Notes on Developing Personal Laptop Improvisation Software

Nathan Ho

July 26, 2019

Abstract

This paper describes the author's philosophy on laptop improvisation as developed in the a personal software instrument, titled *FORAY*, which runs on a laptop with no additional gear and makes heavy use of sound synthesis and algorithmic composition. *FORAY* is constantly evolving and personal to the author, so instead of describing the architecture of the system, this paper serves as a personal essay on the process of developing and using the instrument. It is hoped that aspects of *FORAY* might resonate with other artists working on similar projects.

1 Introduction

The motivations and approaches to live creation of electronic music are highly diverse and personal. Many musicians wish to produce electronic music in real-time and escape the non-improvisatory workflows offered by traditional DAWs, and luckily the endless modularity and hackability of modern music technology enables them to devise their own live musical workstation tailored to their needs. When the system they develop reaches maturity, feedback loops happen as the artist develops their rig: their musical style motivates them to extend their workstation, they experiment with it and discover surprising outcomes that shape their stylistic practice, and these results in turn encourage them to develop the system further. The artist entering a dialog with themselves as an instrument builder and as a musician allows them to explore sonic and expressive avenues unconstrained by the availability of existing tools.

Years ago, as a programmer who had just started learning about electroacoustic music, I had a vision to create a live electronics improvisation rig of my own, and began work on it with fantasies that I would use and develop it for years. I quickly discovered that this was harder than I had imagined, and I grew discouraged about the quality of its output and abandoned it months later. This grew into a cycle — restart project with delusions that "this time around I'll finally make *the* instrument I've been dreaming of!", realize that I have been working on it for weeks with no good musical output, trash it, and start over. It took several failures over the years, but the most recent incarnation, titled *FORAY*, proved to be the most successful. In the process, I learned many important lessons about live electronics and refined my philosophy over time. This article serves as an informal reflection to share my approach to live electronic music as developed in the project. It contains advice that wished I could read before I started building a custom live electronic music rig.

Contrary to expectations set up by most academic writing in the field of live electronics, the specifics of the *FORAY* project are not an important part of this paper. In fact, I will deliberately sidestep elaboration on many of the technical details. Much of it is constantly in flux, and I consider it more important to communicate the approach and philosophy rather than the specific ways they manifest in the project. Other details I have made a point of keeping secret. *FORAY* is an artistic work rather than a research project, and I am entitled to draw such boundaries. Given these omissions, the amount of detail expressed in different parts of the paper may be strange or lopsided, and the reader is not expected to form a complete mental picture of *FORAY*. This is intentional, since I am more interested in discussing the abstract process rather than my personal system (emphasis on "personal").

Section 2 describes the design goals and philosophy of the project, and Section 3 discusses several systems that influenced it. Section 4 discusses some of the approach to algorithmic composition, and Section 5 the philosophy toward sound design, mixing, and interaction.

2 The *FORAY* Philosophy

As a likely unnecessary disclaimer, the goals of the *FORAY* project are a product of personal experiences and interests, and few readers will fully agree with every detail. Three points stand out to me as particularly core values of the project's philosophy: productivity, price and durability, and diverse output.

Productivity: Musicians (and artists in general) often lament about starting works and never finishing them. Here is a story familiar to any musician: start an interesting new piece, work on it for an hour, leave to run some errands, come back to it the next day, and abandon it since it no longer sounds good. Nothing is inherently wrong with sketching, but many musicians are understandably interested in progressing to completed works of music, and find challenges doing so. Specific to electronic music production, the prominence of the timeline view encourages endless tweaking and caving to perfectionist impulses.

Improvisation is easily one of the quickest workflows in music. The time to produce an improvised work of music is exactly the length of the piece. More importantly, improvisation is an effective remedy for perfectionism. Coming from a background of piano improvisation, I enjoyed the finality of recordings where it is virtually impossible to change a work once it has been created. The artist has to accept the music, complete with mistakes that inevitably occur in live music. *FORAY* is an improvisatory system rooted in the desire for productivity that comes from escaping timeline views and embracing imperfections and flaws.

