to inexistent experience with free improvisation, mixed music, instrument amplification, and acting out on stage. Such circumstances pushed me to improve my communication skills with performers – after certainly failing in the first encounters! Also, I had no choice but to work more efficiently in the rehearsals, given the time constraints usually present in a traditional residence format.

3.8. Analysis of *You can't un-neighbor your neighbor*

Most personalized filters are based on a three-step model. First, you figure out who people are and what they like. Then, you provide them with content and services that best fit them. Finally, you tune to get the fit just right. Your identity shapes your media. There's just one flaw in this logic: Media also shape identity. And as a result, these services may end up creating a good fit between you and your media by changing... you.

[Pariser, 2011]

The above quotation relates to a phenomenon called "filter bubble", which has been directly or indirectly affecting us all for more than a decade now. [57] Such experience has inspired the theme of this piece, in which I intended to musically represent a social outcome through a scenario in which we have lost our most basic conflict resolution skills. There is no direct character representation in the performance, although Parisier's three-step model had influenced the resulting macro structure.

The title comes from a subject-related discussion between historian Yuval Harari and Facebook CEO Mark Zuckerberg. This conversation is part of a series of "public discussions

[57]Filter bubble: `https://fs.blog/filter-bubbles/` (accessed 26 April 2022)

about the future of technology in society", featured by Zuckerberg himself. The video is publicly available on Harari's Youtube channel.[58]

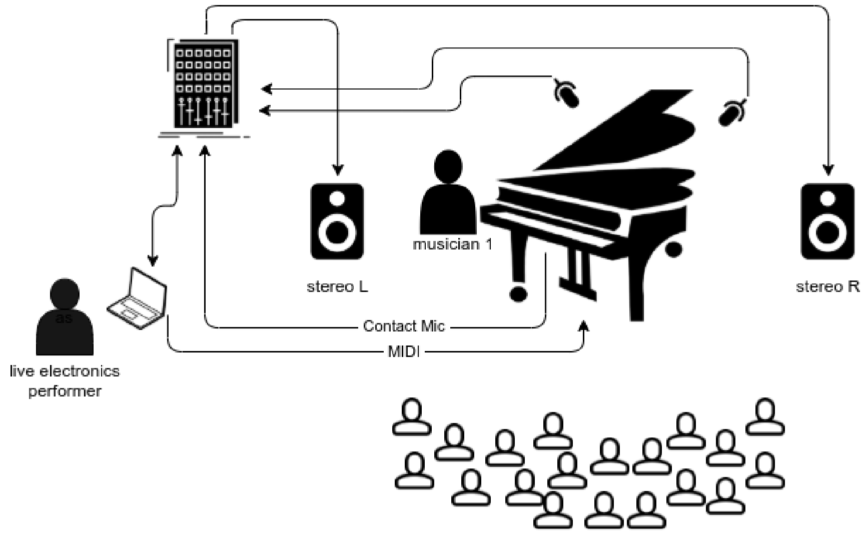


Figure 3.14. Stage organization of *You can't un-neighbor your neighbor.*

The resulting collaborative work ended up as a 15-minute piece composed in 3 scenes, which requires a prepared Disklavier, a computer, a pair of speakers, a pianist, and a live electronics performer. The piano is simultaneously controlled by a live-generated *performance-rnn* and by a live electronics performer manipulating a set of algorithms. The pianist follows a minimal set of instructions for preparing and playing the piano, together with the algorithms, all throughout the performance. The parameters used by the AI system to generate the performances-rnn are presented in table Table 3.2. Figure 3.14 shows how the stage should be set for this piece.

3.8.1. Rehearsals and recording

You can't un-neighbor your neighbor was composed in collaboration with pianist Daniel Áñez, an experienced improviser and open-minded musician based in Montreal. He

[58]Mark Zuckerberg & Yuval Noah Harari in Conversation (Apr 27, 2019): `https://www.youtube.com/watch?v=Boj9eD0Wug8` (accessed 26 April 2022)

	Scene 1	Scene 2	Scene 3
Magenta model	PerformanceRNN	PerformanceRNN	PerformanceRNN
Model type	MPD	MPD	MPD
Input composer(s)	Claude Debussy	J. S. Bach	Claude Debussy
N input performances	29	147	29
Input prime	–	BWV802	–
Training steps	23785	303140	23785
Accuracy	0.7370561	0.71957767	0.7370561
Loss	0.78055185	0.790744	0.78055185
Training time	8.3h	72.4h	8.3h
Output pitch distribution	[0, 1, 0, 0, 2, 1, 0, 1, 1, 0, 1, 1]	[10, 1, 3, 10, 15, 1, 10, 10, 1, 10, 2, 10]	[0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0]
Output temperature	1.4	1.3	0.7
Output notes per second	5	5	20
MIDI duration (num_steps)	120min	300min	120min

Table 3.2. *PerformanceRNN* models with respective parameters as used in the recording of *You can't un-neighbor your neighbor*.

accepted the challenge of musically facing such a self-playing machine surrounded by unexpected events. Endowed with greater time flexibility, we could dedicate more time to sharing impressions about the piece's background and intentions.

This piece is intended to depend on a reduced set of instructions. The final score was supposed to be produced throughout the creation process, as a product of our observations. During the rehearsal sessions I worked with Daniel using a back-and-forth approach, where each performance became itself the object for further discussion and the generation of new (or removal of) instructions for the next.

I coded my own Python tool to control sequence-time events while using Carla as the audio effects host. For the live electronics performer, a similar workflow was designed to be followed all across the piece: just trigger the provided Python code and then manipulate an interface in order to control the effects and the MIDI data sent to the Disklavier. This setup is visually represented in Figure 3.15.

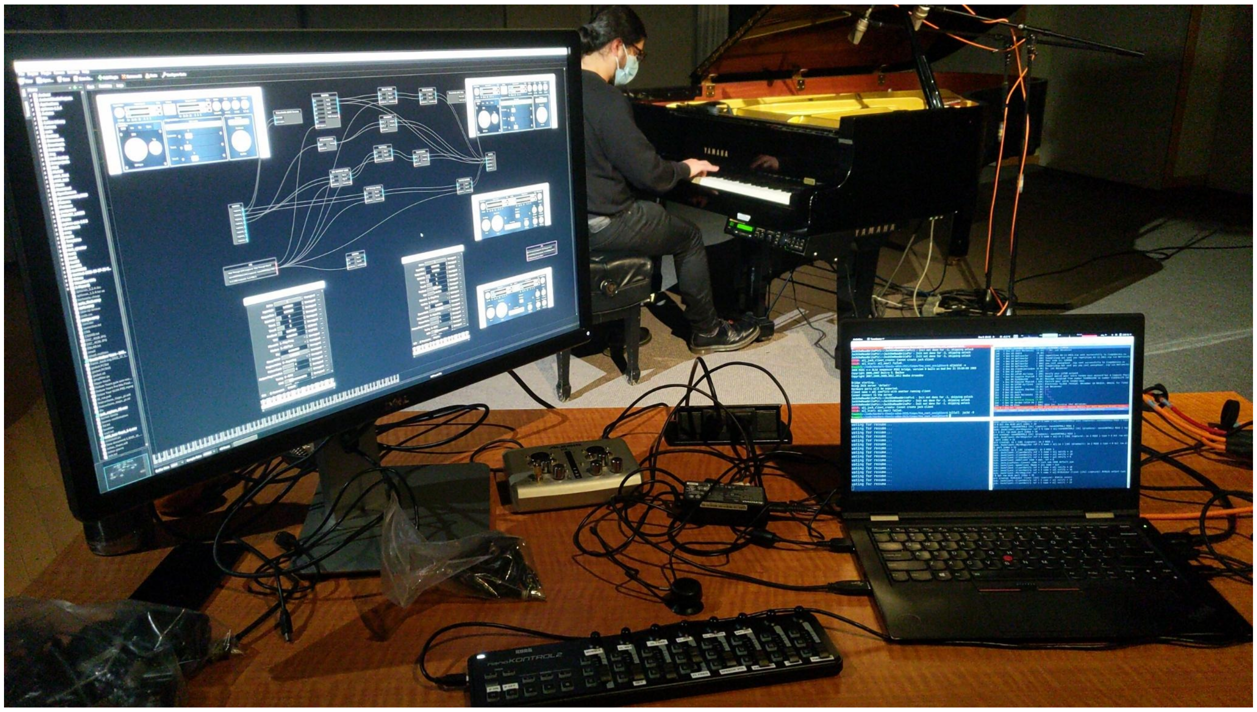


Figure 3.15. Live electronics setup using Carla, Pyo and a MIDI controller in *You can't un-neighbor your neighbor.*

My intention was also to observe our own reactions when facing such a lack of authority over the creation attempts. Further, we were supposed to avoid as much as possible our natural desire for control. With that in mind, Daniel was told at first to freely play, either over the keys or inside the instrument. He was also encouraged to prepare the piano at his will while a performance-rnn was autonomously running. This was basically the first set of instructions provided.

Evidently, his musical interactions should happen with our thematic discussions in mind, contrasting acceptance to confrontation, conviction to confusion, and so on. As expected, after a few sessions we felt the need to apply further control. Although we agreed we had succeeded in producing some interesting musical gestures along with the algorithms, the music clearly lacked form, dynamics, and definitely needed some rest.

Concerns started to be raised by Daniel after each new attempt: "Would you be able to stop the algorithm for a while?", "Can I have a similar performance being played each time so that I can better predict how to react?", "Can I use the sustain pedal to manipulate a bit what's being played?", "Can we be positioned in a way we can look at each other in order to develop some gestures together?", and so on.

Gradually we were agreeing to increase our authority, but no more than necessary. "Necessary" here meaning something close to a blurry imaginary space where we tried to shape the minimal requirements to reach a product that we could call *music*. In light of these considerations, although this piece remained to a great extent conducted by algorithms, both the pianist and the live electronics performer were provided with a fair share of control over the resulting sounds.

A recording session took place in December 2021 and a 30 minute video has been released, featuring two takes for each of the three scenes. This format gives the audience a better idea about what is fixed and what is actually improvised in this piece.

3.8.2. Scene 1

first, you figure out who people are and what they like

This scene starts with a performance-rnn generated from a model trained against a dataset of 29 pieces from Claude Debussy. It plays on average 5 notes per second, has a *temperature* of 1.4 – being the most "creative" among the three scenes in that sense – and follows a pitch distribution represented as `[0, 1, 0, 0, 2, 1, 0, 1, 1, 0, 1, 1]` (or $C \times 0$, $C\sharp \times 1$, $D \times 0$, $D\sharp \times 0$, $E \times 2$, $F \times 1$, $F\sharp \times 0$, $G \times 1$, $G\sharp \times 1$, $A \times 0$, $A\sharp \times 1$, $B \times 1$).

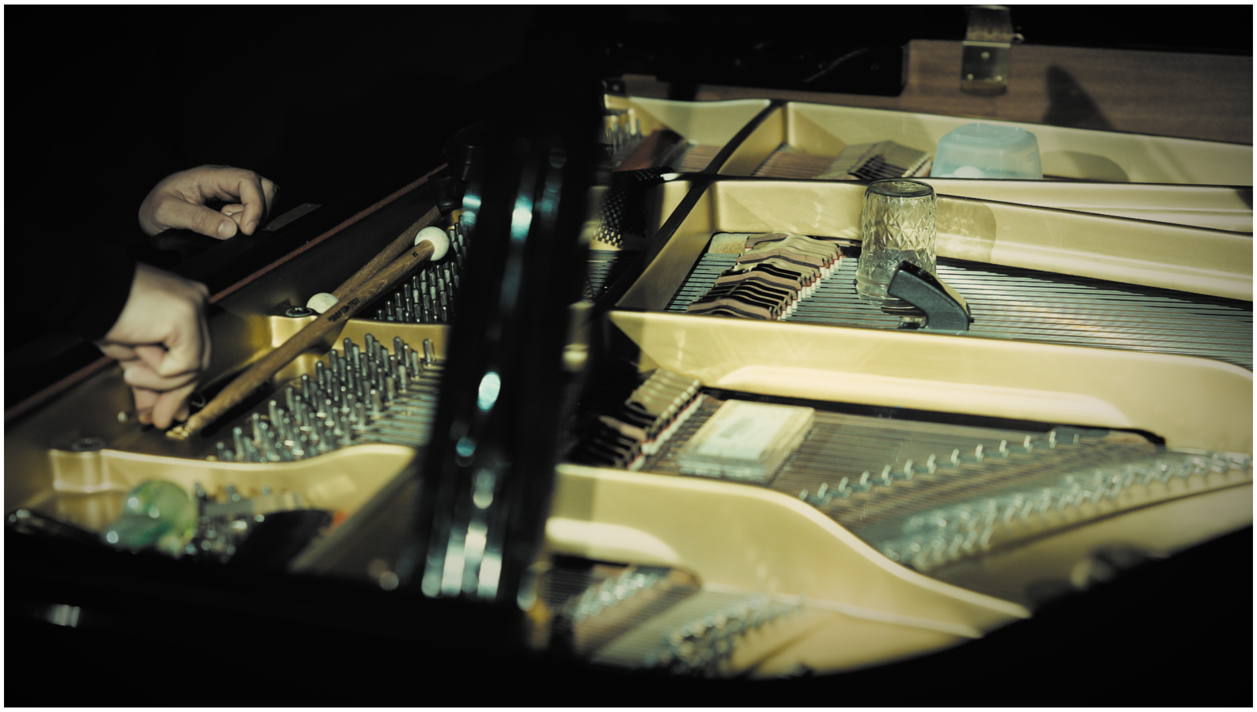


Figure 3.16. Prepared piano in "You can't un-neighbor your neighbor".

The MIDI data is close to being entirely generated by AI. The live electronics performer focuses on the manipulation of a chain of effects instead. As intended, the pianist follows a minimal set of instructions that later became the score (see Appendix 3). Those instructions are mostly written in a descriptive manner and leave room for a good deal of improvisation.

In *Take 1* of Scene 1, the pianist is asked to avoid direct manipulation over the keys in order to privilege the internal acoustic features of the instrument. In the second take, the same intentions should be represented by instead using chords along with the algorithm. Instructions to gradually move towards a continuous sound were given, reached through a pair of *eBows* over middle-range strings.

3.8.3. Scene 2

...then, you provide people with content and services that best fit them

The performance-rnn which guides this scene was generated by a model trained against 147 pieces of J. S. Bach. Like Scene 1, it plays on average 5 notes per second, and has a little less “creativity”, with *temperature* set to 1.3. Its pitch distribution is represented as `[10, 1, 3, 10, 15, 1, 10, 10, 1, 10, 2, 10]`. Unlike the other scenes, the performance-rnn for Scene 2 is additionally fed with with a primer, which is the theme of *Duetto in E minor* (BWV 802). This theme is then first played while the pianist just observes, getting ready to react a few bars later.

Figure 3.17. Scene 2 of “You can’t un-neighbor your neighbor”.

In Scene 2, most of MIDI data sent to the Disklavier is manipulated by the live electronics performer, who triggers and controls algorithms that generate pseudo-random motives. The excerpt in Figure 3.18 shows, for instance, how the performer generates and sends musical motives to the instrument. Here, pitches are chosen through a noise distribution ("drunk" with looped segments) object where the maximum step is defined by a continuous *Phasor* stream (lines 1 and 2). A MIDI controller knob defines min and max values for *velocity* (lines 11 and 15), while another knob controls the *tempo* of the events (line 19). Random generators are used all over the code in order to further "humanize" the outcome (see Appendix D for the full program). An example of this output can be seen in [00:09:28] where the pianist develops a dialogue in *crescendo* with the generated motives.

```
 1 ph = Phasor(.4)
 2 pitch = XnoiseMidi(12, freq=8, x1=1, x2=ph, scale=0, mrange=(21, 108))
 3
 4 count = 0
 5
 6 def midi_event():
 7     if pattern_play == True:
 8         global count
 9         pit = int(pitch.get())
10         if count == 0:
11             vel = random.randint(int(ctl_pattern_velocity.get()), \\
12                                  int(ctl_pattern_velocity.get()) + 30)
13             dur = 500
14         else:
15             vel = random.randint(int(ctl_pattern_velocity.get()), \\
16                                  int(ctl_pattern_velocity.get()) + 10)
17             dur = 125
18         count = (count + 1) % random.randint(2,5)
19         pat.setTime(float(ctl_pattern_time.get()))
20         s.makenote(pitch=pit, velocity=vel, duration=dur)
21
22 pat = Pattern(midi_event, time=0.125).play()
```

Figure 3.18. MIDI pattern with Pyo.

3.8.4. Scene 3

finally, you tune to get the fit just right

This scene, certainly the most aggressive among the three, is based on performances-rnn generated *in extremis*, comprising an unusual pitch distribution of `[0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0]` at 20 notes per second on average. The level of creativity is lower than seen in previous scenes (0.7). This means that notes marked with low probability are more unlikely to be played, as one can verify in [00:20:00]. Also, for Scene 3 I use a higher number of effects, as well as an extra incitement material for the performance, in the form of audible excerpts extracted from the discussion which inspired the piece (as described in section 3.8).

The pianist was instructed to be influenced by – and musically match – the discussion, by making use of repetitive patterns and extrapolating the dynamics in his gestures. The musical outcome of this can be heard, for instance, in [00:30:15].

Conclusions

Through the creation of *Detaching*, I aimed to better understand the roles and boundaries between actors (composer, coder, performer, listener and algorithms) participating in a human-machine collaborative work. Consequently, a significant portion of my work was dedicated to balance this fluid *social* system, by shifting authority from myself to performers, from performs to algorithms, from algorithms to listeners and so forth; and by leaving artistic decisions unmade, thus constantly giving up control – then struggling not to claim it back. Such a cyclical adventure, even though quite challenging at times, has turned out to be a fascinating *generative* exercise by itself.

Regarding the music notation present in my pieces, I have tried to merge several techniques (graphic score, descriptive instructions, and technical documentation) in an attempt to clearly communicate my intentions. The scores of *Detaching* include all the material necessary to execute the performances (i.e. source code, media files, trained models, etc). I also provide the effects *patches* in a *XML* scheme (generated by *Carla*) so that the audio objects and chains could be reproduced using any other same-purpose software – given the human/computer-readability aspect of this format. How outdated will these instructions be in a few years? Hard to predict. However, I see the use of FLOSS and open standards as a good starting point to avoid obsolescence – or at least to delay it.

