COMPOSING WITH BLOOMS

Elliot Cooper Cole

A DISSERTATION

PRESENTED TO THE FACULTY

OF PRINCETON UNIVERSITY

IN CANDIDACY FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE BY

THE DEPARTMENT OF MUSIC

Adviser: DMITRI TYMOCZKO

January 2023

Abstract

This dissertation contends that, beyond its practical utility, computer programming offers artists a valuable mode of thought, and a path of creative discovery. As an example, I explore the insights, potentials, and contexts extending from my own path designing software in pursuit of new musical languages.

This software begins with a musical object I call a *bloom*. First I introduce the project, then define this object and explore the perceptual effects that underly my attraction to it. Exploring these effects in terms of information theory suggests broader insights about meaning in music—what it is, how we experience it, and how we can create it. Second, I present a narrative of the project and reflect on its lessons. This section includes analyses of a number of pieces I have composed with its aid. These compositions are included in the Appendix.

The third section draws historical and philosophical connections between the project and other music, people and ideas, including the parametric and chance traditions, Cage and Xenakis, nature painting, foraging, gardening, Brian Eno, and modular synthesis, in order to better understand bloom praxis and ergodynamics.

Then I return to the project as it stands today, with a description of the current version of the software and an introduction to the code examples, as an invitation for others to explore alongside me.

I close with a meditation on advice from John Cage, that “what we need is a computer that isn't labor saving but which increases the work for us to do.”

Table of Contents

Could we do it with a computer?
I don't mean make computer art but
cd. we sit in an audience
and enjoy it once it was made?
Don't think the answer's No;
it's inevitably Yes. **What we
need is a computer that
isn't labor saving but which
increases the work for us
to do**, that as McLuhan says
can puns as well as Joyce
revealing bridges where we thought
there weren't none.

John Cage *Diary: Audience 1966*, first draft (emphasis mine).

It happens that computers can be useful in certain ways.

Iannis Xenakis, *Formalized Music*, 1963.

Introduction

The purpose of this paper is to consolidate and create a record of a long-pursued project and to think through and share its lessons. I hope to illuminate not only my own work, but also, for those curious to take it up, the value of the path itself: to learn to write your own software as a site of creative discovery, and as a way to create new musical languages.

The project is a software tool I called *bloom*. Initially sketched to help me write a very specific type of music, over time it became a more general tool as I pursued new and different uses for it. I have written notated scores with it, created electronic music, used it in live-coding performances and hands-on interfaces, and still see many possibilities I haven't explored yet. Early on, I used the word *bloom* to refer to a specific type of musical gesture, which I will explore extensively in the next section. But once I had devised a way to represent and transform a bloom, I recognized that my tools that could be generalized to work with any material. In the process I deepened and clarified my ideas about music, and created a body of compositions stemming from those ideas.

This process of generalization is at the heart of why writing your own software can be a process of self-discovery, and open new pathways for creative thought. Just as we can sit down with a piece of paper with only a vague idea and, through the act of writing, draw out much more, coding also can show us that we know more than we think we know, and lead us to understand things we hadn't. The constant test of code—does it run?—pushes us to be clear and precise with our ideas, and the programming principle *abstraction* constantly prompts us to draw and clarify the line from the particular to the general. For example, when composing a single melody, it is enough to find a particular shape that pleases us; we don't need to know why it does. But to tell a computer how to compose melodies that please us, we must interrogate our preferences, abstract the essential from the specific, and define a general schema instead of a particular form. This brings us into a deeper understanding. This clarity can be particularly illuminating because so much of our musical knowledge is implicit, revealed through our intuition about what "works" and what doesn't, our likes and dislikes, responses that are often more felt than interrogated. Any endeavor to get a computer to make music that "works" for us requires us to make a hypothesis about what "working" is. We test, we refine, we learn. It can often feel like doing music the hard way, in line with Cage's suggestion that what we need is "a computer that isn't labor saving but increases the work for us to do." But I believe that any such project, no matter how quixotic, has much to teach.

One purpose of this paper is to learn from my own project, and share what it has taught me. But this document serves other purposes as well. It is important to create a record of our private projects simply to honor the years of energy that goes into them. With software projects, it is also important to combat their natural decay, as the technology around them inevitably evolves and leaves them behind. Coding projects by self-taught programmers, like me, for private use are particularly vulnerable. They are often poorly documented, inconsistently written, informally tested, and depend heavily on their creator's working knowledge of naming conventions, quirks, bugs and

workarounds that fades over time. Returning to one's own code after some years can be baffling, so a large part of this project is revising, testing and documenting it so that my past effort can continue to be useful in the future.