Productivity also encompasses more practical issues of accessing the workspace — thus, *FORAY* has been designed to be easy to transport, set up, and tear down, and easily allow making music on the go. For this reason, it is a purely software instrument that requires only a basic laptop, as will be discussed later.

Price and durability: Keeping *FORAY* as cheap as possible to build and maintain is an important goal. Furthermore, when I sit down to make music, I should not be worrying about whether my software or hardware will break. *FORAY* is therefore a laptop-only rig with no outboard gear. Laptops are not a perfect solution to anxiety about technical failures, but most of the alternatives for electronic music production present few relative advantages.

Diverse output: To justify the amount of work involved in building and maintaining the system itself, the system must be useful for creating multiple clearly distinct musical works. If *FORAY* only has the flexibility to make one piece, the time spent building *FORAY* itself would not be justifiable. One way diverse output can be achieved is through the design of an interface that surprises the user and may take unexpected (but controlled) turns. This is a motivating factor for an overall heavy use of randomness in the sequencing and synthesis components of *FORAY*.

With these principles in mind, the design decisions made in *FORAY* are discussed in the next sections.

2.1 Hardware and Software

FORAY takes a minimalist approach to both hardware and software. The instrument requires nothing more than a basic laptop, and live control is exerted using only the laptop trackpad and keyboard. The on-screen interface consists of a simple array of knobs and buttons to control parameters and trigger musical events. No MIDI controllers, outboard gear, or fragile homemade interfaces are used. On the software end, *FORAY* is built entirely in the free and open source SuperCollider[1] programming language. There are no other programs involved so that the number of moving parts can be minimized (inspired by personally witnessing performances disrupted by software programs refusing to connect).

Since I do not normally have access to a multichannel speaker setup, *FORAY* is strictly for the creation of stereo music.

2.2 Pure Synthesis

FORAY uses only real-time audio synthesis, with no use of sampling. Sampling vs. synthesis is a classic dichotomy in electronic music production, and the decision to choose synthesis is largely a personal preference. Pure synthesis also aligns slightly better with the project goals over sampling, since synthesis is more likely to yield surprising and diverse output, and the process of selecting and auditioning samples is not ideal in an improvisational setting.

Live audio input is avoided due to material disadvantages that contradict the goals of the project. It requires the setup of a microphone, which immediately opens up a number of potential issues: the need to use an audio interface, the need for an environment with low background noise, and the danger of feedback when dealing with a speaker system. All these are points where the system can fail.

2.3 Real-Time Composition

FORAY sequences notes and sound effects with algorithmic composition methods. No timeline views, piano rolls, or step sequencers are used. Naturally, this is informed by personal interests, but I have experimented

[1]https://supercollider.github.io/

with more traditional note entry methods and found that they encourage too much tweaking, and tend to be too slow for real-time music creation.

In place of repetition and traditional note entry, *FORAY* employs methods for real-time algorithmic composition, parametrically controlled in real time. In this way, creating music in *FORAY* is similar to conducting an improvising ensemble. Delegating compositional processes to algorithms allows the user to focus on music at the macro level.

2.4 Non-goals

Research: *FORAY* is not a research project. It has no novelty in signal processing, interface design, or algorithmic composition. It is not a project built for other people, and there are no plans to make it publicly available. *FORAY* is a system built solely for my own musical practice, so it is constantly undergoing evolution.

Performance: I am careful to avoid the term "performance" when describing *FORAY*, because a "performance" recalls the history of acoustic music traditions which have aspects of visual engagement. A tenet of my system — one which I hope is not too delusional — is that good-sounding music is sufficient for an enjoyable listener experience, and that the visuals of a stage performance need not be a priority.

Live Coding: Despite the fact that I use similar hardware and software tools, *FORAY* is not a live coding project. There is no live design or modification of algorithms, only simple scalar controls like knobs, buttons, and sliders that control parameters on pre-made algorithms.

AI Hype: In recent years, many artists have made use of computer-generated music as a promotional aspect of their artistry. With no intentions to criticize such works, I am drawn to algorithmic composition for its convenience rather than commentary on the implications of computational creativity.

Stylistic Neutrality: While the ability to produce contrasting musical works is a project goal, *FORAY* is opinionated on the style of music it generates — Western art electronic music with influences of ambient music. As will be discussed in Section 5, I have found such focus to be critical for a successful live electronics rig. Overfocusing on stylistic generality usually does little more than delay the production of music.