I would like to highlight one reservation regarding the reproduction of this work: in a few occasions I mentioned affordability as a feature of FLOSS. However, in its current format, *Detaching* depends on a mechanical device which portrays the antithesis of affordability. As discussed in section 3.1, a player piano such as a Disklavier is hardly accessible to musicians without institutional ties. My hope is – and this is a true motivation for future work – that this series of pieces evolves to become a more accessible project, not only on the software side, but also with regard to hardware. This would imply in keeping its core concepts, but moving from a player piano to a more accessible DIY instrument. Thanks to the use of open standards, I am confident that this work could easily accommodate other musical devices, such as a hacked piano[59], or even a hacked piano toy, i.e. based on cheap microcontrollers, solenoids and some FLOSS.[60]

On a less technical note, this work has given me the opportunity to reflect on larger and future-facing questions concerning AI and its implications, not only in the art world, but in the future of our societies. In this sense, I feel lucky to have encountered authors of much-needed clarity, capable of stripping away the mystical layers that tend to surround our imagination on this subject. Some of them being brave enough to face head on and defy the discourses imposed by the big AI industry. Surrounded by so many captivating readings, at certain moments it became hard to open an editor and write computer code (which pushed me to learn a little more about discipline).

Having finished this step of my work, many questions, like those listed in chapter 1, are probing my thoughts and imagination. In the end, I have the impression that, although no big answer has calmed my spirit, I have perhaps succeed to de-dramatize the questions I originally had, giving some room for others to derive from them.

Ultimately, I hope that the resulting material from this work, whether in the form of code, thesis, image, or sound, has managed to capture the essence of this somewhat unmanageable experiment.

[59] Here is a well documented 700$ DIY Disklavier build by the young maker Brandon Switzer, based on Arduino: `https://brandonswitzer.squarespace.com/player-piano` (accessed 26 April 2022)

[60] An example of a hacked piano toy which simulates a player piano using MIDI: `https://hackaday.com/2019/08/13/itty-bitty-midi-piano-sings-with-solenoids/` (accessed 26 April 2022)

Bibliography

[Boden 2010] Boden, Margaret A.: The Turing test and artistic creativity. In: *Kybernetes* 39 (2010), Mai, Nr. 3, S. 409–413. – URL `https://doi.org/10.1108/03684921011036132`

[Brün 1961] Brün, Herbert: *Music and Existentialism.* 1961. – URL `https://sites.evergreen.edu/arunchandra/`. – (accessed on Aug 26, 2021)

[Chamberlain 2017] Chamberlain, Alan: Are the Robots Coming? Designing with Autonomy; Control for Musical Creativity; Performance. In: *Proceedings of the 12th International Audio Mostly Conference on Augmented and Participatory Sound and Music Experiences.* New York, NY, USA : Association for Computing Machinery, 2017 (AM '17). – URL `https://doi.org/10.1145/3123514.3123568`. – ISBN 9781450353731

[Coeckelbergh 2017] Coeckelbergh, Mark: Can Machines Create Art? In: *Philosophy & Technology* 30 (2017), Sep, Nr. 3, S. 285–303. – URL `https://doi.org/10.1007/s13347-016-0231-5`. – ISSN 2210-5441

[Couldry and Mejias 2019] Couldry, Nick ; Mejias, Ulises A.: Making data colonialism liveable: how might data's social order be regulated? In: *Internet Policy Review* 8 (2019), Juni, Nr. 2. – URL `https://doi.org/10.14763/2019.2.1411`

[Cuthbert 2021] Cuthbert, Michael S.: *music21: a Toolkit for Computer-Aided Musicology.* `https://web.mit.edu/music21/`. 08 2021. – (accessed on Aug 26, 2021)

[Dahlstedt 2014] Dahlstedt, Palle: Circle Squared and Circle Keys Performing on and with an Unstable Live Algorithm for the Disklavier. In: Caramiaux, Baptiste (Hrsg.) ; Tahiroglu, Koray (Hrsg.) ; Fiebrink, Rebecca (Hrsg.) ; Tanaka, Atau (Hrsg.): *14th*

International Conference on New Interfaces for Musical Expression, NIME 2014, London, United Kingdom, June 30 - July 4, 2014, nime.org, 2014, S. 114–117. – URL http://www.nime.org/proceedings/2014/nime2014_534.pdf

[Daniele et al. 2021] Daniele, Antonio ; Di Bernardi Luft, Caroline ; Bryan-Kinns, Nick: "What Is Human?" A Turing Test for Artistic Creativity. In: Romero, Juan (Hrsg.) ; Martins, Tiago (Hrsg.) ; Rodríguez-Fernández, Nereida (Hrsg.): *Artificial Intelligence in Music, Sound, Art and Design*. Cham : Springer International Publishing, 2021, S. 396–411. – ISBN 978-3-030-72914-1

[Goossens et al. 1994] Goossens, M. ; Mittelbach, F. ; Samarin, A.: *The LaTeX companion*. New-York : Addison-Wesley, 1994

[Gorz 2010] Gorz, André: *The Immaterial*. University of Chicago Press, Febrero 2010 (University of Chicago Press Economics Books 9781906497613). – URL https://ideas.repec.org/b/ucp/bkecon/9781906497613.html. – ISBN ARRAY(0x46896880)

[Harari 2019] Harari, Y.N.: *21 Lessons for the 21st Century*. Random House Publishing Group, 2019. – URL https://books.google.ca/books?id=MSKEDwAAQBAJ. – ISBN 9780593132814

[Harari et al. 2018] Harari, Yuval N. ; Harari, Yuval N. ; Harari, Yuval N.: *Homo deus: a brief history of tomorrow*. Harper Perennial, 2018

[Harari 2018] Harari, Yuval N.: *Why Technology Favors Tyranny*. 2018. – URL https://www.theatlantic.com/magazine/archive/2018/10/yuval-noah-harari-technology-tyranny/568330/. – (accessed on Dec 01, 2022)

[Hobson 2001] Hobson, Archie: *The Oxford dictionary of difficult words*. Oxford , Eng.: : Oxford University Press, 2001

[Hong and Curran 2019] Hong, Joo-Wha ; Curran, Nathaniel M.: Artificial Intelligence, Artists, and Art: Attitudes Toward Artwork Produced by Humans vs. Artificial Intelligence. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 15 (2019), Juli, Nr. 2s. – URL https://doi.org/10.1145/3326337. – ISSN 1551-6857

[Jackson 2017] Jackson, Tony E.: Imitative Identity, Imitative Art, and "AI: Artificial Intelligence". In: *Mosaic: An Interdisciplinary Critical Journal* 50 (2017), Nr. 2, S. 47–63. – URL `http://www.jstor.org/stable/45158927`. – ISSN 00271276, 19255683

[Kong et al. 2020.] Kong, Qiuqiang ; Li, Bochen ; Chen, Jitong ; Wang, Yuxuan: *GiantMIDI-Piano: A large-scale MIDI dataset for classical piano music.* 2020.

[Kong et al. 2020] Kong, Qiuqiang ; Li, Bochen ; Song, Xuchen ; Wan, Yuan ; Wang, Yuxuan: High-resolution Piano Transcription with Pedals by Regressing Onsets and Offsets Times. In: *CoRR* abs/2010.01815 (2020). – URL `https://arxiv.org/abs/2010.01815`

[Kuperberg 2021] Kuperberg, Benjamin: *Meet Chataigne. Artist-friendly Modular Machine for Art and Technology.* `https://benjamin.kuperberg.fr/chataigne/en`. 08 2021. – (accessed on Aug 26, 2021)

[Lee 2008] Lee, Chun: Art Unlimited: An investigation into contemporary digital arts and the free software movement. (2008)

[Leotti et al. 2010] Leotti, L. ; Iyengar, S. ; Ochsner, K.: Born to choose: the origins and value of the need for control. In: *Trends in Cognitive Sciences* 14 (2010), S. 457–463

[Magnusson 2014] Magnusson, Thor: Scoring with Code: Composing with algorithmic notation. In: *Organised Sound* 19 (2014), Nr. 3, S. 268–275.

[Mansoux and Valk 2008] Mansoux, A. ; Valk, M. d.: *FLOSS+ART.* GOTO10, 2008

[Marier 2017] Marier, Martin: Musiques pour éponge : la composition pour un nouvel instrument de musique numérique. (2017). – URL `https://papyrus.bib.umontreal.ca/xmlui/handle/1866/20810`

[MEI 2021] MEI: *What is MEI?* `https://music-encoding.org/about/`. 08 2021. – (accessed on Aug 26, 2021)

[Michaud et al. 2015] Michaud, Pierre ; Bélanger, Olivier ; Paris, Lucas: LE PROJET Q-LIVE. In: *Journées d'Informatique Musicale.* Montréal, Canada, Mai 2015. – URL `https://hal.archives-ouvertes.fr/hal-03104607`

[Montague 1985] Montague, Stephen: John Cage at Seventy: An Interview. In: *American Music* 3 (1985), Nr. 2, S. 205–216. – URL `http://www.jstor.org/stable/3051637`. – ISSN 07344392, 19452349

[Morshad et al. 2020] Morshad, Sarwar ; Mazumder, Md ; Ahmed, Fahad: Analysis of Brain Wave Data Using Neurosky Mindwave Mobile II, 01 2020, S. 1–4.

[New Scientist 2017.] New Scientist: *Machines that think : everything you need to know about the coming age of artificial intelligence.* London, England : John Murray, 2017.

[Nieva et al. 2018] Nieva, Alex ; Wang, Johnty ; Malloch, Joseph W. ; Wanderley, Marcelo M.: The T-Stick: Maintaining a 12 year-old Digital Musical Instrument. In: *18th International Conference on New Interfaces for Musical Expression, NIME 2018, Blacksburg, VA, USA, June 3-6, 2018*, nime.org, 2018, S. 198–199. – URL `http://www.nime.org/proceedings/2018/nime2018_paper0042.pdf`

[Pariser 2011] Pariser, Eli: *The Filter Bubble: What the Internet Is Hiding from You.* 1st ptg. Penguin Press HC, The, 2011. – ISBN 1594203008,9781594203008,9781101515129

[Parviainen 2021] Parviainen, Tero: *How Generative Music Works: A Perspective.* 2021. – URL `https://teropa.info/loop/`. – (accessed on Dec 01, 2022)

[Radford et al. 2019.] Radford, A. ; Wu, Jeffrey ; Child, R. ; Luan, David ; Amodei, Dario ; Sutskever, Ilya: Language Models are Unsupervised Multitask Learners, 2019.

[Raghunathan 2021] Raghunathan, Raj: *Why Losing Control Can Make You Happier.* 2021. – URL `https://greatergood.berkeley.edu/article/item/why_losing_control_make_you_happier`. – (accessed on Dec 01, 2022)

[Russell and Norvig 2009] Russell, Stuart J. ; Norvig, Peter: *Artificial Intelligence: a modern approach.* 3. Pearson, 2009

[Ryan and Deci 2000] Ryan, Richard M. ; Deci, Edward L.: Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. In: *American Psychologist* 55 (2000), Nr. 1, S. 68–78. – URL `http://doi.apa.org/getdoi.cfm?doi=10.1037/0003-066X.55.1.68`. – ISSN 1935-990X, 0003-066X

[Simon and Oore 2017] Simon, Ian ; Oore, Sageev: *Performance RNN: Generating Music with Expressive Timing and Dynamics.* `https://magenta.tensorflow.org/performance-rnn`. 2017. – (accessed on Dec 01, 2022)

[Sullivan et al. 2018] Sullivan, John ; Tibbitts, Alexandra ; Gatinet, Brice ; Wanderley, Marcelo M.: Gestural Control of Augmented Instrumental Performance: A Case Study of the Concert Harp. In: *Proceedings of the 5th International Conference on Movement and Computing.* New York, NY, USA : Association for Computing Machinery, 2018 (MOCO '18). – URL `https://doi.org/10.1145/3212721.3212814`. – ISBN 9781450365048

[de Valk 2009] Valk, Marloes de: Tools to Fight Boredom: FLOSS and GNU/Linux for Artists Working in the Field of Generative Music and Software Art. In: *Contemporary Music Review* 28 (2009), S. 89–101.

[Van Nort and Castagné 2007] Van Nort, Doug ; Castagné, Nicolas: Mapping, in digital musical instruments. In: *Enaction and enactive interfaces : a handbook of terms.* Enactive Systems Books, 2007, S. 191–192. – URL `https://hal.archives-ouvertes.fr/hal-00977538`

[Van Rossum and Drake Jr 1995] Van Rossum, Guido ; Drake Jr, Fred L.: *Python tutorial.* Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995

[Vaz 2021] Vaz, Tiago B.: *The Beauty of Loosing Control (source code).* 2021. – URL `https://github.com/magenta/magenta/tree/main/magenta/models/performance_rnn`. – (accessed on Dec 01, 2022)

[Waite et al. 2016] Waite, E. ; Eck, D. ; Roberts, A. ; Abolafia, D.: *Magenta: Music and art generation with machine intelligence.* 2016. – URL `https://github.com/magenta/magenta`. – (accessed on Dec 01, 2022)

[Wang et al. 2016] Wang, Cheng-I ; Hsu, Jennifer ; Dubnov, Shlomo: Machine Improvisation with Variable Markov Oracle: Toward Guided and Structured Improvisation. In: *Comput. Entertain.* 14 (2016), Nr. 3. – URL `https://doi.org/10.1145/2905371`

[Willey 2014] Willey, Robert: The Editing and Arrangement of Conlon Nancarrow's Studies for Disklavier and Synthesizers. In: *Music Theory Online* 20 (2014)

[Youyou et al. 2015] Youyou, Wu ; Kosinski, Michal ; Stillwell, David: Computer-based personality judgments are more accurate than those made by humans. In: *Proceedings of the National Academy of Sciences* 112 (2015), Nr. 4, S. 1036–1040. – URL `https://www.pnas.org/doi/abs/10.1073/pnas.1418680112`

[Zylinska 2020] Zylinska, Joanna: *AI Art*. Open Humanities Press, 2020 (Media: Art: Write: Now)

Appendix A. List of additional files

Additional files

The following files are a complementary part of this thesis and therefore should be made available along with the present document:

```
Vaz_Tiago_2022_audio_simulation_01_Control_Taking_Control_Losing.wav
Vaz_Tiago_2022_video_capture_02_Youre_Thinking_Im_a_Good_Person.mp4
Vaz_Tiago_2022_video_capture_03_You_Cant_Un-neighbor_Your_Neighbor.mp4
```

Appendix B. Score of *Control (taking) / Control (losing)*

Control (taking) / Control (losing)

mixed media installation controlled by algorithms

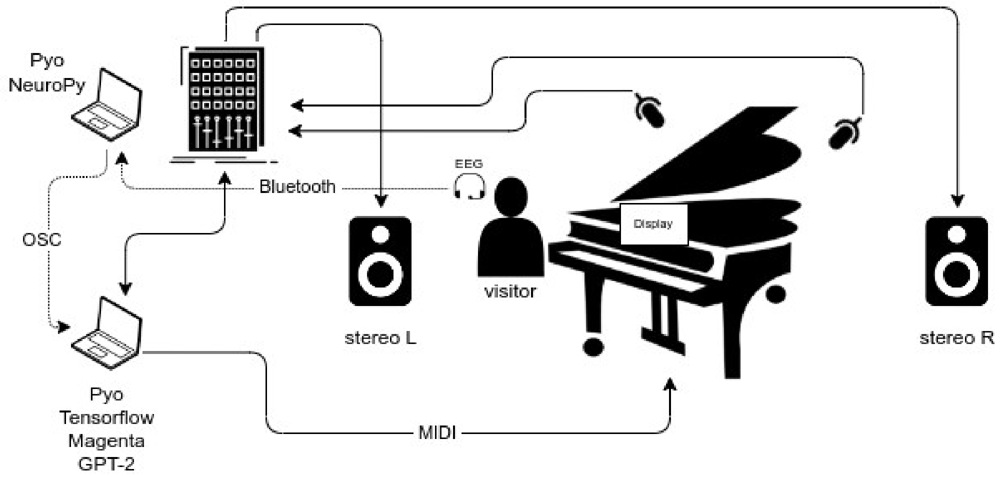
Tiago Vaz

Introductory notes

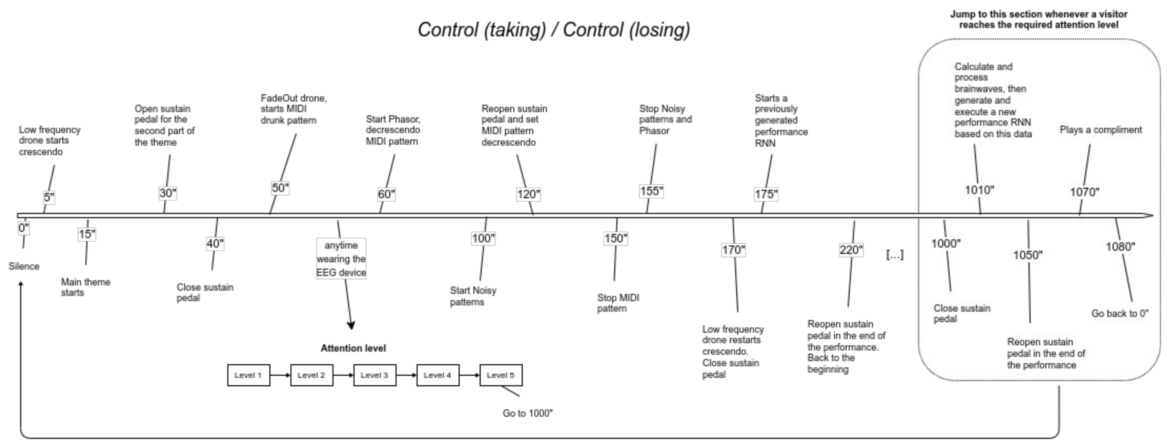
Control (taking) / Control (losing) is a mixed work presented as an interactive installation. It is structured in two sections and keeps running by itself once set as described in the next lines.

Scene

The figure below illustrates how the stage should be organized for the execution of this installation, as described in the next section.