I am also, for the first time, making this code available to others[1]. I do not expect much adoption; it grew out of a private line of inquiry, not an ambition to create a broadly accessible tool. But I do see many possible directions available to those who want to enter my world. It can be used as a compositional aid, a live coding instrument, a generative music platform, or a deep listening practice. One purpose of this paper is to lay those out and invite others in.

Another is to help others understand some of my compositions better. I share and analyze many of my pieces to show connections between the software practice and the work that grew out of it. This is to aid people curious about my music, or just curious to see an example of how computers and humans can live in creative co-operation, and specifically how through that co-operation we can create new musical languages.

Bloom: introduction

In my private musical lexicon, a bloom is a gesture archetype. A bloom is ~4-10 random notes played quickly in sequence with short, irregular durations, in a decrescendo, followed by silence, on a piano with the pedal down.

The charm[2] of the gesture is in the tension between its unity and disunity. The disorder of the random pitches and durations is unified in the time, amplitude, and timbre domains.

[1] http://github.com/elliotcole/bloom

2 Messiaen's phrase "the charm of impossibilities" refers to a similar experience of pleasure in paradox, "at once voluptuous and contemplative." Olivier Messiaen, The Technique of My Musical Language: Text with Musical Examples (Paris: Alphonse Leduc, 2007), 12.

In the time domain, the notes are unified by proximity, short durations between the notes, and boundary, a longer silence after. Random durations are also unified by their irreducibility: without any repeated pattern or apparent subdivisions, they are more likely to be perceived as a single unit.

The amplitude domain is unified by a single shape: a decrescendo, loud to quiet.

The sustain pedal creates unity in the timbral domain, fusing each separate note into a single resonance. This resonance is important. Music for a monophonic instrument like a flute can be derived from blooms, but a flute cannot play an ideal bloom. The notes need to ring.

I find that these unities of time, amplitude and timbre are enough to make any collection of pitches sound intentional, musical, and very frequently beautiful. From this I draw a broader insight about composition: music is a multidimensional space, and those dimensions share responsibility for "making sense." If one dimension is clearly structured, other dimensions can be more free.

Repetition / Information / Meaning

More "charming impossibilities" arise when the bloom is repeated. A random collection of notes played once can appear haphazard, casual, an accident. Play the same notes and match their timing and intensities *perfectly*, and suddenly the image is revised: the figure is no longer casual but filled with a fierce intention. Despite having the identical content, the second iteration creates a radically different impression than the first, and it prompts us to retroactively re-experience the first impression in a different way.

This is a revision of meaning, specifically the type of meaning that Leonard Meyer calls "embodied meaning":

> A stimulus or process may acquire meaning because it indicates or refers to something which is like itself in kind—as when the rumble of distant thunder on a sultry day and the piling up of storm clouds… indicate the coming of a rain storm.[3]

This form of meaning is distinct from "designative meaning," as in the symbolic connection between, for example, the word "apple" and the object it designates. Although music can create designative meaning, embodied meaning dominates the musical experience. As thunder "means" that rain is likely to follow, meaning in music involves causality, expectation, and probability. It arises from the experience of the antecedent/consequent relationship. In that experience, the listener first apprehends a single object, as in the first presentation of a bloom. At this stage it has what Meyer calls "hypothetical meaning" because it prompts the listener to form a hypothesis about what is likely to happen next. This hypothesis is informed both by the content of the antecedent and the listener's past experience with the style of music. When the consequent appears, the antecedent takes on "evident meaning" in retrospect when the relationship is revealed. This is an atomic description of what is of course a multilayered and open-ended process. It is multilayered because music is multidimensional (pitch, timbre, amplitude, pattern, etc.), and the play of hypothesis/evidence operates within and across all of these dimensions and on multiple time-scales. And it is open-ended because music is a flow; as it unfolds, all features at all scales are both antecedents and consequents, and the meanings of their relationships undergo constant revision in what Meyer calls "an evolving discovery of attributes."[4]

Limiting a phrase to a pair of blooms reduces this complexity greatly, which brings the essence of the experience of meaning into focus. I think of blooms as "lab-ready" samples, ideal for experiments in perception. When presented in a pair, a bloom is a discrete event with clear boundaries, clearly either an antecedent or consequent, and

---

[3] Leonard B. Meyer, "Meaning in Music and Information Theory," *The Journal of Aesthetics and Art Criticism* 15, no. 4 (January 1957): pp. 412-424, https://doi.org/10.1111/1540_6245.jaac15.4.0412, 413.