3 Influences

3.1 Yotaro Shuto's *2020*

The primary inspiration for *FORAY* is Yotaro Shuto's *2020*,[2] a live performance software instrument developed for macOS, originally released in 2015 via a Kickstarter campaign and currently in a beta stage. Immediately impressive about this interface is its maximalism, featuring hundreds of knobs, buttons, and toggles — it is intended as a complete music-making solution where every parameter is in reach. 2020's documentation contains a particularly striking passage on live electronic performance [*sic*]:

> 2020 is like a musical instrument. to become a master of it, you'll need time, inspiration, and endurance.
>
> **5 minutes :** Install it in your computer.
>
> **1 hour :** You will slightly understand how is it.
>
> **1 day :** You will be able to run through all functions.
>
> **1 week :** You will be able to compose FIRST SONG.
>
> **1 month :** You will be able to acquire Tips & Tricks.
>
> **6 month :** You will be ready to perform on a stage!
>
> **1 year :** 2020 will be a part of your body.

FORAY began as my own personal take on *2020*. Like *FORAY*, *2020* contains features for randomized sequencing and sound design. Unlike *FORAY*, *2020* is heavily sample-based (although it contains some useful synthesis features), employs a steady tempo, and contains "semi-modular" features.

3.2 Hardware Modular Synthesizers

Hardware modular synthesizers are inherently live environments, intended for the real-time creation of music without timeline views. Members of the modular community often mention the emphemeral nature of creating music entirely on such systems. A patch cannot be perfectly reconstructed after taking it down, so the user has to be content with recorded audio output alone. This notion of finality is a spirit that *FORAY* intends to capture.

[2]Shuto, Yotaro. 2015. "2020: Getting Started." http://2020.dubrussell.com/beta/getting_started.html

The general Eurorack market has increasingly explored two particular directions relevant to the current project: generative sequencing and experimental sound design. Notable generative sequencing products include 2hp's Euclid (a Euclidean rhythm sequencer), Music Thing Modular's Turing Machine (a partially randomized sequencer), and Mutable Instruments' Grids (a drum sequencer built on a zero-order Markov chain table).

FORAY itself has no modular aspects, and the actual interface is more akin to controlling an existing modular patch. Eventual addition of modular features to *FORAY* is not out of the question, but I would discourage other rig developers from concentrating on generality in the early stages of development (see Section 5).

3.3 Thomas Fay's *Algorithmically Assisted Improvised Music*

Thomas Fay's *Algorithmically Assisted Improvised Music*[3] is a collection of software modules for algorithmic manipulation of pre-composed materials in a live setting. In the state as presented in Fay's thesis, *AAIM* is a collection of Max patches and Python scripts, including a rhythm generator and modules for manipulating rhythm triggers, sample playback, and melodies.

AAIM has close philosophical commonalities with *FORAY* (and, unsurprisingly in this small space, some shared influences). In describing the artistic goals of *AAIM*, one particular passage applies well to *FORAY*:

> Ultimately, the goal of the *AAIM* system is to enable the user to act as a conductor or band leader, that is, to control the overarching nature of the performance while leaving the performance of individual parts up to the players, in this case the algorithms. This approach is common to much of electronic music, and is evident, for example, in the use of sequencers to control modular synthesisers. However, in contrast to the traditional use of sequencers, *AAIM* focusses on using this approach to allow users to determine the type, and magnitude, of variations and manipulations applied to the musical material, in a sense enabling the user to tell "the orchestra how to improvise."

FORAY shares the aim of an interface analogous to conducting an ensemble of improvisers, using macro-control over sequencing rather than

[3]Fay, Thomas. 2016. *AAIM: Algorithmically Assisted Improvised Music*. PhD thesis. https://prism.ucalgary.ca/handle/11023/3073

individual note entry. While *AAIM* is beat-based and uses some looping elements, it shares with *FORAY* an interest in experimenting with non-repetitive material in contrast to many sequencers in commercial music technology products.