A timeline representation of this piece, taken from its main algorithm, is illustrated bellow:


Control (taking) / Control (losing)
Jump to this section whenever a visitor reaches the required attention level
0"
Silence
5"
Low frequency drone starts crescendo
15"
Main theme starts
30"
Open sustain pedal for the second part of the theme
40"
Close sustain pedal
50"
FadeOut drone, starts MIDI drunk pattern
anytime wearing the EEG device
Attention level
Level 1
Level 2
Level 3
Level 4
Level 5
Go to 1000"
60"
Start Phasor, decrescendo MIDI pattern
100"
Start Noisy patterns
120"
Reopen sustain pedal and set MIDI pattern decrescendo
150"
Stop MIDI pattern
155"
Stop Noisy patterns and Phasor
170"
Low frequency drone restarts crescendo. Close sustain pedal
175"
Starts a previously generated performance RNN
220"
Reopen sustain pedal in the end of the performance. Back to the beginning
[...]
1000"
Close sustain pedal
1010"
Calculate and process brainwaves, then generate and execute a new performance RNN based on this data
1050"
Reopen sustain pedal in the end of the performance
1070"
Plays a compliment
1080"
Go back to 0"

Setup requirements

Hardware

- Two computers and one audio/MIDI interface
- A pair of speakers
- A digital player piano (i. e. a *Disklavier*)
- Two microphones to amplify the player piano (on the soundboard)
- Two contact microphones to amplify the mechanical structure under the player piano

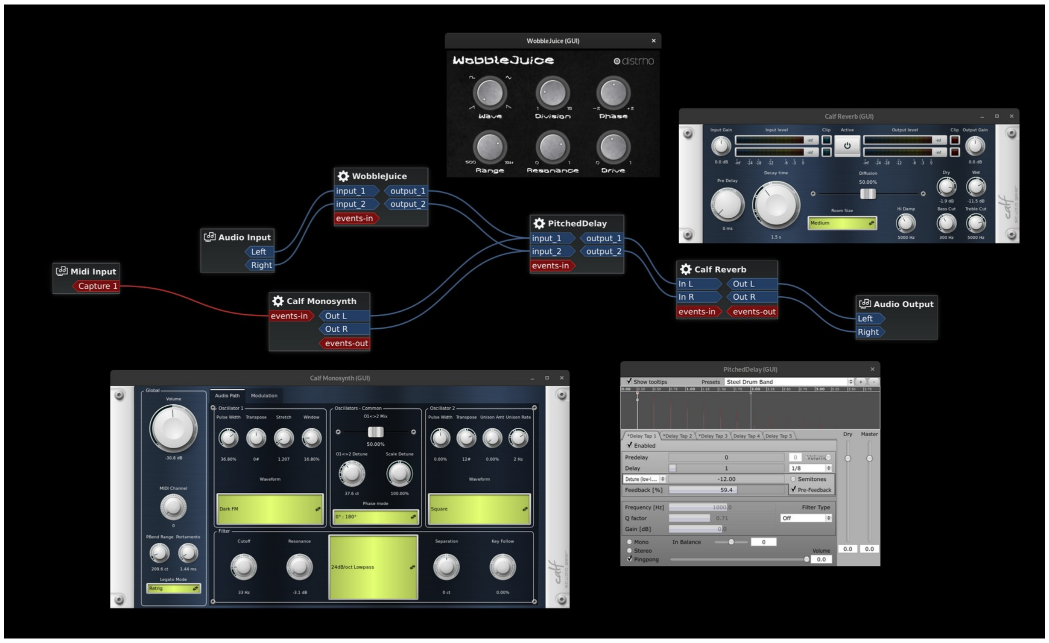
Software

- GNU/Linux distribution (might also run on other systems)
- Python version 3 and libraries installed as documented in the git repository located at https://github.com/tiagovaz/detaching
- Carla or similar audio plugins host with a similar effects chain as illustrated in the next section, item 5.

General instructions

Once all hardware and software are set up according to the instructions provided in the repository (https://github.com/tiagovaz/detaching), the following steps should be done for the installation:

1. On the main computer, open a terminal and run the Python script "`GPT2_daemon.py`". This script will be in charge of generating new lines of text to be displayed to visitors.
2. Open a terminal and run the Python script "`display.py`". This script will display in fullscreen the texts being generated by "`GPT2_daemon.py`". The screen will be placed on the music desk.
3. In the *NeuroPy* computer, run the Python script "`Control_EEG_daemon.py`". This script will connect to the Mindwave device and be in charge of collecting the brainwaves data and sending them to the main program via OSC, over a network. The IP address of the OSC receiver should be set at the beginning of the code.
4. Back to the main computer, run the Python script "`Control_part1.py`". This script will open a Pyo GUI which gives you the option to start the piece.
5. A *XML* file containing the configuration of audio objects and MIDI assignments is provided in the repository ("`carla_patch.carxp`"). This file is to be used by *Carla audio plugin host* software. Any other similar software can be used as long as the following chain of effects is respected:



PitchedDelay (GUI): Show tooltips ✓; Presets: Steel Drum Band; *Delay Tap 1 — Enabled ✓; Predelay 0; Delay 1, 1/8; Detune (low-l… -12.00; Semitones; Pre-Feedback ✓; Feedback [%] 59.4; Frequency [Hz] 1000.0; Q factor 0.71; Gain [dB] 0.0; Filter Type Off; Mono; Stereo; Pingpong ✓; In Balance 0; Volume 0.0; Dry 0.0; Master 0.0

Calf Monosynth (GUI): Global — Volume -30.8 dB; MIDI Channel 0; PBend Range 209.6 ct; Portamento 1.44 ms; Legato Mode Retrig. Oscillator 1 — Pulse Width 36.80%; Transpose 0#; Stretch 1.207; Window 16.80%; Waveform Dark FM. Oscillators - Common — O1<>2 Mix 50.00%; O1<>2 Detune 37.6 ct; Scale Detune 100.00%; Phase mode 0° : 180°. Oscillator 2 — Pulse Width 0.00%; Transpose 12#; Unison Amt 0.00%; Unison Rate 2 Hz; Waveform Square. Filter — Cutoff 33 Hz; Resonance -3.1 dB; 24dB/oct Lowpass; Separation 0 ct; Key Follow 0.00%

Control_part1.py

```python
import subprocess
from pyo import *
import mido
import random
import os

# Pyo initial setup
s = Server(audio='jack', duplex=0).boot()

# Dev purpose, use a virtual MIDI port (for mido)
DEV = True

if DEV is True:
    # Open virtual MIDI device
    port = mido.open_output('Midi Through:Midi Through Port-0 14:0')
else:
    # Open my actual MIDI hardware
    port = mido.open_output('io|2:io|2 MIDI 1 28:0')

########################### OSC RECEIVER ################################

# OSC receiver IP address
IP_RECEIVER = '127.0.0.1'

osc = OscReceive(port=10001, address=['/attention'])

tab_m = HarmTable([1,0,0,0,0,.3,0,0,0,0,0,.2,0,0,0,0,0,.1,0,0,0,0,.05]).normalize()
tab_p = HarmTable([1,0,.33,0,.2,0,.143,0,.111])

class Ring:
    def __init__(self, fport=250, fmod=100, amp=.3):
        self.mod = Osc(tab_m, freq=fmod, mul=amp)
        self.port = Osc(tab_p, freq=fport, mul=self.mod)

    def out(self):
        self.port.out()
        return self

    def sig(self):
        return self.port

res_mul = SigTo(Scale(osc['/attention'], 0, 100, 0, .05, exp=0), time=0.1)
res_amp = Fader(fadein=5, fadeout=5, mul=res_mul).stop()

lf = Sine(.03, mul=.5, add=1)
rg = Ring(fport=[random.choice([62.5, 125, 187.5, 250]) * random.uniform(.98, 1.02) for i
     in range(8)],
          fmod=lf * [random.choice([25, 50, 75, 100]) * random.uniform(.98, 1.02) for i
    in range(8)],
         amp=res_amp)

res = Waveguide(rg.sig(), freq=[30.1, 60.05, 119.7, 181, 242.5, 303.33, 599.8], dur=30,
    mul=res_amp).out()

################# SYNTH INSTRUMENTS CLASSES #######################################

# Need to call an external py in background to loop over the performance-rnn midi file
# no need to play with threads
class Performance():
    def __init__(self, cmd, arg):
        self.cmd = cmd
        self.arg = arg

    def play(self):
        p = subprocess.Popen([sys.executable, self.cmd, self.arg], stdout=subprocess.PIPE
    , stderr=subprocess.STDOUT)

class LowDrone():
    # Generated with Calf monosynth (avoid another live effect)
    def __init__(self):
        self.amp = Fader(fadein=20, fadeout=10, mul=.2)
        self.sf = SfPlayer("drone_part1.wav", loop=True, interp=2, mul=self.amp).out()

    def play(self):
        self.amp.play()
        return self

    def stop(self):
        self.amp.stop()
        return self

class Speech():
    def __init__(self, soundfile=[], loop=False, mul=.5, fadein=.01, fadeout=.01,
    duration=0, chnl=0, inc=1):
        self.amp = Fader(fadein=fadein, fadeout=fadeout, dur=duration, mul=mul)
        self.chnl = chnl
        self.inc = inc
        self.soundfile = soundfile
        self.soundfile_to_play = random.choice(soundfile)
        self.player = SfPlayer(self.soundfile_to_play, mul=[self.amp/2., self.amp/1.95],
    loop=loop).stop()
        self.player_rev = Freeverb(self.player, size=[.3,.25], damp=.6, bal=.4, mul=.8).
    out(chnl=self.chnl, inc=self.inc)

    def setDur(self, dur):
        self.amp.dur = dur
        return self

    def play(self):
        self.player.setSound(random.choice(self.soundfile))
        self.player.play()
        self.amp.play()
        return self

    def stop(self):
        self.amp.stop()
        return self

    def getOut(self):
        return self.amp

class CustomPattern():
    """
    Instruments is a dict instrument:beats (pyoObj:[list of int])
    """

    def __init__(self, instruments={}, time=.25, beats=32):
        self.current_beat = 1
        self.time = time
        self.instruments = instruments
        self.beats = beats
        self.p = Pattern(self.pat, time).stop()

    def pat(self):
        for k, v in self.instruments.items():
            if self.current_beat in v:
                k.play()
            if self.current_beat == self.beats:
                self.current_beat = 0
        self.current_beat += 1

    def play(self):
        self.p.play()

    def stop(self):
        self.p.stop()

class HighFreq():
    def __init__(self, freq=[11200, 11202], dur=.4, mul=.5):
        self.amp = Fader(fadein=.01, fadeout=.01, dur=dur, mul=mul)
        self.sine = SineLoop(freq=freq, mul=self.amp * .05).out()
        self.rev = Freeverb(self.sine, size=.84, damp=.87, bal=.9, mul=self.amp * .2).out
    ()

    def setDur(self, dur):
        self.amp.dur = dur
        return self

    def play(self):
        self.amp.play()
        return self

    def stop(self):
        self.amp.stop()
        return self

    def getOut(self):
        return self.amp

class SmoothNoise():
    def __init__(self, dur=1.3, mul=.5):
        self.amp = Fader(fadein=.1, fadeout=.01, dur=dur, mul=mul)
        self.noise = PinkNoise(self.amp * .01).mix(2).out()

    def setDur(self, dur):
        self.amp.dur = dur
        return self

    def play(self):
        self.amp.play()
        return self

    def stop(self):
        self.amp.stop()
        return self

    def getOut(self):
        return self.amp

    def setInput(self, x, fadetime=.001):
        self.input.setInput(x, fadetime)

class BigPhasor():
    def __init__(self):
        self.amp = Fader(fadein=.1, fadeout=.01, dur=0, mul=.03)
        self.freq = Phasor(freq=1, mul=100, add=100)
        self.inst = Sine(freq=self.freq, mul=.2)
        self.outsig = STRev(self.inst, mul=self.amp*.1).out()

    def play(self):
        self.amp.play()

    def stop(self):
        self.amp.stop()

############ MIDI EVENTS CLASSES ################################################

class MidiEvent1():
    def __init__(self):
        # self.pitch = XnoiseMidi('walker', freq=0.125, scale=0, mrange=(0, 127))
        self.pitch_range_freq = Randi(0.01, 0.5)
        self.pitch_range = Randi(-10, 100, freq=self.pitch_range_freq)
        self.pitch = Randi(21, 25, freq=125, add=self.pitch_range)
        self.velocity = Randi(10, 30, freq=Randi(1, 7))
        self.duration = 100

    def send_midi(self):
        # self.pitch = random.randint(21, 88)
        s.makenote(pitch=int(self.pitch.get()), velocity=int(self.velocity.get()),
    duration=self.duration)
        # print("playing", self.pitch)

# Add similar to the piece with daniel (repetition/minimalist)
class MidiEvent2():
    def __init__(self):
        pass

################# MIDI EVENT OBJECTS ################################################

m = MidiEvent1()

################# INSTRUMENTS OBJECTS ###############################################

big_phasor = BigPhasor()
snoise = SmoothNoise(mul=.15)
high = HighFreq(mul=.05)
low_drone = LowDrone()
noise_pattern = CustomPattern({high: [3, 30], snoise: [15, 31]}, time=.25, beats=64)

# Pick the theme
theme = Performance('./play_midi.py', 'theme')

# Pick a random PerformanceRNN
performance = Performance('./play_midi.py', 'rnn')

# Patterns won't work inside the the classes, so here they are:
r = Randi(0.1, 0.125, freq=Randi(.1, .5))
midi_pat = Pattern(m.send_midi, time=r)

# Voices
wearing = Speech(['voice_wearing.wav']).stop()
level1 = Speech(['voice_01.wav']).stop()
level2 = Speech(['voice_02.wav']).stop()
level3 = Speech(['voice_03.wav']).stop()
level4 = Speech(['voice_04.wav']).stop()
level5 = Speech(['voice_05.wav']).stop()

################# MAIN SCORE ################################################

# Main counter
time_count = -1

# Initial level
```

```
level_next = 0

# Counter which determines how long a high level of attention has been reached
high_attention_counter = 0

# Global attention counter, to calculate attention average
attention_counter = 1

# Attention sum, used to calculate average
attention_sum = 0

# Attention samples counter, to calculate attention average
attention = 1

# Send a value to the MIDI player so it can stop playing once a high
# level of attention is reached
stop_midi_osc = OscDataSend("i", 10003, "/stop_midi")

# Attention average, used to generate the next performance-rnn
attention_average = 0

attention_done = False
attention_init = True

# To calculate delay between attention levels and allow voice to finish playing
attention_timestamp = 0

# After playing
compliments_list = ['breathtaking.wav', 'brilliant.wav', 'fabulous.wav', 'magnificent.wav
    ']
compliment = Speech(compliments_list).stop()

def score():
    global \
        time_count, \
        level1, level2, level3, level4, level5, wearing, level_next, \
        m, \
        mainScore, \
        attention_counter, \
        attention_sum, \
        attention_average, \
        stop_midi_osc,\
        attention_done,\
        attention_init,\
        attention_timestamp,\
        high_attention_counter

    time_count = time_count + 1
    attention = int(osc.get('/attention'))

    print("-------------------------------------------------------------------")
```

```python
print("MAIN COUNTER: ", time_count)
print("ATTENTION:", attention)
print("NEXT LEVEL:", level_next)

# Someone wears the EEG device
if attention != 0:
    if attention_init is True:
        print("WEARING EEG")
        attention_init = False
        level_next = 1
        wearing.play()
        attention_timestamp = time_count

    # Average of attention level at the beginning will impact the next performance
generation
    attention_sum = attention_sum + attention
    attention_counter = attention_counter + 1
    if attention_counter == 31:
        attention_average = attention_sum / 30

    # Audible feedback from attention
    if attention_done is False:
        res_amp.out()

    # Good level of attention reached
    if attention > 40 and attention_done is False:
        print("ATTENTION TIMESTAMP:", attention_timestamp)
        if level_next == 1 and time_count > (attention_timestamp + 11):
            high_attention_counter = high_attention_counter + 1
            print("HIGH ATTENTION COUNTER:", high_attention_counter)

    if high_attention_counter == 1 and level_next == 1 and time_count > (
attention_timestamp + 11):
        attention_timestamp = time_count
        level_next = 2
        print("LEVEL 1 REACHED")
        level1.play()
        high_attention_counter = high_attention_counter + 1
        print("HIGH ATTENTION COUNTER:", high_attention_counter)

    elif high_attention_counter == 2 and level_next == 2 and time_count > (
attention_timestamp + 12):
        attention_timestamp = time_count
        level_next = 3
        print("LEVEL 2 REACHED")
        level2.play()
        high_attention_counter = high_attention_counter + 1
        print("HIGH ATTENTION COUNTER:", high_attention_counter)

    elif high_attention_counter == 3 and level_next == 3 and time_count > (
attention_timestamp + 8):
```

```python
            attention_timestamp = time_count
            level_next = 4
            print("LEVEL 3 REACHED")
            level3.play()
            high_attention_counter = high_attention_counter + 1
            print("HIGH ATTENTION COUNTER:", high_attention_counter)

        elif high_attention_counter == 4 and level_next == 4 and time_count > (attention_timestamp + 10):
            attention_timestamp = time_count
            level_next = 5
            print("LEVEL 4 REACHED")
            level4.play()
            high_attention_counter = high_attention_counter + 1
            print("HIGH ATTENTION COUNTER:", high_attention_counter)

        # If visitor reaches a good level of attention a few times, stop and go to part 2
        if high_attention_counter == 5 and level_next == 5 and time_count > (attention_timestamp + 10):
            level_next = 0
            level5.play()
            attention_done = True
            print("HIGH ATTENTION LEVEL REACHED")

            # Stop MIDI player via OSC
            stop_midi_osc.send([1])
            high_attention_counter = 0
            attention_counter = 0
            midi_pat.stop()

            print("EXECUTING PART 2...")

            # Jump like a coda
            time_count = 1000

    else:
        # No visitor wearing the EEG device
        res_amp.stop()

    if time_count == 5:
        # Start drone
        low_drone.play()

    if time_count == 15:
        # Start main PerformanceRNN theme
        theme.play()

    if time_count == 30:
        # Open sustain pedal in the second part of the theme
        s.ctlout(64, 127)

    if time_count == 40:
        # Close pedal after a few seconds, preparing for the semi-random pattern
        s.ctlout(64, 0)

    if time_count == 50:
        # MIDI pattern starting slowly
        r.setMin(.5)
        r.setMax(1)
        midi_pat.play()
        low_drone.stop()

    if time_count == 60:
        # Add some drama
        big_phasor.play()
        # Pattern goes crescendo
        r.setMin(.1)
        r.setMax(.125)

    if time_count == 100:
        # Stop noise pattern
        noise_pattern.play()

    if time_count == 120:
        # Open sustain and go decrescendo
        s.ctlout(64, 127)
        r.setMin(.25)
        r.setMax(.5)

    if time_count == 150:
        # Stop noise and midi patterns
        midi_pat.stop()

    if time_count == 155:
        # Stop noise
        noise_pattern.stop()
        # Too much drama
        big_phasor.stop()

    if time_count == 170:
        low_drone.play()
        # Close sustain
        s.ctlout(64, 0)

    if time_count == 175:
        # Start brainwave-driven past performance (will last 2 min)
        performance.play()

    if time_count == 220:
        # Reopen sustain
        s.ctlout(64, 127)

    #if time_count == 230:
        # Back to the beginning
        #time_count = -1

    # 2 minutes after part 2
    if time_count == 1000:
        s.ctlout(64, 0)
        print("LOOP X SECONDS DURING PART 2")
        res.stop()
        #TODO: add TTS instruction to remove the device
        #TODO: do mapping, pick better scales etc

    if time_count == 1010:
        temperature = random.uniform(0.7, 1.4)
        notes_per_second_mul = int(attention_average / 10)
        notes_per_second = [notes_per_second_mul - 1, notes_per_second_mul + 2,
    notes_per_second_mul + 10, notes_per_second_mul - 2, 1]

        # Choose performance RNN based on attention average
        if attention_average < 40:
            bundle_file = 'bundles/custom_model.mag'
        if attention_average >= 40 and attention_average < 60:
            bundle_file = 'bundles/magenta.mag'
        if attention_average >= 60 and attention_average < 80:
            bundle_file = 'bundles/bach.mag'
        if attention_average >= 80:
            bundle_file = 'bundles/debussy_rnn.mag'

        cmd = 'python3 custom_generator.py --output_dir=./midi/ ' \
              '--config=multiconditioned_performance_with_dynamics ' \
              '--num_steps=12000 ' \
              '--notes_per_second="' + str(notes_per_second) + \
              '" --pitch_class_histogram="[[0, 1, 3, 0, 1, 1, 0, 1, 0, 2, 1, 0],' \
                                        '[0, 1, 0, 1, 0, 1, 0, 2, 0, 1, 0, 1],' \
                                        '[0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1],' \
                                        '[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0],' \
                                        '[0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 0]]"'\
              '--temperature=' + str(temperature) + \
              '--num_outputs=1 --bundle_file=' + str(bundle_file)

        print(cmd)
        # Calls the RNN generator
        p = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.
    STDOUT)

    # Open sustain in the end of the performance-rnn
    if time_count == 1050:
        s.ctlout(64, 127)

    if time_count == 1070:
        attention_done = False
```

```python
        attention_init = True

        # Compliments after playing generated music
        compliment.play()

    if time_count == 1080:
        # Start over
        time_count = -1

# Cloudy seconds counter
mainTime = Cloud(density=1).play()
mainScore = TrigFunc(mainTime, score)

mainScore.play()

s.gui(locals())
```

Control_EEG_daemon.py

```python
from pyo import *
from NeuroPy import NeuroPy
import time

# OSC receiver IP address
IP_RECEIVER = ’127.0.0.1’

# Simulate EEG signal rather than getting it from an actual device (for dev purpose)
# True or False
SIMULATE_OSC = False

# Pyo initial setup
s = Server(audio=’jack’)
s.boot().start()


# EEG initial setup
if SIMULATE_OSC is False:
    bw = NeuroPy("/dev/rfcomm0")
    bw.start()
    # Send audio stream instead
    bw_input = SigTo([bw.attention, bw.meditation, bw.poorSignal])
    # Send OSC signals to receivers
    bw_stream_part1 = OscSend(bw_input, port=10001, address=[’/attention’, ’/meditation’, ’/poorSignal’],
                                  host=IP_RECEIVER)
    bw_stream_midi_player = OscSend(bw_input, port=10002, address=[’/attention’, ’/meditation’, ’/poorSignal’],
                                        host=IP_RECEIVER)
    # Update values
    while True:
        print([bw.attention, bw.meditation, bw.poorSignal])
        bw_input.setValue([bw.attention, bw.meditation, bw.poorSignal])
        time.sleep(0.01)

else:
    r = Randi(.05, .1)
    bw_input = Sine(freq=[r, 1.5, 1], mul=[50, .1, 1], add=50)
    bw_stream_part1 = OscSend(bw_input, port=10001, address=[’/attention’, ’/meditation’, ’/poorSignal’],
                                  host=IP_RECEIVER)
    bw_stream_midi_player = OscSend(bw_input, port=10002, address=[’/attention’, ’/meditation’, ’/poorSignal’],
                                        host=IP_RECEIVER)
```

play_midi.py

```python
import os
from random import choice
import mido
import sys
import time
from pyo import *
s = Server(audio='jack', duplex=0).boot().start()

# Dev purpose, use a virtual MIDI port (for mido)
DEV = True

if DEV is True:
    # Open virtual MIDI device
    port = mido.open_output('Midi Through:Midi Through Port-0 14:0')
else:
    # Open my actual MIDI hardware
    port = mido.open_output('io|2:io|2 MIDI 1 28:0')

# Stops the play when someone reaches a given level of attention
def stop_now(address, *args):
    print(address, args)
    if args[0] == 1:
        exit(0)

osc = OscReceive(port=10002, address=['/attention'])
osc_stop_midi = OscDataReceive(10003, "/stop_midi", stop_now)

if sys.argv[1] == 'theme':
    mid = mido.MidiFile('theme_tassia.mid')
else:
    mid = mido.MidiFile('midi/' + choice(os.listdir('midi')))
for message in mid.play():
    if message.type == 'note_on' or message.type == 'note_off':
        delay = osc.get('/attention') / 50
        print(delay)
        time.sleep(delay)
        port.send(message)
```

custom_generator.py

```python
#!/usr/bin/python3
#
# Copyright 2021 The Magenta Authors.
# Modified by Tiago Bortoletto Vaz <tiago@acaia.ca>, 2022
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Lint as: python3

"""Generate polyphonic performances from a trained checkpoint.

Uses flags to define operation.
"""
import ast
import os
import time

from magenta.models.performance_rnn import performance_model
from magenta.models.performance_rnn import performance_sequence_generator
from magenta.models.shared import sequence_generator
from magenta.models.shared import sequence_generator_bundle
import note_seq
from note_seq.protobuf import generator_pb2
from note_seq.protobuf import music_pb2
import tensorflow.compat.v1 as tf

import mido


# This will change the way it names the output files
LIVE = True


FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string(
    'run_dir', None,
    'Path to the directory where the latest checkpoint will be loaded from.')
tf.app.flags.DEFINE_string(
    'bundle_file', None,
    'Path to the bundle file. If specified, this will take priority over '
    'run_dir, unless save_generator_bundle is True, in which case both this '
    'flag and run_dir are required')
tf.app.flags.DEFINE_boolean(
    'save_generator_bundle', False,
    'If true, instead of generating a sequence, will save this generator as a '
    'bundle file in the location specified by the bundle_file flag')
tf.app.flags.DEFINE_string(
    'bundle_description', None,
    'A short, human-readable text description of the bundle (e.g., training '
    'data, hyper parameters, etc.).')
tf.app.flags.DEFINE_string(
    'config', 'performance', 'Config to use.')
tf.app.flags.DEFINE_string(
    'output_dir', '/tmp/performance_rnn/generated',
    'The directory where MIDI files will be saved to.')
tf.app.flags.DEFINE_integer(
    'num_outputs', 10,
    'The number of tracks to generate. One MIDI file will be created for '
    'each.')
tf.app.flags.DEFINE_integer(
    'num_steps', 3000,
    'The total number of steps the generated track should be, priming '
    'track length + generated steps. Each step is 10 milliseconds.')
tf.app.flags.DEFINE_string(
    'primer_pitches', '',
    'A string representation of a Python list of pitches that will be used as '
    'a starting chord with a quarter note duration. For example: '
    '"[60, 64, 67]"')
tf.app.flags.DEFINE_string(
    'primer_melody', '', 'A string representation of a Python list of '
    'note_seq.Melody event values. For example: '
    '"[60, -2, 60, -2, 67, -2, 67, -2]". The primer melody will be played at '
    'a fixed tempo of 120 QPM with 4 steps per quarter note.')
tf.app.flags.DEFINE_string(
    'primer_midi', '',
    'The path to a MIDI file containing a polyphonic track that will be used '
    'as a priming track.')
tf.app.flags.DEFINE_string(
    'disable_conditioning', None,
    'When optional conditioning is available, a string representation of a '
    'Boolean indicating whether or not to disable conditioning. Similar to '
    'control signals, this can also be a list of Booleans; when it is a list, '
    'the other conditioning variables will be ignored for segments where '
    'conditioning is disabled.')
tf.app.flags.DEFINE_float(
    'temperature', 1.0,
    'The randomness of the generated tracks. 1.0 uses the unaltered '
    'softmax probabilities, greater than 1.0 makes tracks more random, less '
    'than 1.0 makes tracks less random.')
tf.app.flags.DEFINE_integer(
    'beam_size', 1,
    'The beam size to use for beam search when generating tracks.')
tf.app.flags.DEFINE_integer(
    'branch_factor', 1,
    'The branch factor to use for beam search when generating tracks.')
tf.app.flags.DEFINE_integer(
    'steps_per_iteration', 1,
    'The number of steps to take per beam search iteration.')
tf.app.flags.DEFINE_string(
    'log', 'INFO',
    'The threshold for what messages will be logged DEBUG, INFO, WARN, ERROR, '
    'or FATAL.')
tf.app.flags.DEFINE_string(
    'hparams', '',
    'Comma-separated list of `name=value` pairs. For each pair, the value of '
    'the hyperparameter named `name` is set to `value`. This mapping is merged '
    'with the default hyperparameters.')

# Add flags for all performance control signals.
for control_signal_cls in note_seq.all_performance_control_signals:
  tf.app.flags.DEFINE_string(
      control_signal_cls.name, None, control_signal_cls.description)


def get_checkpoint():
  """Get the training dir or checkpoint path to be used by the model."""
  if FLAGS.run_dir and FLAGS.bundle_file and not FLAGS.save_generator_bundle:
    raise sequence_generator.SequenceGeneratorError(
        'Cannot specify both bundle_file and run_dir')
  if FLAGS.run_dir:
    train_dir = os.path.join(os.path.expanduser(FLAGS.run_dir), 'train')
    return train_dir
  else:
    return None


def get_bundle():
  """Returns a generator_pb2.GeneratorBundle object based read from bundle_file.

  Returns:
    Either a generator_pb2.GeneratorBundle or None if the bundle_file flag is
    not set or the save_generator_bundle flag is set.
  """
  if FLAGS.save_generator_bundle:
    return None
  if FLAGS.bundle_file is None:
    return None
  bundle_file = os.path.expanduser(FLAGS.bundle_file)
  return sequence_generator_bundle.read_bundle_file(bundle_file)


```

```
def run_with_flags(generator):
  """Generates performance tracks and saves them as MIDI files.

  Uses the options specified by the flags defined in this module.

  Args:
    generator: The PerformanceRnnSequenceGenerator to use for generation.
  """
  if not FLAGS.output_dir:
    tf.logging.fatal('--output_dir required')
    return
  output_dir = os.path.expanduser(FLAGS.output_dir)

  primer_midi = None
  if FLAGS.primer_midi:
    primer_midi = os.path.expanduser(FLAGS.primer_midi)

  if not tf.gfile.Exists(output_dir):
    tf.gfile.MakeDirs(output_dir)

  primer_sequence = None
  if FLAGS.primer_pitches:
    primer_sequence = music_pb2.NoteSequence()
    primer_sequence.ticks_per_quarter = note_seq.STANDARD_PPQ
    for pitch in ast.literal_eval(FLAGS.primer_pitches):
      note = primer_sequence.notes.add()
      note.start_time = 0
      note.end_time = 60.0 / note_seq.DEFAULT_QUARTERS_PER_MINUTE
      note.pitch = pitch
      note.velocity = 100
      primer_sequence.total_time = note.end_time
  elif FLAGS.primer_melody:
    primer_melody = note_seq.Melody(ast.literal_eval(FLAGS.primer_melody))
    primer_sequence = primer_melody.to_sequence()
  elif primer_midi:
    primer_sequence = note_seq.midi_file_to_sequence_proto(primer_midi)
  else:
    tf.logging.warning(
        'No priming sequence specified. Defaulting to empty sequence.')
    primer_sequence = music_pb2.NoteSequence()
    primer_sequence.ticks_per_quarter = note_seq.STANDARD_PPQ

  # Derive the total number of seconds to generate.
  seconds_per_step = 1.0 / generator.steps_per_second
  generate_end_time = FLAGS.num_steps * seconds_per_step

  # Specify start/stop time for generation based on starting generation at the
  # end of the priming sequence and continuing until the sequence is num_steps
  # long.
  generator_options = generator_pb2.GeneratorOptions()
  # Set the start time to begin when the last note ends.
```

```
generate_section = generator_options.generate_sections.add(
    start_time=primer_sequence.total_time,
    end_time=generate_end_time)

if generate_section.start_time >= generate_section.end_time:
  tf.logging.fatal(
      'Priming sequence is longer than the total number of steps '
      'requested: Priming sequence length: %s, Total length '
      'requested: %s',
      generate_section.start_time, generate_end_time)
  return

for control_cls in note_seq.all_performance_control_signals:
  if FLAGS[control_cls.name].value is not None and (
      generator.control_signals is None or not any(
          control.name == control_cls.name
          for control in generator.control_signals)):
    tf.logging.warning(
        'Control signal requested via flag, but generator is not set up to '
        'condition on this control signal. Request will be ignored: %s = %s',
        control_cls.name, FLAGS[control_cls.name].value)

if (FLAGS.disable_conditioning is not None and
    not generator.optional_conditioning):
  tf.logging.warning(
      'Disable conditioning flag set, but generator is not set up for '
      'optional conditioning. Requested disable conditioning flag will be '
      'ignored: %s', FLAGS.disable_conditioning)

if generator.control_signals:
  for control in generator.control_signals:
    if FLAGS[control.name].value is not None:
      generator_options.args[control.name].string_value = (
          FLAGS[control.name].value)
if FLAGS.disable_conditioning is not None:
  generator_options.args['disable_conditioning'].string_value = (
      FLAGS.disable_conditioning)

generator_options.args['temperature'].float_value = FLAGS.temperature
generator_options.args['beam_size'].int_value = FLAGS.beam_size
generator_options.args['branch_factor'].int_value = FLAGS.branch_factor
generator_options.args[
    'steps_per_iteration'].int_value = FLAGS.steps_per_iteration

tf.logging.debug('primer_sequence: %s', primer_sequence)
tf.logging.debug('generator_options: %s', generator_options)

# Make the generate request num_outputs times and save the output as midi
# files.
date_and_time = time.strftime('%Y-%m-%d_%H%M%S')
digits = len(str(FLAGS.num_outputs))
```

```
  for i in range(FLAGS.num_outputs):
    generated_sequence = generator.generate(primer_sequence, generator_options)

    # Save MIDI files using parameters in the name
    print(FLAGS.pitch_class_histogram)
    duration_in_sec = int(FLAGS.num_steps / 100)
    bundle = FLAGS.bundle_file.split('/')[-1].split('.')[0]
    primer = FLAGS.primer_midi.split('/')[-1].split('.')[0]

    if LIVE is True:
        midi_filename = '%s.mid' % (date_and_time)
    else:
        midi_filename = '%s_%s_%s_%s_%s_%s_%s_%s.mid' % \
                        (date_and_time,  str(bundle), str(FLAGS.temperature),
                         str(duration_in_sec), str(FLAGS.notes_per_second),
                         str(FLAGS.pitch_class_histogram), str(primer),
                         str(i + 1).zfill(digits))
    midi_path = os.path.join(output_dir, midi_filename)
    note_seq.sequence_proto_to_midi_file(generated_sequence, midi_path)

  tf.logging.info('Wrote %d MIDI files to %s',
                  FLAGS.num_outputs, output_dir)
  # Play MIDI after generating performance

  DEV = True

  if DEV is True:
      # Open virtual MIDI device
      port = mido.open_output('Midi Through:Midi Through Port-0 14:0')
  else:
      # Open my actual MIDI hardware
      port = mido.open_output('io|2:io|2 MIDI 1 28:0')

  mid = mido.MidiFile(midi_path)

  for message in mid.play():
      if message.type == 'note_on' or message.type == 'note_off':
          port.send(message)

def main(unused_argv):
  """Saves bundle or runs generator based on flags."""
  tf.logging.set_verbosity(FLAGS.log)

  bundle = get_bundle()

  config_id = bundle.generator_details.id if bundle else FLAGS.config
  config = performance_model.default_configs[config_id]
  config.hparams.parse(FLAGS.hparams)
  # Having too large of a batch size will slow generation down unnecessarily.
  config.hparams.batch_size = min(
      config.hparams.batch_size, FLAGS.beam_size * FLAGS.branch_factor)

  generator = performance_sequence_generator.PerformanceRnnSequenceGenerator(
      model=performance_model.PerformanceRnnModel(config),
      details=config.details,
      steps_per_second=config.steps_per_second,
      num_velocity_bins=config.num_velocity_bins,
      control_signals=config.control_signals,
      optional_conditioning=config.optional_conditioning,
      checkpoint=get_checkpoint(),
      bundle=bundle,
      note_performance=config.note_performance)

  if FLAGS.save_generator_bundle:
    bundle_filename = os.path.expanduser(FLAGS.bundle_file)
    if FLAGS.bundle_description is None:
      tf.logging.warning('No bundle description provided.')
    tf.logging.info('Saving generator bundle to %s', bundle_filename)
    generator.create_bundle_file(bundle_filename, FLAGS.bundle_description)
  else:
    run_with_flags(generator)

def console_entry_point():
  tf.disable_v2_behavior()
  tf.app.run(main)

if __name__ == '__main__':
  console_entry_point()
```

NeuroPy.py

```
##Copyright (c) 2013, sahil singh
##Deprecated code fixed by Tiago Bortoletto Vaz, 2021, 2022
##
##All rights reserved.
##
##Redistribution and use in source and binary forms, with or without modification,
##are permitted provided that the following conditions are met:
##
##    * Redistributions of source code must retain the above copyright notice,
##      this list of conditions and the following disclaimer.
##    * Redistributions in binary form must reproduce the above copyright notice,
##      this list of conditions and the following disclaimer in the documentation
##      and/or other materials provided with the distribution.
##    * Neither the name of NeuroPy nor the names of its contributors
##      may be used to endorse or promote products derived from this software
##      without specific prior written permission.
##
##THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
##"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
##LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
##A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
##CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
##EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
##PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
##PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
##LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
##NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
##SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

import serial
import time
import sys
import struct
import _thread as thread


SYNC = b'\xaa'
POOR_SIGNAL = b'\x02'
ATTENTION = b'\x04'
MEDITATION = b'\x05'
BLINK_STRENGTH = b'\x16'
RAW_VALUE = b'\x80'
ASIC_EEG_POWER = b'\x83'

class NeuroPy(object):
    __attention=0
    __meditation=0
    __rawValue=0
    __delta=0
    __theta=0
    __lowAlpha=0
    __highAlpha=0
    __lowBeta=0
    __highBeta=0
    __lowGamma=0
    __midGamma=0
    __poorSignal=0
    __blinkStrength=0

    srl=None
    __port=None
    __baudRate=None

    callBacksDictionary = {}  # keep a track of all callbacks
    running = True #here we mark it start running
    def __init__(self, port, baudRate=57600):
        platform = sys.platform
        print(platform)
        self.__port,self.__baudRate=port,baudRate
        self.__packetsReceived = 0

    def __del__(self):
        if self.running == True:
            self.running = False
        self.srl.close()

    #need stderr in python
    def eprint(*args, **kwargs):
        print(*args, file=sys.stderr, **kwargs)
    # disconnect function should be no longer needed as long as we stop the srl
    '''def disconnect(self):
        self.__srl.write(DISCONNECT) '''
    # connect function no longer needed, we will do it in the start function
    '''def connect(self):
        self.__connected = True
        return # Only connect RF devices
        self.__srl.write(''.join([CONNECT, self.__devid.decode('hex')])) '''

    def start(self):
        # Try to connect to serial port and start a separate thread
        # for data collection
        self.running = True
        print ("Mindwave has already started!")
        try:
            self.srl=serial.Serial(self.__port)
        except serial.serialutil.SerialException:
            eprint ("Mindewave could not start due to Serial Exception")
        thread.start_new_thread(self.__packetParser,(self.srl,))
        self.__packetsReceived = 0