There are, however, some considerable differences in scope. *AAIM* is not a self-contained application, but rather a set of algorithmic sequencing tools that make no assumptions about actual sound sources. This follows from *AAIM*'s artistic goals to be stylistically neutral. *FORAY* is a much more stylistically opinionated system, and the current development focus is more on expressiveness and creative flow rather than generality.

4 Algorithmic Composition

The *FORAY* algorithmic composition system is built with the aim of creating traditional tonal harmony. The general strategy to algorithmic composition in the program is entirely rule-based, in contrast to popular corpus-based approaches employing Markov chains or sophisticated machine learning methods. Compositional algorithms based on simple, handmade rules give the artist fine control, and importantly do not require the collection of a dataset.

The specific rule-based composition methods in *FORAY* are rudimentary. It turns out that surprisingly simple algorithms are adequate for producing musically coherent results. Again I will stress that *FORAY* is an artistic project rather than a research project, so I have not necessarily explored the implications of the system beyond reaching a point of satisfactory musical output.

4.1 Harmony Model

The traditional definition of a "chord," as a collection of three or more pitch classes, is not an adequate foundation for algorithmic tonal harmony. Jazz musicians know that a chord always comes attached to a scale, and figuring out that exact scale is a matter of context. For example, in the chord progression Dm-G-C, one cannot take the G chord out of context and play a full G major scale. An experienced improviser would recognize this as a ii-V-I progression, and likely default to a G Mixolydian scale over the G chord instead.

To give the melody generator contextual awareness, we generalize the

Name	Scale	Chord (root first)
i	$\{0, 2, 3, 5, 7, 8, 10\}$	$\{\mathbf{0}, 3, 7\}$
$\mathrm{ii}^{ø7}$	$\{0, 2, 5, 7, 8\}$	$\{\mathbf{2}, 0, 5, 8\}$
III	$\{0, 2, 3, 5, 7, 8, 10\}$	$\{\mathbf{3}, 7, 10\}$
iv	$\{0, 2, 3, 5, 7, 8, 10\}$	$\{\mathbf{5}, 0, 8\}$
v	$\{0, 2, 3, 5, 7, 8, 10\}$	$\{\mathbf{7}, 2, 10\}$
V	$\{0, 2, 3, 5, 7, 8, 11\}$	$\{\mathbf{7}, 2, 11\}$
VI^{7}	$\{0, 2, 3, 5, 7, 8, 10\}$	$\{\mathbf{8}, 0, 3, 7\}$
VII	$\{0, 2, 3, 5, 7, 8, 10\}$	$\{\mathbf{10}, 2, 5\}$
N^{6}	$\{0, 1, 3, 5, 7, 8, 10\}$	$\{\mathbf{5}, 1, 8\}$

Table 1: Examples of harmonies for a C minor tonic entered into the *FORAY* library.

chord into a larger object called, for lack of a better term, a "harmony." A harmony is a chord augmented with information on a scale to which it belongs. More specifically, a harmony is modeled with three components: a nonempty set of pitch classes designated as the *scale*, another nonempty set of *chord tones* which is a subset of the scale, and a *root*, a member of the set of chord tones. This formalization of harmony, while certainly not comprehensive in the grander scheme of tonal harmony, allows the melody generator to have full awareness of both chord tones and non-chord tones.

Using this system, we can build up a library of harmonies. Table 1 displays an example of some harmonies entered specifically for a C minor tonic in the *FORAY* system. Other keys are formed in the obvious way by transposition. A few details of the table are worth noting. Most chords occur within a natural minor scale, but some chords are not — e.g. V, which includes scale tone 11 (B-natural) rather than 10 (B-flat). The scale for the chord $\mathrm{ii}^{ø7}$ specifically omits pitch classes 3 and 10, or the 3rd and 7th scale degrees of the tonic. This is not a reflection of any properties of the established theory, but simply the removal of tones which sounded awkward, an example of personally doctoring chords and scales to "teach" the system my own ideas of how tonal harmony should sound.

Currently, *FORAY* employs only tonal harmony, but it is designed with separation of concerns between harmony objects and the melody generator, making the system simple to generalize to non-tonal paradigms of harmony. Generalization to non-12ET pitch spaces is also straightforward.