```

```

def __packetParser(self, srl):
    "packetParser runs continously in a separate thread to parse packets from mindwave and update the corresponding variables"
    while self.running:
        p1 = srl.read(1).hex()  # read first 2 packets
        p2 = srl.read(1).hex()
        while (p1 != 'aa' or p2 != 'aa'):
            p1 = p2
            p2 = srl.read(1).hex()
        else:
            if self.running == False:
                break
            # a valid packet is available
            self.__packetsReceived += 1
            payload = []
            checksum = 0
            payloadLength = int(srl.read(1).hex(), 16)
            for i in range(payloadLength):
                tempPacket = srl.read(1).hex()
                payload.append(tempPacket)
                checksum += int(tempPacket, 16)
            checksum = ~checksum & 0x000000ff
            if checksum == int(srl.read(1).hex(), 16):
                i = 0

                while i < payloadLength:
                    code = payload[i]
                    if (code == 'd0'):
                        print("Headset connected!")
                    elif (code == 'd1' or code == 'd2' or code == 'd3' or code == 'd4'):
                        eprint("Headset connection failed")
                    elif(code == '02'):  # poorSignal
                            i = i + 1
                            self.poorSignal = int(payload[i], 16)
                    elif(code == '04'):  # attention
                            i = i + 1
                            self.attention = int(payload[i], 16)
                    elif(code == '05'):  # meditation
                        i = i + 1
                        self.meditation = int(payload[i], 16)
                    elif(code == '16'):  # blink strength
                        i = i + 1
                        self.blinkStrength = int(payload[i], 16)
                    elif(code == '80'):  # raw value
                        i = i + 1  # for length/it is not used since length =1 byte long and always=2
                        i = i + 1
                        val0 = int(payload[i], 16)
                        i = i + 1
                        self.rawValue = val0 * 256 + int(payload[i], 16)
                        if self.rawValue > 32768:
                            self.rawValue = self.rawValue - 65536
                    elif(code == '83'):  # ASIC_EEG_POWER
                        i = i + 1  # for length/it is not used since length =1 byte long and always=2
                        # delta:
                        i = i + 1
                        val0 = int(payload[i], 16)
                        i = i + 1
                        val1 = int(payload[i], 16)
                        i = i + 1
                        self.delta = val0 * 65536 + val1 * 256 + int(payload[i], 16)
                        # theta:
                        i = i + 1
                        val0 = int(payload[i], 16)
                        i = i + 1
                        val1 = int(payload[i], 16)
                        i = i + 1
                        self.theta = val0 * 65536 + val1 * 256 + int(payload[i], 16)
                        # lowAlpha:
                        i = i + 1
                        val0 = int(payload[i], 16)
                        i = i + 1
                        val1 = int(payload[i], 16)
                        i = i + 1
                        self.lowAlpha = val0 * 65536 + val1 * 256 + int(payload[i], 16)
                        # highAlpha:
                        i = i + 1
                        val0 = int(payload[i], 16)
                        i = i + 1
                        val1 = int(payload[i], 16)
                        i = i + 1
                        self.highAlpha = val0 * 65536 + val1 * 256 + int(payload[i], 16)
                        # lowBeta:
                        i = i + 1
                        val0 = int(payload[i], 16)
                        i = i + 1
                        val1 = int(payload[i], 16)
                        i = i + 1
                        self.lowBeta = val0 * 65536 + val1 * 256 + int(payload[i], 16)
                        # highBeta:
                        i = i + 1
                        val0 = int(payload[i], 16)
                        i = i + 1
                        val1 = int(payload[i], 16)
                        i = i + 1
```

```
                            self.highBeta = val0 * 65536 + val1 * 256 + int(payload[i], 16)
                            # lowGamma:
                            i = i + 1
                            val0 = int(payload[i], 16)
                            i = i + 1
                            val1 = int(payload[i], 16)
                            i = i + 1
                            self.lowGamma = val0 * 65536 + val1 * 256 + int(payload[i], 16)
                            # midGamma:
                            i = i + 1
                            val0 = int(payload[i], 16)
                            i = i + 1
                            val1 = int(payload[i], 16)
                            i = i + 1
                            self.midGamma = val0 * 65536 + val1 * 256 + int(payload[i], 16)
                        else:
                            pass
                        i = i + 1


    def stop(self):
        # Stops a running parser thread
        if self.running == True:
            self.running = False
            self.__srl.close()

    def setCallBack(self, variable_name, callback_function):
        """Setting callback:a call back can be associated with all the above variables so that a function is called when the variable is updated. Syntax: setCallBack("variable",callback_function)
            for eg. to set a callback for attention data the syntax will be setCallBack("attention",callback_function)"""
        self.callBacksDictionary[variable_name] = callback_function

    # setting getters and setters for all variables
    # packets received
    @property
    def packetsReceived(self):
        return self.__packetsReceived

    @property
    def bytesAvailable(self, srl):
        if self.running:
            return srl.inWaiting()
        else:
            return -1

    # attention
    @property
    def attention(self):
        "Get value for attention"
        return self.__attention

    @attention.setter
    def attention(self, value):
        self.__attention = value
        # if callback has been set, execute the function
        if "attention" in self.callBacksDictionary:
            self.callBacksDictionary["attention"](self.__attention)

    # meditation
    @property
    def meditation(self):
        "Get value for meditation"
        return self.__meditation

    @meditation.setter
    def meditation(self, value):
        self.__meditation = value
        # if callback has been set, execute the function
        if "meditation" in self.callBacksDictionary:
            self.callBacksDictionary["meditation"](self.__meditation)

    # rawValue
    @property
    def rawValue(self):
        "Get value for rawValue"
        return self.__rawValue

    @rawValue.setter
    def rawValue(self, value):
        self.__rawValue = value
        # if callback has been set, execute the function
        if "rawValue" in self.callBacksDictionary:
            self.callBacksDictionary["rawValue"](self.__rawValue)

    # delta
    @property
    def delta(self):
        "Get value for delta"
        return self.__delta

    @delta.setter
    def delta(self, value):
        self.__delta = value
        # if callback has been set, execute the function
        if "delta" in self.callBacksDictionary:
            self.callBacksDictionary["delta"](self.__delta)

    # theta
    @property
    def theta(self):
        "Get value for theta"
        return self.__theta

    @theta.setter
    def theta(self, value):
        self.__theta = value
        # if callback has been set, execute the function
        if "theta" in self.callBacksDictionary:
            self.callBacksDictionary["theta"](self.__theta)

    # lowAlpha
    @property
    def lowAlpha(self):
        "Get value for lowAlpha"
        return self.__lowAlpha

    @lowAlpha.setter
    def lowAlpha(self, value):
        self.__lowAlpha = value
        # if callback has been set, execute the function
        if "lowAlpha" in self.callBacksDictionary:
            self.callBacksDictionary["lowAlpha"](self.__lowAlpha)

    # highAlpha
    @property
    def highAlpha(self):
        "Get value for highAlpha"
        return self.__highAlpha

    @highAlpha.setter
    def highAlpha(self, value):
        self.__highAlpha = value
        # if callback has been set, execute the function
        if "highAlpha" in self.callBacksDictionary:
            self.callBacksDictionary["highAlpha"](self.__highAlpha)

    # lowBeta
    @property
    def lowBeta(self):
        "Get value for lowBeta"
        return self.__lowBeta

    @lowBeta.setter
    def lowBeta(self, value):
        self.__lowBeta = value
        # if callback has been set, execute the function
        if "lowBeta" in self.callBacksDictionary:
            self.callBacksDictionary["lowBeta"](self.__lowBeta)

    # highBeta
    @property
    def highBeta(self):
        "Get value for highBeta"
        return self.__highBeta

    @highBeta.setter
    def highBeta(self, value):
        self.__highBeta = value
        # if callback has been set, execute the function
        if "highBeta" in self.callBacksDictionary:
            self.callBacksDictionary["highBeta"](self.__highBeta)

    # lowGamma
    @property
    def lowGamma(self):
        "Get value for lowGamma"
        return self.__lowGamma

    @lowGamma.setter
    def lowGamma(self, value):
        self.__lowGamma = value
        # if callback has been set, execute the function
        if "lowGamma" in self.callBacksDictionary:
            self.callBacksDictionary["lowGamma"](self.__lowGamma)

    # midGamma
    @property
    def midGamma(self):
        "Get value for midGamma"
        return self.__midGamma

    @midGamma.setter
    def midGamma(self, value):
        self.__midGamma = value
        # if callback has been set, execute the function
        if "midGamma" in self.callBacksDictionary:
            self.callBacksDictionary["midGamma"](self.__midGamma)

    # poorSignal
    @property
    def poorSignal(self):
        "Get value for poorSignal"
        return self.__poorSignal

    @poorSignal.setter
    def poorSignal(self, value):
        self.__poorSignal = value
        # if callback has been set, execute the function
        if "poorSignal" in self.callBacksDictionary:
            self.callBacksDictionary["poorSignal"](self.__poorSignal)

    # blinkStrength
    @property
    def blinkStrength(self):
        "Get value for blinkStrength"
        return self.__blinkStrength

    @blinkStrength.setter
    def blinkStrength(self, value):
        self.__blinkStrength = value
        # if callback has been set, execute the function
        if "blinkStrength" in self.callBacksDictionary:
            self.callBacksDictionary["blinkStrength"](self.__blinkStrength)
```

Appendix C. Score of *You're thinking I'm a good person*

You're thinking I'm a good person

For two instruments and electronics (~15min)

Introductory notes

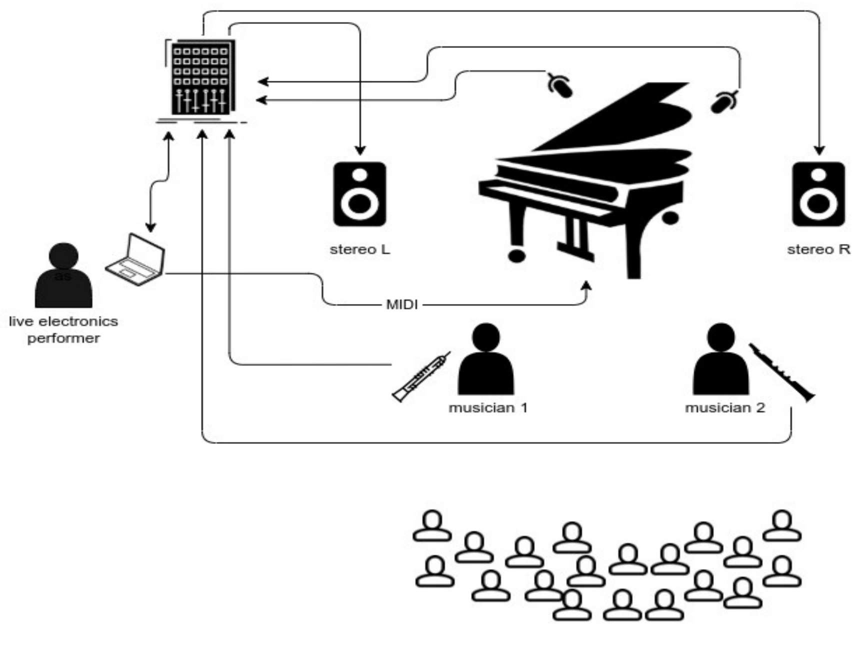
You're thinking I'm a good person is a 15-minute piece for two instruments, a player piano and live electronics. The theme of this piece is related to *social awareness*, more specifically to its most well-known feature: *empathy*. Here, two opposing currents are confronted against each other by algorithms in order to inspire the composition and the performance itself.

This piece is highly flexible in terms of instrumentation, as it is strongly based on a continuous sound approach together with some theatrical performance. Instruments, however, should be capable of producing a continuous sound in a great range with very little attack (i.e. using techniques such as circular breathing, continuous bowing, or sustained by digital effects).

Musicians with interest in free improvisation will certainly be a better match to this experience.

Scene

The figure below illustrates how the stage should be organized for the execution of this piece. The unnamed connections are all audio signals, except the MIDI communication coming from the computer to the *Disklavier*. Although not explicitly indicated in the diagram, the live electronics performer is encouraged to use an external audio device and a MIDI controller to manipulate the effects as described in the next section.



Setup requirements

Hardware

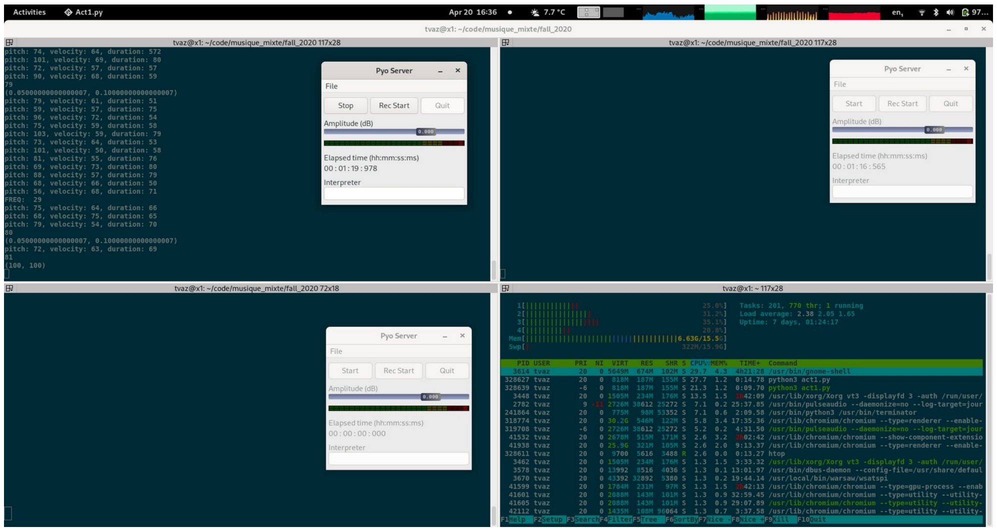
- A computer with an audio/MIDI interface
- A pair of speakers
- Stage spotlights
- A digital player piano
- Two microphones to amplify instruments 1 and 2
- Two microphones to amplify the player piano

Software

- GNU/Linux distribution (might also run on other systems)
- Python version 3 and a libraries installed as documented in the git repository located at https://github.com/tiagovaz/detaching
- Carla or similar audio plugins host with a similar effects chain as illustrated in the table bellow

Performance instructions

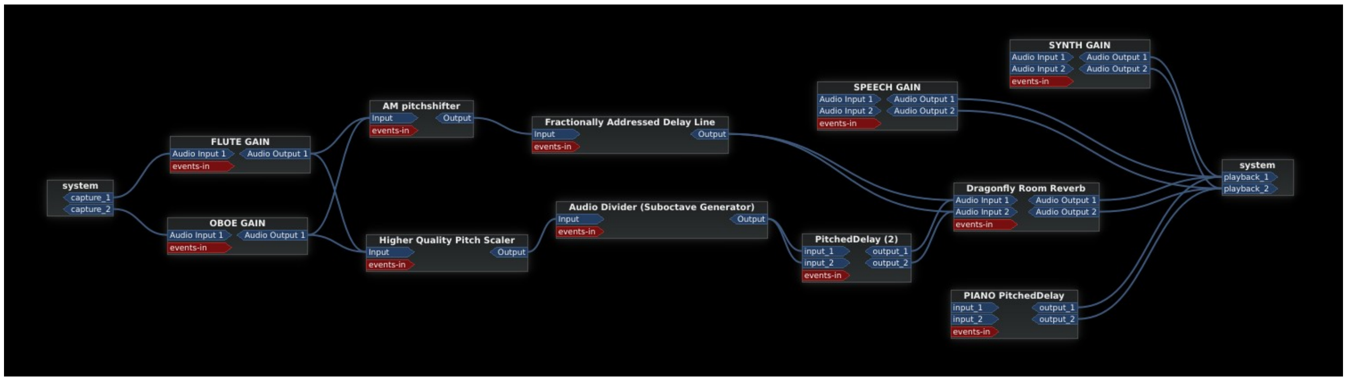
The piece is structured in 3 scenes. Each scene is controlled by a computer code to be executed by the live electronics performer operating the computer. It's recommended to open 3 terminals, one for each *Python* script. Each script can be run through the command "`python3 sceneX.py`" (where X is 1, 2 and 3). This command opens a simple interface in which the *Start/Stop* button should be called sequentially for each scene, as presented in the picture bellow:


Activities
Act1.py
tvaz@x1: ~/code/musique_mixte/fall_2020 117x28
Pyo Server
File
Stop
Rec Start
Quit
Amplitude (dB)
Elapsed time (hh:mm:ss:ms)
00 : 01 : 19 : 978
Interpreter
tvaz@x1: ~/code/musique_mixte/fall_2020 72x18
tvaz@x1: ~ 117x28

```
pitch: 74, velocity: 64, duration: 572
pitch: 101, velocity: 69, duration: 80
pitch: 72, velocity: 57, duration: 57
pitch: 90, velocity: 68, duration: 59
79
(0.05000000000000007, 0.10000000000000007)
pitch: 79, velocity: 61, duration: 51
pitch: 59, velocity: 57, duration: 75
pitch: 96, velocity: 72, duration: 54
pitch: 75, velocity: 59, duration: 58
pitch: 103, velocity: 59, duration: 79
pitch: 73, velocity: 64, duration: 53
pitch: 101, velocity: 50, duration: 58
pitch: 81, velocity: 55, duration: 76
pitch: 69, velocity: 73, duration: 80
pitch: 88, velocity: 57, duration: 79
pitch: 68, velocity: 66, duration: 50
pitch: 56, velocity: 68, duration: 71
FREQ:  29
pitch: 75, velocity: 64, duration: 66
pitch: 68, velocity: 75, duration: 65
pitch: 79, velocity: 54, duration: 70
80
(0.05000000000000007, 0.10000000000000007)
pitch: 72, velocity: 63, duration: 69
81
(100, 100)
```

```
    1[||||||||||||                           25.0%]   Tasks: 201, 770 thr; 1 running
    2[|||||||||||||||                        31.2%]   Load average: 2.38 2.05 1.65
    3[|||||||||||||||||||                    35.1%]   Uptime: 7 days, 01:24:17
    4[||||||||||                             20.8%]
  Mem[|||||||||||||||||||||||||||||||||||6.63G/15.5G]
  Swp[|                                 322M/15.9G]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
   3614 tvaz       20   0 5649M  674M  102M S 29.7  4.3  4h21:28 /usr/bin/gnome-shell
 328627 tvaz       20   0  818M  187M  155M S 27.7  1.2  0:14.78 python3 act1.py
 328639 tvaz       -6   0  818M  187M  155M S 21.3  1.2  0:09.70 python3 act1.py
   3448 tvaz       20   0 1505M  234M  176M S 13.5  1.5  1h42:09 /usr/lib/xorg/Xorg vt3 -displayfd 3 -auth /run/user/
   2782 tvaz        9 -11 2726M 38612 25272 S  7.1  0.2 25:37.85 /usr/bin/pulseaudio --daemonize=no --log-target=jour
 241864 tvaz       20   0  775M   98M 53352 S  7.1  0.6  2:09.58 /usr/bin/python3 /usr/bin/terminator
 318774 tvaz       20   0 30.2G  546M  122M S  5.8  3.4 17:35.36 /usr/lib/chromium/chromium --type=renderer --enable-
 319708 tvaz       -6   0 2726M 38612 25272 S  5.2  0.2  4:31.50 /usr/bin/pulseaudio --daemonize=no --log-target=jour
  41532 tvaz       20   0 2678M  515M  171M S  2.6  3.2  2h02:42 /usr/lib/chromium/chromium --show-component-extensio
  41938 tvaz       20   0 25.9G  321M  105M S  2.6  2.0  9:13.37 /usr/lib/chromium/chromium --type=renderer --enable-
 328611 tvaz       20   0  9700  5616  3488 R  2.6  0.0  0:13.27 htop
   3462 tvaz       20   0 1505M  234M  176M S  1.3  1.5  3:33.32 /usr/lib/xorg/Xorg vt3 -displayfd 3 -auth /run/user/
   3578 tvaz       20   0 13992  8516  4036 S  1.3  0.1 13:01.97 /usr/bin/dbus-daemon --config-file=/usr/share/defaul
   3670 tvaz       20   0 43392 32892  5380 S  1.3  0.2 19:44.14 /usr/local/bin/warsaw/wsatspi
  41599 tvaz       20   0 1784M  231M   97M S  1.3  1.5  2h42:13 /usr/lib/chromium/chromium --type=gpu-process --enab
  41601 tvaz       20   0 2088M  143M  101M S  1.3  0.9 32:59.45 /usr/lib/chromium/chromium --type=utility --utility-
  41605 tvaz       20   0 2088M  143M  101M S  1.3  0.9 29:07.89 /usr/lib/chromium/chromium --type=utility --utility-
  42112 tvaz       20   0 1435M  108M 96064 S  1.3  0.7  3:37.58 /usr/lib/chromium/chromium --type=utility --utility-
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

The live electronics performer also controls the live effects applied to instruments and to the piano, as described in the attached score. A patch file containing the enchained audio objects and MIDI assignments is provided in the repository. This file (`carla_patch.carxp`) is to be used by *Carla audio plugin host* software. Any other similar software can be used once the following chain is respected (details of each of these effects are present in the table following the image) :



The instructions in the next lines are to be presented to – and assimilated by – all musicians and the lightning operator. Musicians 1 and 2 won't be provided with a score during the performance of scenes 1 and 2. Gesture-based cues by musician 3 (the live electronics performer in the picture) can be an option for them, if necessary. In Scene 3, a music sheet is provided for musicians 1 and 2 and should be followed as described bellow.

Note: as in May 2022, a video recording is available at the Music Faculty of Université de Montréal's Youtube channel and can be used as a reference for the reproduction of the piece: https://www.youtube.com/watch?v=47xVxk-pEM4

Scene 1

- Lights off. The TTS-driven voice introduces the piece by proclaiming the ***Speech #1***
- Musicians 1 and 2 stand side-by-side in front of the Disklavier. They remain quiet; no reaction is expected.
- ***Speech #2*** is played.
- After about 10 seconds, lights progressively on the piano spot.
- Piano plays an algorithmic music and go *crescendo* for about a minute. Musicians stay seated looking at void audience.
- ***Speech #3***

Scene 2

- ***Speech #2*** restarts: "Empathy works..."
- Regular lights on musicians during the speech.
- Same speech is presented 3 times and gets interrupted. After the third time (about 1min15s), musicians wait for a few seconds and start playing an air sound, as if they were trying to perform a proper sound. They play as if the other didn't exist.
- At some point (about 2min30s) the first empathy statement is presented. Musicians start to improvise, keeping no musical dialogue between them. They can play short, noisy, and contrasting sounds, as well as give some time to silence or degrading little phrases from classical repertoire, as if trying to succeed a performance to the audience.
- At some point, after about 5min, a loud piano note is be played. From this time on, musicians stand up and play more intensively, trying to overlap the other musicians' sound, as a subtile competition.
- Once voices (***Speech #4, #5 and #6)*** start overlapping each other, musicians stop playing and leave the scene.

Scene 3

- Low lights spot the musicians. After about 10 seconds, regular lights spot the piano.
- ***Speech #7***: "You must be so helpless, think of Mary..."
- When speech starts, musicians slowly bring their music stands with scores and place them backing each other (so that musicians face each other).
- After the speech, the piano starts playing short notes. Musicians start getting ready to play.
- Hight lights towards musicians as soon as they seem to be ready to play.
- At some point, the piano opens the sustain pedal. From that time on, musicians wait about 10 seconds, quickly look at each other, then start playing their score.
- Right after they finish playing their parts, they leave the scene, put their instruments aside, come back, take the score off the stand, take the stand with another hand and leave the scene with them. This whole process should suggest a total indifference to the music, and to each other; despite the fact that the music is kept playing due to the digital effects.
- ***Speech #8***
- About 10 seconds after musicians leave the scene, musicians lights slowly go off, followed by piano lights going off.
- After another 10 seconds, all sound goes off.

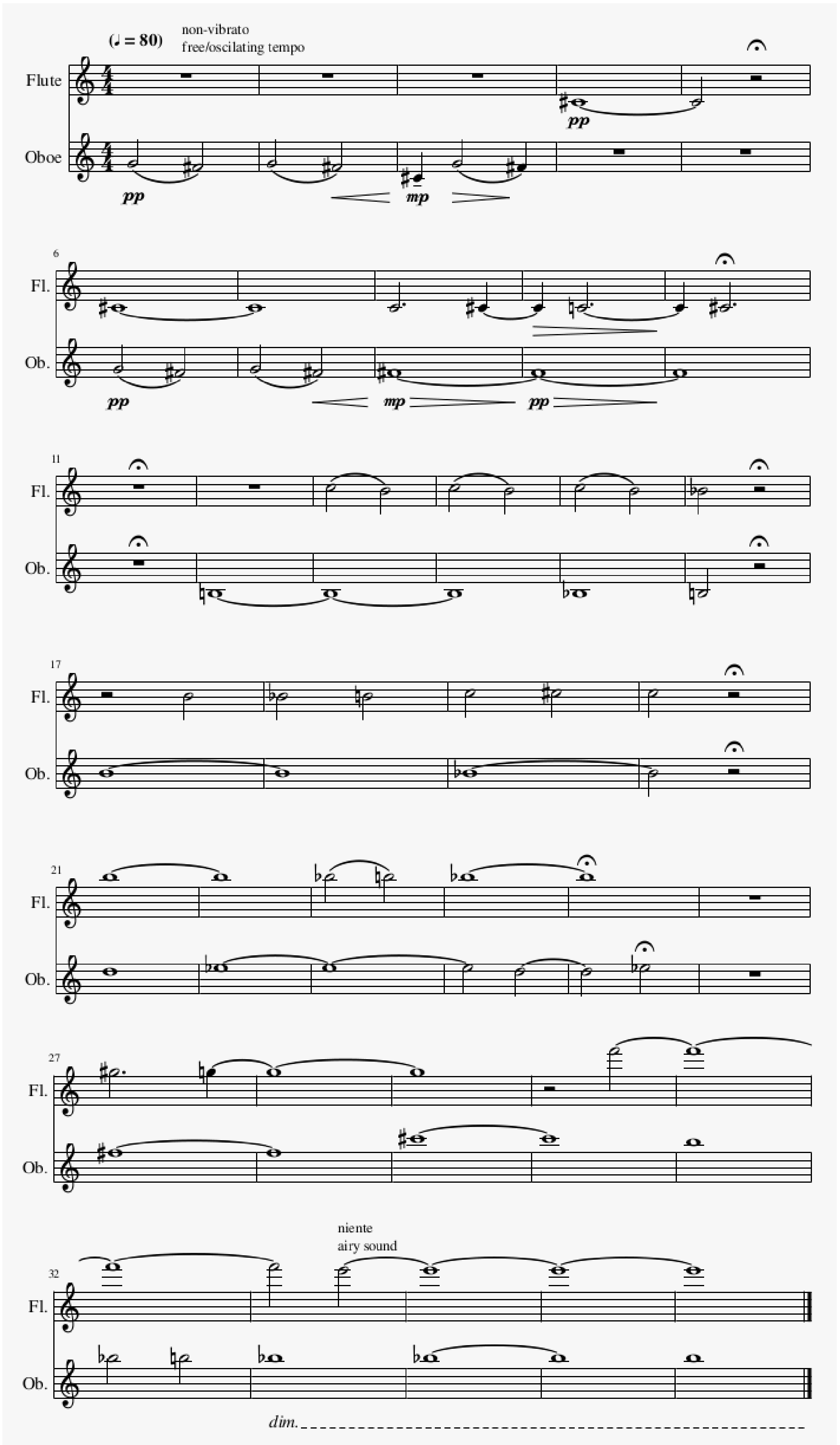
“44 empathy statements that will make you the greatest listener”

#	Statement	#	Statement
01	You’re making total sense.	23	You are in a lot of pain here. I can feel it.
02	I understand how you feel.	24	It would be great to be free of this.
03	You must feel so hopeless.	25	That must have annoyed you.
04	I just feel such despair in you when you talk about this.	26	That would make me mad too.
05	You’re in a tough spot here.	27	That sounds frustrating.
06	I can feel the pain you feel.	28	That is very scary.
07	The world needs to stop when you’re in this much pain.	29	Well I agree with most of what you’re saying.
08	I wish you didn’t have to go through that.	30	I would have also been disappointed by that.
09	I’m on your side here.	31	That would have hurt my feelings also.
10	I wish I could have been with you in that moment.	32	That would make me sad too.
11	Oh, wow, that sounds terrible.	33	POOR BABY!
12	You must feel so helpless.	34	Wow, that must have hurt.
13	That hurts me to hear that.	35	I understand what you’re feeling.
14	I support your position here.	36	You are making a lot of sense to me.
15	I totally agree with you.	37	Okay, I think I get it. So what you’re feeling is…
16	You are feeling so trapped!	38	Let me try to paraphrase and summarize what you’re saying.
17	You are making total sense.	39	You’re saying…
18	That sounds like you felt really disgusted!	40	I would have trouble coping with that.
19	No wonder you’re upset.	41	What I admire most about what you’re doing is…
20	I’d feel the same way you do in your situation.	42	That would make me feel insecure.
21	I think you’re right.	43	That sounds a little frightening.
22	I see. Let me summarize: What you’re thinking here is…	44	Tell me what you see as your choices here.

AI-generated texts used in the piece (in order of appearance)

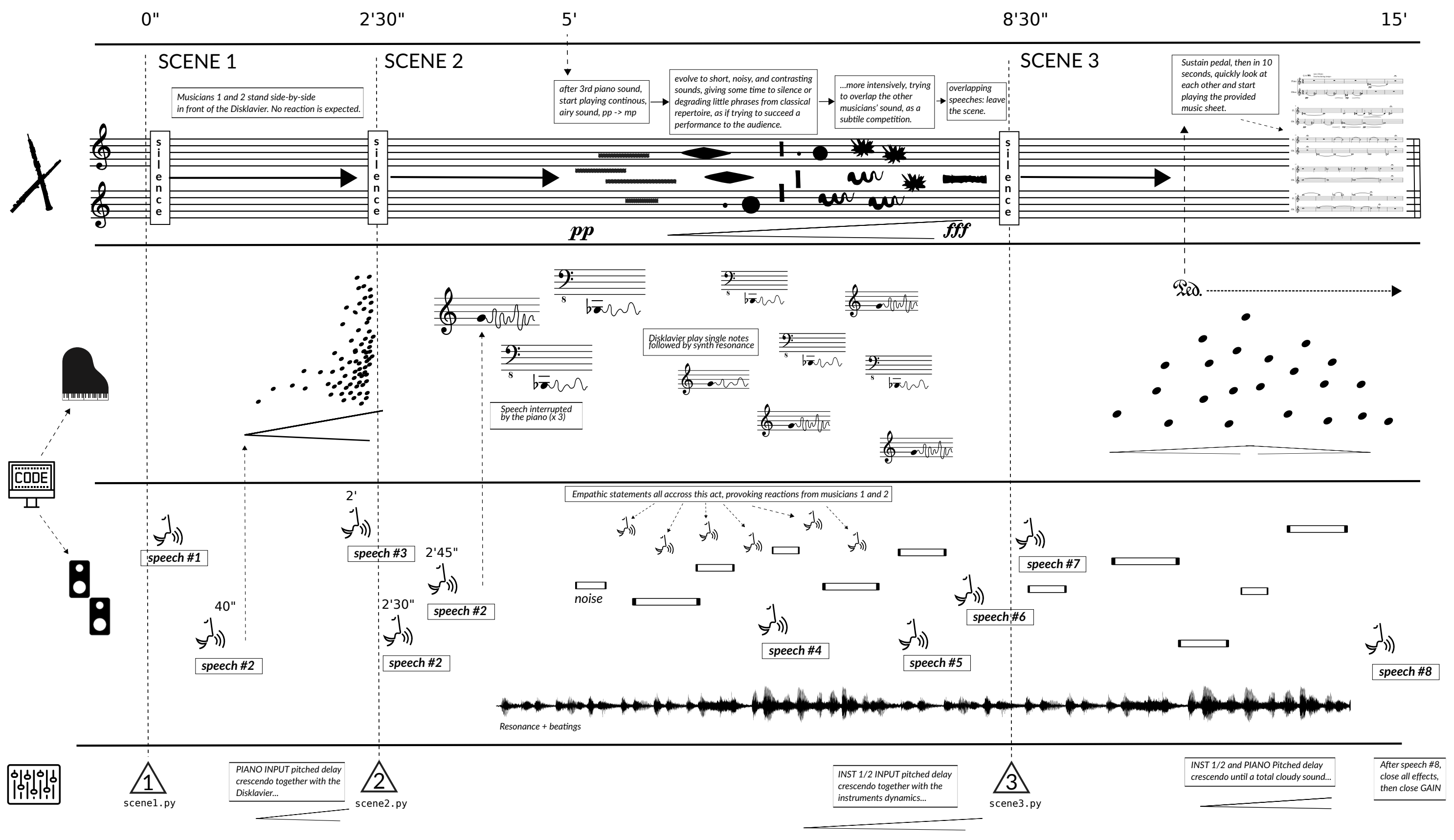
#	Text	TTS wav file
01	The first nearly reliable way that emotions can arise is from direct experience with others. You'll remember the emotion you're feeling in this session if you experienced the same emotion in a previous session. Activation of your empathic awareness—and possible neurologic and neurological effects—in others can affect your ability to quality-quantify comebacks for others in your life.	*None* *(text used in the video only)*
02	Empathy works by increasing the activation of your empathic awareness in others, which literally increases the effort of positive empathy work on their part. Thus, if you activate empathy near others, empathically speaking, this will help them feel more closely and empathically attached to you. It shouldn't be surprising then, that empathy interactions also have a direct relationship to empathy intensity.	`intro.wav`
03	*You're making total sense*, note me. Let's begin again, shall we?	`restart.wav`
04	*Oh, wow, that sounds terrible.* Don't waste your time learning style by imitating Beyoncé or learn to enliven yourself with sensory terms like great'Ease, Adorable, Mellonicious! — because these styles don't exist yet. There will always be trouble on the underground level — even within the own organization, and nobody with CREEPY IS BOYS (okay, so maybe Beyoncé isn't some do-it-right revolutionary self).	`beyonce.wav`
05	*You must feel so helpless.* Think of Mary, whom she will hold most dear: Action, noise, retreat, purpose, right and wrong, fear, wonder, grief, gratitude, devotion, status, perverted idealism, boundary usage, guilt, wrongdoing, beautiful imaginations, planned states, imaginal behavior, finds ahead, offered knowledge, dietary patterns with dear remembering, ancestors, grandchildren, singing, drama, fencing (fat cats killed babies's babies completely uncountably), attacks. Victimization. Victimization. Victimization.	`mary.wav`
06	*I see. Let me summarize: What you're thinking here is* that you can make money from doing nothing, and the only way to make money is to sell your services to companies. You have a very good point. The problem is that your idea is an idea that is not particularly interesting. It's an idea that does not make any money. It's a bad idea. That's why you're not getting rich, and that's why you're not doing anything interesting.	`money.wav`
07	Another way that emotions can happen in a crowd is from previously unspecific empathic reactions to stimuli. Annie, an 18-year-old classically trained musician, got her first emergency beat early, spooked by the sounds.	`18.wav`
08	*I see. Let me summarize: What you're thinking here is* that I'm not so bad. No. You're thinking I'm a shit-stain. You're thinking I'm a terrible person. You're thinking I'm a creep. You're thinking I'm a monster. You're thinking I'm a fucking lunatic. You're thinking I'm a dumbass. No. You're thinking I'm a good person. You're thinking I'm a good person. You're thinking I'm a good person.	`insult.wav`

Proposed score for Scene 3 (can be adapted as needed)



You're thinking I'm a good person

Tiago Vaz, 2020



Musicians 1 and 2 stand side-by-side in front of the Disklavier. No reaction is expected.

after 3rd piano sound, start playing continous, airy sound, pp -> mp

evolve to short, noisy, and contrasting sounds, giving some time to silence or degrading little phrases from classical repertoire, as if trying to succeed a performance to the audience.

...more intensively, trying to overlap the other musicians' sound, as a subtile competition.

overlapping speeches: leave the scene.

Sustain pedal, then in 10 seconds, quickly look at each other and start playing the provided music sheet.

Speech interrupted by the piano (x 3)

Disklavier play single notes followed by synth resonance

Empathic statements all accross this act, provoking reactions from musicians 1 and 2

PIANO INPUT pitched delay crescendo together with the Disklavier...

INST 1/2 INPUT pitched delay crescendo together with the instruments dynamics...

INST 1/2 and PIANO Pitched delay crescendo until a total cloudy sound...

After speech #8, close all effects, then close GAIN

scene1.py

```python
import random
from pyo import *
from instruments import Speech
import subprocess

pm_list_devices()

s = Server(audio='jack', duplex=0, nchnls=2)

# Open all MIDI output devices.
s.setMidiOutputDevice(99)

# Then boot the Server.
s.boot()

speech_intro = Speech(['intro.wav'], loop=0)
speech_intro.play()

# close pedal
s.ctlout(64, 0)

# set random-ish pattern time
pat_time = XnoiseDur(dist=11, min=15, max=20)
speech = Speech(['restart.wav'])

time_counter = 0
def time_events():
    global s, time_counter, pat_time, pat
    time_counter = time_counter + 1
    print(time_counter)
    print((pat_time.min, pat_time.max))

    d = random.choice([0,1])
    if d == 1:
        s.ctlout(64, 0)
    else:
        s.ctlout(64, 127)

    if time_counter == 10:
       pat_time.max = 10
       pat_time.min =  5

    if time_counter == 20:
        pat_time.max = 5
        pat_time.min = .5

    if time_counter > 50 and pat_time.min > .1:
        pat_time.max = pat_time.min
        pat_time.min = pat_time.min - .05

    if time_counter == 50:
        s.ctlout(64, 127)

    if time_counter == 80:
        pat_time.max = 100
        pat_time.min = 100
        vel = 50
        dur = 2000
        # Mega chord to end
        s.makenote(pitch=20, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=21, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=22, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=23, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=24, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=25, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=26, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=27, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=28, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=29, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=40, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=81, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=82, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=83, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=84, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=85, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=86, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=87, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=88, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=89, velocity=vel, duration=dur, channel=1)

    if time_counter == 85:
        speech.play()

    if time_counter == 90:
        s.ctlout(64, 0)
        pat.stop()