4.2 Chord Progression Generation

FORAY's current implementation of chord progression generation is only preliminary, but musically serviceable. It uses a small bank of manually written short chord progressions ranging from three to six chords, all of them ending on the tonic, and simply selects random progressions in sequence.

This scheme is inspired by Luce Beaudet's analytical model of tonal harmony expanding on the work of Goldman and others. This model describes tonal harmony in terms of *harmonic structural units* (HSUs), which are fragments of a descending diatonic circle of fifths (IV $\rightarrow$ VII $\rightarrow$ III $\rightarrow$ VI $\rightarrow$ II $\rightarrow$ V $\rightarrow$ I) that end on I, V, or occasionally IV. That an HSU must end on a chord close to the tonic is a way of "grounding" progressions and making the listener aware of the tonic. Certain progressions that do not exactly follow the circle of fifths, such as IV-V-I, are explained using a vocabulary of transformations, such as substitution of a chord for a related chord or insertion of a closely related chord. A complete explanation of the model is beyond the scope of this paper, but the reader is invited to visit Beaudet and Menard's entertainingly illustrated online textbook on the subject.[4]

Even a simple sequence of partially randomized chord progressions produce convincing results, generating tension and release that the user can react to while creating music in the system. More advanced uses of Beaudet's analytical model, especially in how it relates to live user control, may prove to be an interesting avenue of exploration for generative systems of tonal harmony.

4.3 Melody Generation

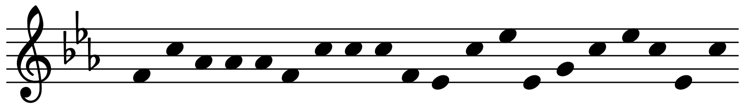
The melody generation techniques used in *FORAY* are, as always, simple and only developed to the point of producing satisfactory output. The methods presented are loosely inspired by the work of Brown and et al.[5] on melody generation based on Gestalt psychology.

The strategy for developing this component is "start boring but pleasant, then make it less boring." The simplest "pleasant" melody generator would be a random arpeggiator that selects independent random chord tones within the currently active harmony, with sample output in Figure 1a. Immediate

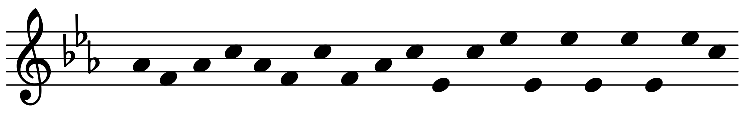
[4]Beaudet, Luce and Ménard, Sylvie-Ann. *L'œil qui entend, l'oreille qui voit — un modèle d'analyse du discours harmonique tonal*. http://bw.musique.umontreal.ca/

[5]Brown, Andrew R. et al. 2015. "Techniques for Generative Melodies Inspired by Music Cognition." Computer Music Journal, Vol. 39, No. 1.

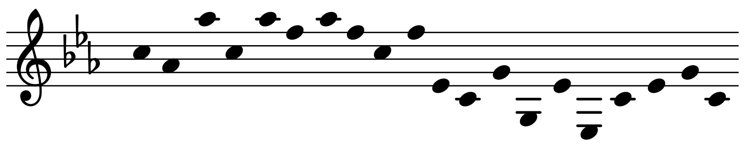
a) Random arpeggiator:



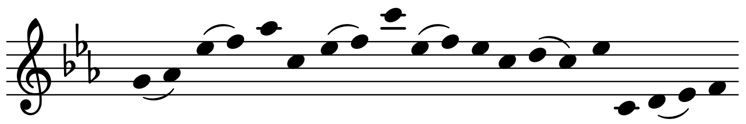
b) Avoiding immediate repetitions:



c) Modulation of range with a sine wave:



d) Neighbor tones:



e) Additional three-note gesture:

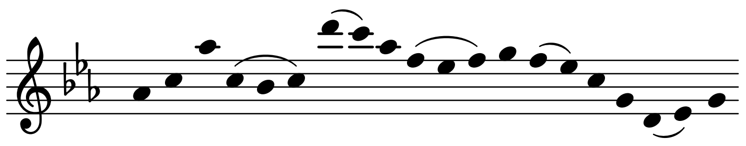


Figure 1: Sample outputs at different stages of the melody generator with underlying chord progression of a minor plagal cadence, iv-i, in the key of C minor.

repetitions sound awkward, so they are disallowed (Figure 1b). To add more variety and directionality over time, we introduce gradual modulation of register by slowly changing the center of the range. This modulation may be controlled directly by the user with a knob, and/or autonomously varied using any kind of slow random LFO. In the latter case, a triangle or sine wave with randomly varied frequency produces a nice balanced usage of high and low register (Figure 1c). Random walks must be used with caution since they may cause the range to wander off too high or too low, and rarely in a musical way.

A melody generator that adheres to only chord tones is already quite useful as a textural element, but its output is limited. For features more typical of a real melody, we may introduce non-chord tones. In place of a random chord tone, the generator has a chance of instead composing a small two-note gesture, consisting of a chord tone preceded by a neighboring tone in the scale (Figure 1d). Expanding on this concept, the melody generator can be augmented with a more elaborate library of non-chord-tone gestures (Figure 1e). I built up this library (not shown) by improvising melodies at the piano and analyzing what gestures seem to come naturally.

The melody generator currently in use in *FORAY* is somewhat more complicated than this (in particular I have not discussed its handling of rhythm), but this should give the basic idea of its operation. As one suggestion for improvement, the numerous skips and leaps in the melody can be smoothed out by using an appropriate set of constraints. Many other developments in algorithmic composition literature may be incorporated. While the output of the current melody generator is not always as memorable or as interesting as a human improviser, even at their most bland the lines generated are still serviceable and musical. It makes for good polyphonic textures when multiple voices run at once, even when no polyphonic constraints are specifically designed — the only unifying factor is that all melodies derive generate in parallel from the same chord progression.

When a generative melody system developed in conjunction with synthesizer patches, as is the case with *FORAY*, there is interesting potential for the melody generator to attach special semantic information on how the notes were generated. For example, notes coming from a two-note neighbor tone gesture are tagged as such, and the synthesis patch might decide to interpret such moments as portamento. This is analogous to a human perfomer reading music notation and using contextual and music-theoretic cues to intelligently inform their use of phrasing, dynamics, and articulation.

Close coupling between synthesis and sequencing is particularly easy in the SuperCollider language, which permits users to encode musical events with arbitrary key-value data, and in particular does not force the user to shoehorn auxiliary data into MIDI CC numbers. With effort on the designer's part, giving the synthesizer access to deeper semantic information than simply notes and rhythms can bring dull patches to life.

5 Sound Design and Interaction

5.1 One Big Patch Syndrome

One Big Patch Syndrome is my term for a mistake that was responsible for many failures in previous iterations of *FORAY*. The demographic particularly affected by this issue is software engineers interested in creating their own live electronics rig, but it may apply to many others.

One Big Patch Syndrome happens when an artist overthinks the generality of their system and starts developing technical scaffolding instead of making music. "If I just create the one big patch that allows me to make any sound I want, I'll finally achieve musical expression in my electronics!" This attitude just leads to frustration, and usually bad music. For example, in one earlier attempt at an electronics rig, I started by attempting to classify as many timbres as I could think of, organizing them into a set of categories, and mapping different parts of a MIDI keyboard to different categories. I will leave it to the reader to imagine how musically interesting the result was.

To avoid One Big Patch Syndrome, stop thinking about whether the expression of your system is optimal and whether it achieves full generality in timbre. Instead, prioritze the creation of musical works. Design some sounds, assemble them into a composition, and add more and more live-controlled parameters. When you have great musical materials, live control will immediately follow. Worrying about the diversity or stylistic generality of your patch in the early stages of the project will paralyze its development.

The first steps toward the creation of *FORAY* was a set of three different patches that I realized had some properties in common, and could be merged into a single synthesizer with a flexible set of parameters. I then made a GUI interface with knobs to dial in parameters. Over time, I added more parameters that seemed interesting, removed parameters that felt less musically useful, and eventually added a preset system for saving and loading settings. I used a basic algorithmic sequencer that simply picked random chord tones, and slowly made it smarter and more musically sophisticated. By starting with sound design and incrementally building the system around it, the results proved far more motivating and fruitful than my previous attempts at overgeneralization.