# Actual time counter
global_time = Pattern(time_events, 1).play()

pitch = Phasor(freq=1, mul=48, add=40)

count = 0
mul_count = 0
freq_count = 0

def midi_event():
    global count, mul_count, pitch, freq_count, s

    pit = int(pitch.get())

    # each 23 seconds
    mul_count = mul_count + 1
    if mul_count == 23:
        pitch.add = pitch.add + 1
        print("MUL: ", pitch.mul)
        mul_count = 0;

    # each 35 seconds
    freq_count = freq_count + 1
    if freq_count == 35:
        pitch.freq = random.randint(1,30)
        print("FREQ: ", pitch.freq)
        freq_count = 0;

    if count == 0 and random.randint(0,1) < .5: # half chance
        vel = random.randint(40, 70)
        dur = random.randint(9,2000)
        #chord
        s.makenote(pitch=pit+12, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=pit+14, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=pit+16, velocity=vel, duration=dur, channel=1)
    else:
        vel = random.randint(50, 80)
        dur = random.randint(50, 80)
        s.makenote(pitch=pit, velocity=vel, duration=dur, channel=1)

    count = (count + 1) % random.randint(12,13)

    print("pitch: %d, velocity: %d, duration: %d" % (pit, vel, dur))

def start_pat():
    global pat
    pat.play()

# Generates a MIDI event every 125 milliseconds.
pat = Pattern(midi_event, pat_time)
a = CallAfter(start_pat, 30)

s.gui(locals())
```

scene2.py

```
from pyo import *
import random
import os
from instruments import *

s = Server(audio='jack', duplex=0, nchnls=2)
s.setMidiOutputDevice(99)
s.boot()

m = MyFreezing()
m2 = MyFreezing()
m3 = MyFreezing()
m.stop()
m2.stop()
m3.stop()

# Open pedal
s.ctlout(64, 127)

################# BEGIN  GESTURE 00 ##################

intro_speech = Speech(['intro.wav'])
sines = IntroSines()

def g00():
    global intro_speech
    intro_speech.play()

g00Time = Metro(time=Randi(31, 39)).stop()
g00Func = TrigFunc(g00Time, g00)

################# BEGIN  GESTURE 01 ##################

piano_flag = True

def g01():
    global piano_flag
    sines.play()
    intro_speech.stop()
    if piano_flag == True:
       s.makenote(pitch=22, velocity=random.randint(30, 45), duration=20000)
       s.makenote(pitch=79, velocity=random.randint(60, 70), duration=20000)
       s.makenote(pitch=91, velocity=random.randint(70, 90), duration=20000)
    m.pvb.setPitch(random.uniform(0.90, 1.1))
    m.refresh()

g01Time = Metro(time=Randi(20, 35)).stop()
g01Func = TrigFunc(g01Time, g01)

```

```
################# BEGIN  GESTURE 02 ##################

high = HighFreq(mul=.05)

def g02():
    global high
    high.play()

g02Time = Metro(time=Randi(10, 30)).stop()
g02Func = TrigFunc(g02Time, g02)

################# BEGIN  GESTURE 03 ##################

snoise = SmoothNoise(mul=.25, dur=0.8)

def g03():
    global snoise
    snoise.play()

g03Time = Metro(time=Randi(10, 30)).stop()
g03Func = TrigFunc(g03Time, g03)

################# BEGIN  GESTURE 04 ##################

def g04():
    s.makenote(pitch=22, velocity=random.randint(60, 70), duration=20000)
    s.makenote(pitch=79, velocity=random.randint(70, 90), duration=20000)
    s.makenote(pitch=91, velocity=random.randint(90, 100), duration=20000)
    m.pvb.setPitch(random.uniform(0.9, 1.1))
    m2.pvb.setPitch(random.uniform(0.9, 1.1))
    m3.pvb.setPitch(random.uniform(0.9, 1.1))
    m.refresh()
    m2.refresh()
    m3.refresh()
    # send midi note

g04Time = Metro(time=Randi(20, 25)).stop()
g04Func = TrigFunc(g04Time, g04)

##################### SCORE #######################

time = -1

# Random speech to be called
speech_random = Speech(soundfile=os.listdir("44_statements"))
speech_random_time = Metro(time=Randi(25, 40)).stop()
speech_random_func = TrigFunc(speech_random_time, speech_random.play)

# Random speech to be called 2
speech_random2 = Speech(soundfile=os.listdir("44_statements"))
speech_random_time2 = Metro(time=Randi(10, 23)).stop()
```

```
speech_random_func2 = TrigFunc(speech_random_time2, speech_random2.play)

# GPT2 texts right before interlude
interlude_text = Speech(os.listdir('texts_speech'))
interlude_text_time = Metro(time=Randi(25, 40)).stop()
interlude_text_func = TrigFunc(interlude_text_time, interlude_text.play)

def score():
    global time, m, m2, m3, interlude_text, piano_flag, g01Time
    time += 1
    high.setDur(random.uniform(0.5, 1.5))
    snoise.setDur(random.uniform(0.5, 1.5))

    if time == 1:
        print(time)
        m.play()
        g00Time.play()

    if time == 50:
        print(time)
        m2.play()
        g01Time.play()

    if time == 80:
        print(time)
        g00Time.stop()

    if time == 120:
        print(time)
        piano_flag = False

    if time == 140:
        print(time)
        speech_random_time.play()

    ## Two minutes no piano only flute and voice
    if time == 200:
        print(time)
        g04Time.play() # starts new piano with low notes
        g01Time.stop() # stops initial piano
        speech_random_time.stop()
        m3.play()

    if time == 260:
        print(time)
        g02Time.play() # high pitch

    if time == 270:
        print(time)
        interlude_text_time.play()

```

```python
    if time == 280:
        print(time)
        g04Time.stop() # stops all piano
        g03Time.play()  # snoise
        speech_random_time.setTime(Randi(5, 10)) # overlapping voices
        speech_random_time.play() # overlapping voices

    # stop everything but high/snoise and call the serial (part3) script
    if time == 300:
        speech_random_time2.play()

    if time == 315:
        vel = 50
        dur = 2000

        s.ctlout(64, 127)

        s.makenote(pitch=20, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=22, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=24, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=26, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=28, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=80, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=82, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=84, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=86, velocity=vel, duration=dur, channel=1)
        s.makenote(pitch=88, velocity=vel, duration=dur, channel=1)

        print(time)
        m2.stop()
        m3.stop()
        m.stop()
        sines.stop()
        g01Time.stop()
        g02Time.stop()
        g03Time.stop()
        speech_random_time.stop()
        speech_random_time2.stop()
        interlude_text_time.stop()

mainTime = Metro(time=1).play()
mainFunc = TrigFunc(mainTime, score)

s.gui(locals())
```

scene3.py

```python
import random
from pyo import *
from instruments import *
import time

pm_list_devices()

s = Server(audio='jack', duplex=0, nchnls=2)

# Open all MIDI output devices.
s.setMidiOutputDevice(99)

# Then boot the Server.
s.boot()

s.ctlout(64, 127)
speech_start = Speech(['mary.wav'], loop=True)
speech_start.play()

# Kinderstuck serial sequence
notes_seq = [3, 4, 0, 11, 10, 1, 2, 9, 8, 7, 6, 5]

index = 0
index2 = 0
index3 = 0
pedal_flag = True

def intro_event():
    global s, pat, speech_start
    # close pedal
    s.ctlout(64, 0)
    pat.play()
    speech_start.stop()

def midi_event():
    global notes_seq, index, pat2, pat, s

    index = index + 1
    n, d = divmod(index, 12)
    print(index, n, d)

    vel = random.randint(25, 35)
    dur = random.randint(20, 1000)

    octave = random.choice([48, 60])
    s.makenote(pitch=notes_seq[d]+octave, velocity=vel, duration=dur, channel=1)
    if n == 1:
        s.makenote(pitch=notes_seq[d]+octave+12, velocity=vel, duration=dur, channel=1)

    print("pitch: %d, velocity: %d, duration: %d" % (notes_seq[d], vel, dur))

    if n == 2:
        pat2.play()
        final_event2Time.play()

event2_part2 = 0
event2_flag = False

def midi_event2():
    global notes_seq, index2, pat2, pat, event2_part2, event2_flag, pedal_flag
    if pedal_flag == True:
        s.ctlout(64, 127)
        pedal_flag = False

    vel = random.randint(20, 30)
    dur = 100

    octave = random.choice([60, 72])
    s.makenote(pitch=notes_seq[index2]+octave, velocity=vel, duration=dur, channel=1)
    if event2_flag == True:
        s.makenote(pitch=notes_seq[index2]+octave-14, velocity=vel, duration=dur, channel
    =1)

    print("pitch: %d, velocity: %d, duration: %d" % (notes_seq[index2], vel, dur))

    index2 = index2 + 1
    if index2 == 12:
        final_eventTime.play()
        index2 = 0
        event2_flag = True

    event2_part2 = event2_part2 + 1
    if event2_part2 == 48:
        pat3.play()

speech_final = Speech(['insult.wav'])
def midi_event3():
    global notes_seq, index3, pat3, speech_final

    vel = random.randint(20, 30)
    dur = 100

    octave = random.choice([24, 84, 96])
    s.makenote(pitch=notes_seq[index3]+octave, velocity=vel, duration=dur, channel=1)
    s.makenote(pitch=notes_seq[index3]+octave-14, velocity=vel, duration=dur, channel=1)
    s.makenote(pitch=notes_seq[index3]+octave-16, velocity=vel, duration=dur, channel=1)
    print("pitch: %d, velocity: %d, duration: %d" % (notes_seq[index3], vel, dur))

    index3 = index3 + 1
    if index3 == 12:
        index3 = 0
        pat.stop()
        pat2.stop()
        pat3.stop()
        speech_final.play()

snoise = SmoothNoise(mul=.25, dur=0.3)
high = HighFreq(mul=0.5)

def final_event():
    global high
    high.setDur(random.uniform(0.3, 0.6))
    high.play()

final_eventTime = Metro(time=Randi(10, 20)).stop()
final_eventFunc = TrigFunc(final_eventTime, final_event)

def final_event2():
    global snoise
    snoise.setDur(random.uniform(0.3, 0.6))
    snoise.play()

final_event2Time = Metro(time=Randi(10, 20)).stop()
final_event2Func = TrigFunc(final_event2Time, final_event2)

# set random-ish pattern time
pat_time = XnoiseDur(dist=11, min=.1, max=8)
pat = Pattern(midi_event, pat_time)

# set random-ish pattern time
pat_time2 = XnoiseDur(dist=11, min=0.5, max=10)
pat2 = Pattern(midi_event2, pat_time2)

# set random-ish pattern time
pat_time3 = XnoiseDur(dist=11, min=.1, max=4)
pat3 = Pattern(midi_event3, pat_time3)

a = CallAfter(intro_event, random.randint(30,40))

s.gui(locals())
```

instruments.py

```python
import random
from pyo import *

########### INSTRUMENTS ##########################

class Speech():
    def __init__(self, soundfile=[], loop=False, mul=.5, fadein=.01, fadeout=.01, duration=0, chnl=0, inc=1):
        self.amp = Fader(fadein=fadein, fadeout=fadeout, dur=duration, mul=mul)
        self.chnl = chnl
        self.inc = inc
        self.soundfile = soundfile
        self.soundfile_to_play = random.choice(soundfile)
        self.player = SfPlayer(self.soundfile_to_play, mul=[self.amp/2., self.amp/1.95], loop=loop).stop()
        self.player_rev = Freeverb(self.player, size=[.3,.25], damp=.6, bal=.4, mul=.8).out(chnl=self.chnl, inc=self.inc)

    def setDur(self, dur):
        self.amp.dur = dur
        return self

    def play(self):
        self.player.setSound(random.choice(self.soundfile))
        self.player.play()
        self.amp.play()
        return self

    def stop(self):
        self.amp.stop()
        return self

    def getOut(self):
        return self.amp

class IntroSines():
    def __init__(self, freq=[3000, 3000.01, 3000.03], harms=400, mul=.8):
        self.amp = Fader(fadein=10, fadeout=10, dur=0, mul=mul)
        self.sines = Blit(freq=freq, harms=harms, mul=self.amp * .01).out()
        self.rev = Freeverb(self.sines, size=.84, damp=.87, bal=.9, mul=self.amp * .2).out()

    def setDur(self, dur):
        self.amp.dur = dur
        return self

    def play(self):
        self.amp.play()
        return self

    def stop(self):
        self.amp.stop()
        return self

    def getOut(self):
        return self.amp


class HighFreq():
    def __init__(self, freq=[11200, 11202], dur=.4, mul=.4):
        self.amp = Fader(fadein=.01, fadeout=.01, dur=dur, mul=mul)
        self.sine = SineLoop(freq=freq, mul=self.amp * .05).out()
        self.rev = Freeverb(self.sine, size=.84, damp=.87, bal=.9, mul=self.amp * .2).out
    ()

    def setDur(self, dur):
        self.amp.dur = dur
        return self

    def play(self):
        self.amp.play()
        return self

    def stop(self):
        self.amp.stop()
        return self

    def getOut(self):
        return self.amp

class SmoothNoise():
    def __init__(self, dur=1.3, mul=.4):
        self.amp = Fader(fadein=.1, fadeout=.01, dur=dur, mul=mul)
        self.noise = PinkNoise(self.amp * .01).mix(2).out()

    def setDur(self, dur):
        self.amp.dur = dur
        return self

    def play(self):
        self.amp.play()
        return self

    def stop(self):
        self.amp.stop()
        return self

    def getOut(self):
        return self.amp

    def setInput(self, x, fadetime=.001):
        self.input.setInput(x, fadetime)

class MyFreezing():
    def __init__(self, mul=1):
        global s
        f = 'sound_bank/444166__cloe-king__wine-glass-ring.wav'
        f_len = sndinfo(f)[1]
        #s.startoffset = f_len

        self.globalamp = Delay(Fader(fadein=100, dur=0).play(), delay=f_len, maxdelay=
    f_len)

        src = SfPlayer(f, loop=True, mul=0.8)

        # When this number increases, more analysis windows are randomly used.
        spread = Sig(0.1, mul=0.1)

        # The normalized position where to freeze in the sound.
        index = Sig(0.25, add=Noise(spread))

        self.pva = PVAnal(src, size=4096, overlaps=8)
        self.pvb = PVBuffer(self.pva, index, pitch=1.02)
        self.pvv = PVVerb(self.pvb, revtime=0.999, damp=0.995)
        self.pvs = PVSynth(self.pvv, mul=0.3)
        self.rev = STRev(self.pvs, roomSize=1, revtime=1)
        self.outsig= Delay(self.rev, delay=.1, feedback=0.2, mul=self.globalamp * mul).
    stop()

    def play(self):
        self.pvb.play()
        self.outsig.out()

    def stop(self):
        self.outsig.stop()