As specific advice for experienced software engineers building their own workstation, and particularly other SuperCollider users: code quality should be extremely low on the priority list in this pursuit. When developing custom music software, it is important to work quickly and sloppily, focusing on creating sound rather than making well-architected software. There are many technologically oriented artists who possess only rudimentary coding skills, but are still able to produce successful works because their focus is the quality of the results.

5.2 Mixing and Dynamics

Dynamics are a particular challenge for musicians working with real-time generative and algorithmic music. Heavy use of algorithmically generated note sequences and random modulation makes the volume and frequency balance between different tracks in the mix inherently unpredictable.

The first and foremost issue is avoidance of clipping, and the obvious solution is a limiter. After adding this in the early stages of the project, I quickly realized that the limiter could be driven hard as a creative effect. All elements are being mixed in real-time *through* the limiter, rather than the limiter being applied as a downstream stage after mixing. Overdriven limiters often create sweet spots where two elements interact with each other and fight for headroom. *FORAY*'s architecture also places the limiter after reverberation, which has an interesting effect on the perception of space.

Seasoned audio professionals frown on aggressive master compression, usually preferring to carefully fine-tune balance at the stem level rather than allow an algorithm to set levels for them. While good advice for more traditional audio production settings, this is absolutely not something I am concerned with as an experimental electronic musician. Imperfect mixing is something to be expected in a live rig with unpredictable generative elements, and I am more than happy to allow an algorithm to exert control over dynamics so I can prioritize sound design and arrangement. Furthermore, the musical style I have developed through my use of *FORAY* benefits from heavy compression. Other forms of electronic music, however, may tends more toward quietness and high dynamic range, and may prefer the use of a limiter only as an anti-clipping measure.

In the master bus, I use two soft-knee limiters in series: first a "color" limiter with a fast attack and release, and a second "mastering" limiter with slow attack and release and a bit of lookahead. Both are set to nearly the same threshold. The color limiter is intended to be driven hard to create the aforementioned dynamic interactions, and the mastering limiter tames transients so the output track is slightly brickwalled. There is an additional 10 dB of headroom just to be absolutely safe from digital overs. I found no need to adjust these limiter settings while playing, so no controls were created — just a visualization of the gain reduction amount of the color limiter.

Exploring the use of limiters in the creation of musical works, I arrived at a playing style I am satisfied with. During quieter sections of works, I tend to stay 5-10 dB below the limiter threshold. During louder sections I aim for moderate gain reduction of 3-5 dB, and in climactic moments I drive the limiter to 10 dB or more. This way, the overall work is dynamic, and the loudest sections are appropriately loud.

5.3 Sound Design

Sound design is a personal practice that can only be developed with experimentation over time, so I am most comfortable keeping secret the specific synthesis patches I created in *FORAY*. The following tips summarize my overarching philosophy on sound design.

Replication Exercises: Much of my sound design comes from "replication exercises." Typically, I try to locate interesting-sounding source material, focus on a single element, and attempt to mimic it with a synthesizer patch. Whether or not the replication is successful, it is often at least interesting, and forms a starting point for a a new instrument.

Early on, I used mostly other electronic music to explore replication challenges, but increasingly, I have deliberately avoided over-listening to electronic music in related styles to resist the temptation to mimic it too closely for myself. Instead, I have found acoustic music a valuable resource for studying sound design, since acoustic instruments often have unique sonic properties that a synthesist would not otherwise consider. Western art music with unique orchestration and numerous non-Western musical traditions are particularly fruitful sources of inspiration.

Effects Cocktail: various combinations of filter, chorus, phaser, flanger, reverb, sample rate reduction, pitch shifting, saturation, multitap delays, wavefolding, extreme compression and limiting, granulators, and feedback across any of those. I tend to use simple sound sources and let the effects sculpt them into something more complex, so this technique may be viewed as a generalization of subtractive synthesis. The "effects cocktail" was directly inspired by the production methods used in brostep and dubstep bass synthesis.[6]

With overcomplicated effects chains, every element in the chain adds its own flavor, especially if it contains artifacts of some kind. The signal path accumulates a cascade of imperfections and complexities, resulting in wonderfully organic sounds. It can be used subtly as well as dramatically — by mixing a small amount back into the dry signal, a bit of space and character is added.