    def refresh(self):
        self.play()
```

Appendix D. Score of *You can't un-neighbor your neighbor*

You can't un-neighbor your neighbor

For player piano and electronics (~15min)

Introductory notes

"Most personalized filters are based on a three-step model. First, you figure out who people are and what they like. Then, you provide them with content and services that best fit them. Finally, you tune to get the fit just right. Your identity shapes your media. There's just one flaw in this logic: Media also shape identity. And as a result, these services may end up creating a good fit between you and your media by changing. . . you."

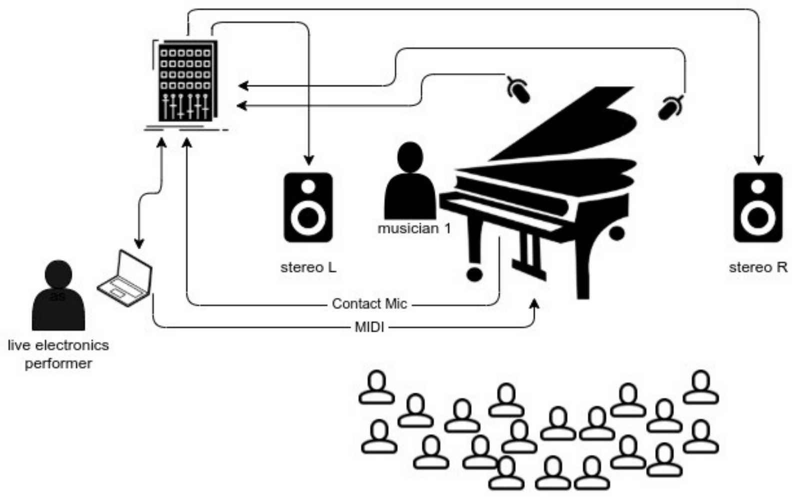
"The Filter Buble", Eli Parisier, 2011

You can't un-neighbor your neighbor is a 15-minute piece composed in 3 scenes. The piece requires a prepared piano player (i. e. A *Disklavier)*, a computer, a pair of speakers, a pianist, and a live electronics performer. The player piano is simultaneously controlled by a live-generated *performance-rnn* and by a live electronics performer manipulating a set of algorithms. The *performance-rnn* is generated through a provided *Python* script which uses a pre-trained *Magenta PerformanceRNN* model to produce the MIDI files.

The pianist follows a minimal set of instructions for preparing and playing the piano together with the algorithms, all throughout the performance.

Scene

The figure below illustrates how the stage should be organized for the execution of this piece. Although not explicitly indicated in the diagram, the live electronics performer is encouraged to use an external audio device and a MIDI controller to manipulate both the player piano and the effects, as described in the next section.



Setup requirements

Hardware

- A computer with an audio/MIDI interface
- A MIDI controller
- A pair of speakers
- A digital player piano (i. e. a *Disklavier*)
- Two microphones to amplify the player piano (on the soundboard)
- Two contact microphones to amplify the mechanical structure under the player piano

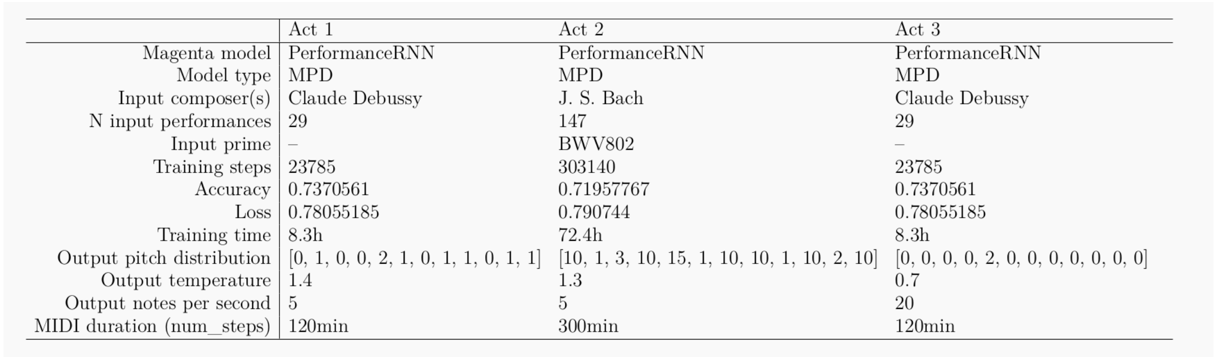
Software

- GNU/Linux distribution (might also run on other systems)
- Python version 3 and libraries installed as documented in the git repository located at https://github.com/tiagovaz/detaching
- Carla or similar audio plugins host with a similar effects chain as illustrated in the table bellow

General instructions

Once all hardware and software are set up by the instructions provided in the repository, two steps should be followed for the performance to take place:

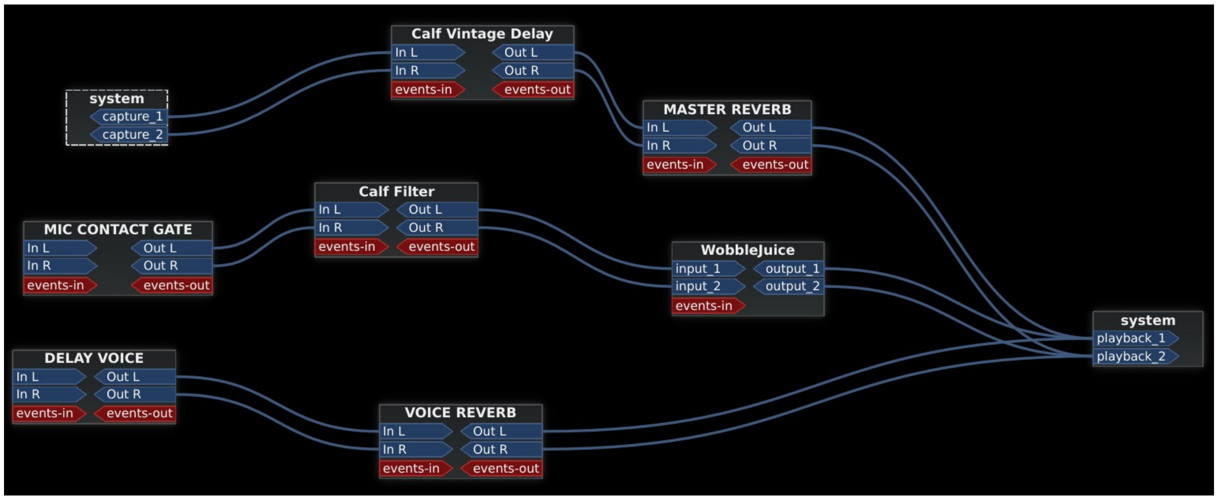
1. **Generate the *performance-rnn* MIDI sequences to be played on the piano.** In this step, all the needed MIDI files are generated through the pre-trained models available in a bundle format (“`bach.mag`” and “`debussy.mag`”). The *Python* script which generates these files is the “`performance_generator.py`” and should be called without parameters. It will generate 10 *performances-rnn* for each scene. The characteristics of each model used by this script (and how they are used to generate sequences for each scene) is illustrated in the table bellow:



	Act 1	Act 2	Act 3
Magenta model	PerformanceRNN	PerformanceRNN	PerformanceRNN
Model type	MPD	MPD	MPD
Input composer(s)	Claude Debussy	J. S. Bach	Claude Debussy
N input performances	29	147	29
Input prime	–	BWV802	–
Training steps	23785	303140	23785
Accuracy	0.7370561	0.71957767	0.7370561
Loss	0.78055185	0.790744	0.78055185
Training time	8.3h	72.4h	8.3h
Output pitch distribution	[0, 1, 0, 0, 2, 1, 0, 1, 1, 0, 1, 1]	[10, 1, 3, 10, 15, 1, 10, 10, 1, 10, 2, 10]	[0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0]
Output temperature	1.4	1.3	0.7
Output notes per second	5	5	20
MIDI duration (num_steps)	120min	300min	120min

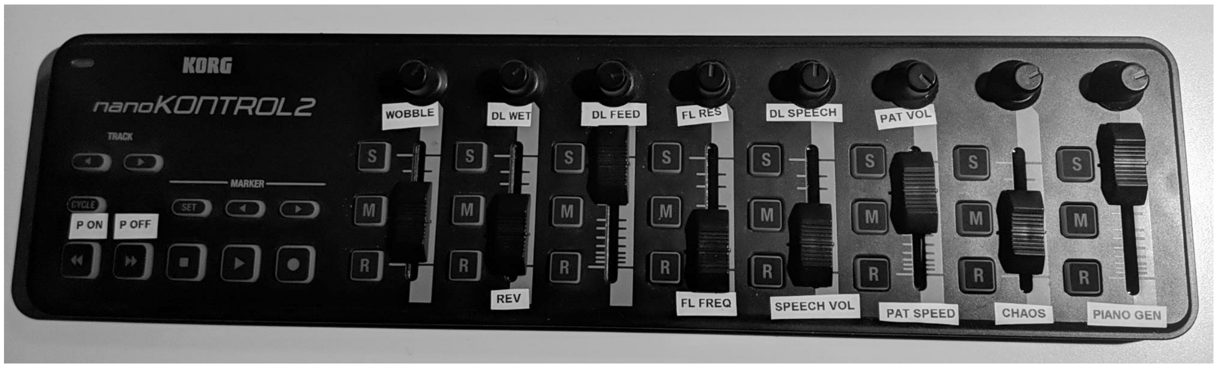
2. **Run the performance code.** A parameter indicating the number of the scene (1, 2, or 3) should be provided to the script “performance`.py`”. For instance, for scene 1: “`python3 performance.py 1`”. Once the code starts running, the live electronics performer controls the effects and, to some extent, the MIDI data being sent to the piano, according to the attached score. A *XML* file containing the configuration of audio objects and MIDI

assignments is provided in the repository ("`carla_patch.carxp`"). This file is to be used by *Carla audio plugin host* software. Any other similar software can be used as long as the following chain of effects is respected:



The audio effects in *Carla* can be assigned to a MIDI controller through its graphical interface. The MIDI manipulation can be assigned by editing the "`performance.py`" script, as documented in the code, by attributing the CC values to the variables listed in the above table.

The following picture and table might be useful as a guidance for this configuration, which supports a *nanoKONTROL2* controller by *Korg* and was used in the video-recorded performance available in https://archive.org/details/you-cant-uneighbor-your-neighbor.



Label	Source	Var name / plugin name	Action
P ON	Python code	PEDAL_ON	*Disklavier* sustain pedal ON
P OFF	Python code	PEDAL_OFF	*Disklavier* sustain pedal OFF
WOBBLE	Carla	*Wobble Juice Plugin*	Piano mechanics *Wobblejuice* effect gain
DL WET	Carla	*Calf Vintage Delay*	Piano soundboard delay dry/wet
REV	Carla	*Calf Reverb*	Piano soundboard reverb gain
DL FEED	Carla	*Calf Vintage Delay*	Piano soundboard delay feed

FL RES	Carla	*Calf Filter*	Piano mechanics filter resonance
FL FREQ	Carla	*Calf Filter*	Piano mechanics filter frequence
DL SPEECH	Carla	*Calf Vintage Delay*	Speech (scene 3 only) delay dry/wet
SPEECH VOL	Carla	*Carla Input Gain*	Speech volume (scene 3 only)
PAT VOL	Python code	PATTERN_VOL	Controls the MIDI velocity average of algorithmic musical patterns sent to the *Disklavier*
PAT SPEED	Python code	PATTERN_SPEED	Controls the density (or *tempo*) of MIDI data of algorithmic musical patterns sent to the *Disklavier*
CHAOS	Python code	CHAOS	Adds chaos (random notes) to the *performance-rnn* sequence being played on the *Disklavier*
PIANO GEN	Python code	PIANO_GEN	Skips notes (replaces by silence) from the *performance-rnn* sequence being played on the *Disklavier*

Per scene instructions

This is a highly improvised piece, where both the pianist and the live electronics performer follow very little instructions in order to accompany the AI-driven part being played. A sub-theme taken from the quotation in the beginning of this score reflects the composer's intentions and should be kept in mind as inspiration.

The following instructions are nothing more than an open guide for the execution of the piece. This comes from a collaborative work between composer Tiago Vaz and pianist Daniel Áñez, during rehearsals and the performance recorded in December 2021 (https://archive.org/details/you-cant-uneighbor-your-neighbor).

Scene 1 (3-4 min)

"First, you figure out who people are and what they like."

This scene starts with a performance-rnn generated from a model trained against a dataset of 29 pieces from Claude Debussy. It plays on average 5 notes per second, has a temperature of 1.4 – being the most "creative" among the three scenes – and follows a pitch distribution represented as [0, 1, 0, 0, 2, 1, 0, 1, 1, 0, 1, 1]. This distribution is structured as a *Python* list, in wich numbers represent the probability of a note being played, from C to B. In this scene C = 0, C# = 1, D = 0, D# = 2 and so forth. Given the hight level of "creativity", these numbers are approximative. So, C = 0 doesn't necessary mean "C is not to be played". Instead, it can be read as "C is much less likely to be played than D# - but still possible to be played".

This scene is resonant, slow *tempo* and rich in wide-range chords. The musicians are encouraged to attentively observe the sounds and try to *extend* them rather than adding new sounds. A natural direction to long/continuous sounds should be followed. In the recorded video, this was accomplished with the aid of *e-bows* resonating piano strings on the soundboard. The whole scene shouldn't go over a *mezzo-forte* dynamics.

During Scene 1, the live electronics performer mostly improvises with "PIANO GEN" and "CHAOS" generators as described in the table above.

Scene 2 (6-7 min)

"Then, you provide them with content and services that best fit them."

The *performance-rnn* which guides this scene was generated by a model trained against 147 pieces of J. S. Bach. Like Scene 1, it plays on average 5 notes per second, and has a little less "creativity", with temperature set to 1.3. Its pitch distribution is represented as [10, 1, 3, 10, 15, 1, 10, 10, 1, 10, 2, 10]. Unlike the other scenes, the *performance-rnn* for Scene 2 is additionally fed with with a *primer*, which is the theme of *Duetto in E minor* (BWV 802). A music sheet with the first page of this piece should be on the music desk.

Once the code starts running, the theme of BWV 802 is automatically played. The pianist just observes, getting ready to react only a few bars later. This reaction might come from an attempt to create a dialogue with the machine. A *flawed* counterpoint results from this attempt. Little by little, the gestures develop toward another musical universe. This evolution reveals itself initially through modulations, then moving toward new sonorities on the prepared piano's soundboard.

Except for the very beginning of this scene, most of MIDI data sent to the *Disklavier* is manipulated by the live electronics performer, who triggers and controls algorithms that generate pseudo-random motives. This is achieved using "PAT VOL" and "PAT SPEED" parameters. Compared to Scene 1, here the musicians are encouraged to increase the dynamics to *forte* or even *fortíssimo*, yet avoiding aggressive gestures and, like in Scene 1, ending the piece exploring resonance and continuous sounds. Also, in this scene, the live electronics performer is encouraged to make further use of "WOBBLE" and "DL WET" effects.

Scene 3 (5-6 min)

"Finally, you tune to get the fit just right."

This scene, certainly the most "aggressive" among the three, is based on *performances-rnn* generated *in extremis*, comprising an unusual pitch distribution of [0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0] at 20 notes per second on average. The level of creativity is lower than seen in previous scenes (0.7). This means that notes marked with low probability are more unlikely to be played. Also, for Scene 3 the musicians should use a higher amount of effects, as well as an extra incitement material for the performance, in the form of audible excerpts extracted from a conversation which inspired the piece.

This conversation comes from a discussion between historian Yuval Harari and Facebook CEO Mark Zuckerberg, and is part of a series of "public discussions about the future of technology in society", featured by Zuckerberg himself. The video is publicly available on Harari's Youtube channel.[1]

The scene starts with Zuckerberg's proclaiming *"Hey everyone, this year I'm doing a series of public discussions on uh... the future of the Internet and Society and some of the big issues around that..."*, followed by the generated *performance-rnn*. From its very beginning, this scene reveals a certain *anxiety*, *insistence*, and *lack of direction*. Musicians are encouraged to musically represent these intentions through *repetition*, *contrast* and *mechanical/powerful gestures*. Audio excerpts from the same conversation are played all over the piece, motivating immediate reactions from the musicians.

The only direction this scene takes is toward a dramatic *crescendo* in the end, when it goes back down to a non-dramatic repetitive gesture, which should somewhat represent that we ultimately have been tuned to *fit just right*.

1 *Mark Zuckerberg & Yuval Noah Harari in Conversation*: https://www.youtube.com/watch?v=Boj9eD0Wug8

main.py

```python
from pyo import *
import mido
import random
import time

####### MIDI CC variables ##########

PEDAL_ON = 43
PEDAL_OFF = 44
PATTERN_ON = 37
PATTERN_OFF = 53
PATTERN_VOL = 21
PATTERN_SPEED = 5
CHAOS = 6
RNN_ON = 39
RNN_OFF = 55
PIANO_GEN = 7

###################################

# DEV means using internal midi ports
DEV = True

s = Server(audio='jack', duplex=0)

# Randomly set the midi file to be picked from a subset
midi_pick = str(random.randint(1, 5))

# Set the movement to be played:
mov = 1

# 1)
mov_01_midi_file = 'MIDI/01/' + random.choice(os.listdir("MIDI/01/"))

# 2)
mov_02_midi_file = 'MIDI/02/' + random.choice(os.listdir("MIDI/02/"))

# 3)
mov_02_midi_file = 'MIDI/03/' + random.choice(os.listdir("MIDI/03/"))

s.setMidiOutputDevice(99)
s.setMidiInputDevice(99)
s.boot().start()

if DEV is True:
    port = mido.open_output('Midi Through:Midi Through Port-0 14:0')
else:
    port = mido.open_output('io|2:io|2 MIDI 1 28:0')

```

```
conversation = "conversation.flac"
sf = SfPlayer(conversation, speed=1, loop=True, mul=0).stop()

midi_play = False
pattern_play = False

def midi_scanner(ctlnum, midichnl):
    global midi_play
    global pattern_play
    if ctlnum == RNN_ON:
        midi_play = True
    elif ctlnum == RNN_OFF:
        midi_play = False
    elif ctlnum == PATTERN_ON:
        pattern_play = True
    elif ctlnum == PATTERN_OFF:
        pattern_play = False
    elif ctlnum == PEDAL_ON:
        s.ctlout(64, 127)
    elif ctlnum == PEDAL_OFF:
        s.ctlout(64, 0)
    elif ctlnum == 42:
        sf.setMul(0)
    elif ctlnum == 41:
        sf.out()
        sf.setMul(0.02)


# Listen to controller input.
midi_in = CtlScan2(midi_scanner, toprint=True)

counter = 0
counter_tick = 0

velocity = Randi(min=50, max=100, freq=1)

ctl_skip_notes = Midictl(ctlnumber=[PIANO_GEN], minscale=0, maxscale=10, init=0)
ctl_chaos = Midictl(ctlnumber=[CHAOS], minscale=0, maxscale=10, init=0)

######################### PATTERN ###################################

ctl_pattern_time = Midictl(ctlnumber=[PATTERN_SPEED], minscale=0.05, maxscale=1, init
    =0.125)
ctl_pattern_velocity = Midictl(ctlnumber=[PATTERN_VOL], minscale=0, maxscale=70, init=21)

l = Phasor(.4)
pitch = XnoiseMidi(12, freq=8, x1=1, x2=l, scale=0, mrange=(21, 108))

# Global variable to count the down and up beats.
count = 0

```

```
def midi_event():
    if pattern_play == True:
        global count
        pit = int(pitch.get())
        if count == 0:
            vel = random.randint(int(ctl_pattern_velocity.get()), int(ctl_pattern_velocity.get()) + 30)
            dur = 500
        else:
            vel = random.randint(int(ctl_pattern_velocity.get()), int(ctl_pattern_velocity.get()) + 10)
            dur = 125
        count = (count + 1) % random.randint(2,5)
        pat.setTime(float(ctl_pattern_time.get()))
        s.makenote(pitch=pit, velocity=vel, duration=dur)

pat = Pattern(midi_event, time=0.125).play()

###################### MAIN LOOP #####################################

while True:
    if mov == 1:
        mid = mido.MidiFile(mov_01_midi_file)
        for message in mid.play():
            if (counter_tick % 100) == 0:
                tick = mido.Message('note_on', channel=0, note=random.choice([10, 99, 101]), velocity=80, time=6.2)
                tick_off = mido.Message('note_off', channel=0, note=99)
                counter_tick = 0

            if not message.is_meta:
                # PAUSE
                if midi_play is True:
                    # SKIP NOTES
                    if random.randint(0, int(ctl_skip_notes.get())) == 0:
                        print('CONTROLLER IN %d' % int(ctl_skip_notes.get()))
                        if random.randint(0, int(ctl_chaos.get())) == 0:
                            print('CONTROLLER IN %d' % int(ctl_skip_notes.get()))
                            port.send(message)
                        else:
                            # CHAOS
                            s.makenote(pitch=random.randint(48,82), velocity=random.randint(20,50), duration=random.randint(1,1000))
                else:
                    while midi_play is False:
                        print('wating for resume...')
                        time.sleep(1)
            counter = counter + 1
            counter_tick = counter_tick + 1
            print(counter_tick)

```

```
elif mov == 2:
    mid = mido.MidiFile(mov_03_midi_file)
    for message in mid.play():
        if counter == 500:
            delay = random.choice([10,12,15])
            time.sleep(delay)
        if (counter_tick % 100) == 0:
            tick = mido.Message('note_on', channel=0, note=random.choice([10, 99,
101]), velocity=80, time=6.2)
            tick_off = mido.Message('note_off', channel=0, note=99)
            port.send(tick)
            port.send(tick_off)
            counter_tick = 0

        if not message.is_meta:
            if message.type == 'note_on':
                message.velocity = max(0, message.velocity + int(velocity.get()))
            if counter > 50: # modulate
                if message.note == 69:
                    message.note = 70
            if counter > 100: # modulate
                if message.note == 74:
                    message.note = 75
            if counter > 150: # do not play
                if message.note == 76:
                    continue
            port.send(message)
            # print(message)
        counter = counter + 1
        counter_tick = counter_tick + 1
        print(counter_tick)

elif mov == 3:
    mid = mido.MidiFile(mov_03_midi_file)
    for message in mid.play():
        if midi_play is True:
            if message.type == 'note_on' or message.type == 'note_off':
                port.send(message)
        else:
            while midi_play is False:
                print('wating for resume...')
                time.sleep(1)
```