Reverb is a particularly important effect in purely synthetic music. Synthetic sounds tend to be completely dry, so even a small amount of reverb can give the impression akin to the body of a real instrument. (The traditional audio mixing advice of sending all tracks at various levels to a single reverb, while useful, is less important when dealing with synthesis-heavy music.) Furthermore, reverbs create complex signals that have a butterfly effect on downstream elements in the signal path. Unusual reverbs such as banks of comb filters or modal resonators provide interesting alternatives when a standard reverb proves too bland. There is, of course, such a thing as too much reverb, and I often have to suppress the habit of overapplying it to compensate for an undercooked sound source.

I am particularly fascinated by the possibilities when applying an effect and then imperfectly reversing it. These include pitch shifting up and then down, compression followed by expansion, reverberation followed by crude dereverberation (i.e. noise gating), etc.

Modulation: Movement is critical for interesting sound synthesis. I make extensive use of random LFOs created by starting with a random impulse train, using that to trigger random sample-and-hold noise, and then smoothing out the signal with a one-pole lowpass filter. The two controls for a random LFO are its rate and its smoothness. This way, I have control over a continuum from hard glitches to smooth, organic wobbles. Modulation targets can include parameters of filters, and crossfade controls between parallel effects chains.

[6]Mr. Bill. Masterclass at BPF College. https://www.youtube.com/watch?v=XNbYjPQ9Zkg

5.4 Pacing

Once *FORAY* reached a level of maturity where I could create large-scale musical development, the pacing of musical works in a live environment became a concern. This is a lesson I learned when I presented *FORAY* live at a forum — in my nervousness, I burned through the options I had far too quickly, immediately reaching stasis. States of frustration where I cannot decide where to go next are detrimental to creative flow.

A timer added to the software's GUI helped with awareness of real time, but solving pacing issues is largely a human factor. I come from a background of piano improvisation, where I can do spontaneous composition from intuition. However, since I lack experience improvising on the system that I built, I have to relearn how to pace my works. To get satisfactory development, I found I had to think several steps ahead — how will the piece change over the next minute? What is the destination I am trying to arrive at?

Pacing is also perceived differently between the artist and audience. I have been listening and experimenting with these patches for months, but the listener has only been hearing it for 10 seconds. The use of generative sequencing and highly dynamic sound design means that the patch is still evolving quite quickly even if I am not actively changing any parameters. However, during such moments, *I* know how the patch will develop, and I reach impatience much quicker. When I realized how this distorted perspective impacted my music, I started correcting for it by simply developing slower. I lingered much longer on points that I found pleasing. I avoided immediately activating every instrument all at once, instead waiting extended periods of time to introduce them.

By working in a goal-oriented, time-aware manner rather than aimlessly wiggling knobs, and by staying patient and moving slowly, I immediately noticed improvements in the form of my music, a reduced frequency of "stuck" moments, and a much easier time improvising for an hour or more without running out of material or dragging any section out for too long.

Nevertheless, stuck states are still unavoidable. The addition of a preset system proved hugely beneficial as an easy way out of such points, provided that they are not overused. A preset system also has an interesting side effect: Extremely interesting results happen when crossfading the parameters of distant presets. Both settings were human-designed, and intermediate settings tend to have the qualities of both, but a sound completely unpredictable to the designer.

Overall, good pacing with live generative electronics is a problem of finding a balance between interesting development and patient use of material. My personal experiences are that I move too eagerly and had to compensate by slowing down. Other artists might find they have the opposite problem. Either way, developing an awareness for pacing is an important aspect of electronic improvisation, and presents specific challenges for artists building their own live workstations.

6 Conclusions

Building a custom rig for live electronic improvisation is an exciting prospect, but one that takes hard work and an understanding that failure will happen and that the system will never be perfect. In the development of *FORAY*, I experienced a lot of failures. While *FORAY* itself I consider a success, it is still a perpetual work in progress. With constant development and refinement, and more importantly extensive practice with the instrument itself, I have increasingly felt the elusive properties of "expressivity" and "creative flow."