[4] Meyer, 416.

not both. The salient parameters are reduced to three: pitch, amplitude, and duration. Last, while a true stylistic vacuum is impossible, a random bloom usually doesn't particularly suggest any familiar style. It is possible then that its internal features, and not cultural cues, are broadly responsible for the expectations that arise.

They are also highly ambiguous, and ambiguity is an essential part of creating meaning. Meyer explains that "a tonal process which moves in the expected and probable way without deviation may be said to be neutral with regard to meaning," in the same way that we experience advertisements along a highway but barely register them. They are expected, so our response is habitual. We only experience meaning in the events that deviate from expectation. This can happen in two ways. A single, clear, expectation can be created but fulfillment delayed, which creates dramatic tension (as in peek-a-boo, a game omnipresent in music). Or the antecedent can be ambiguous and suggest many possible paths. I sometimes quip that there are two kinds of music: "music where you know *what's* going to happen but you don't know *when*, and music where you know *when* something is going to happen but you don't know *what*." This isn't an accurate division of music, but it does neatly summarize these two paths to creating meaning.

In either case, the experience is uncertainty, and uncertainty is the foundation of information. Claude Shannon began the information revolution with this observation. He identified that the basic unit of information is a unit of uncertainty, or what he calls *entropy*[5]. Consider the amount of information in a sequence of one million zeroes. To communicate this sequence successfully, one million digits are not required: I could compress it to as few as six: 0!10^6, if ! means 'repeat.' Its information content is extremely small because it is highly patterned and highly predictable. We can guess what the million-and-first digit will be with a high level of certainty. But a sequence of a million random digits cannot be reduced, and would require one million digits to

[5] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal* 27, no. 4 (1948): pp. 623-656, https://doi.org/10.1002/j.1538-7305.1948.tb00917.x.

transmit. It contains much more information. Imagine receiving it as a transmission one digit at a time. The past digits give no clues as to what the next digit will be, so we feel constant uncertainty about what might be coming next. Most experiences are in the middle: past information does condition future possibilities, as in patterns of numbers, or context in the telling of a story. We can't always predict what will happen next, but we can reasonably narrow down the possibilities.

The pitches and durations of an antecedent bloom are randomly chosen, and therefore high in uncertainty and packed with information. If it were pure random information, it would not suggest any subsequent information at all, creating no expectations for the listener. It would be inert. But this is clearly not the case. A single bloom can feel full of longing—direction, expectation—as its notes suggest various resolutions. This emerges from the acoustic information that is added when the pitches are turned into sound.

To understand why, we'll borrow another idea from information theory: redundancy. Redundancy has a negative connotation in common usage, but Shannon explained why it is essential for communication. He defined redundancy as the reciprocal of entropy/uncertainty/information. Redundancy is the part of the data that has no information.

If it is not necessary, why is it there? If information is related to *uncertainty*, redundancy gives you *certainty*. Certainty is necessary because no method of transmission is lossless. There will always be noise in the wireless, packet errors in the ethernet, and misunderstandings between people. Redundancy is what allows information to survive noise. One example is the checksum, for instance on a barcode. The final digit of a bar code is computed from all the previous digits. It is possible to identify the item without this last digit, but without it, it is not possible to know if one of the other digits has been mis-read or falsified. The checksum confirms the information. It provides certainty.

Another example is the English language. Shannon estimated that 50% of the language is redundant. One of his tests was to remove a percentage of letters from a message and see if it still made sense. The redundant letters aren't strictly necessary but they are essential for increasing the certainty that we are interpreting a word correctly.

Music similarly depends on the interplay between entropy and redundancy. In a musical performance, a message is transmitted to a listener. Though the processing of that information is extremely complex, crossing multiple dimensions and time scales, even the most unsophisticated listener has an intuition about whether the communication was successful. The bewildered concertgoer who grouses, "I don't get it" after an experimental music concert is reporting an unsuccessful transmission. A transmission fails because the ratio of entropy to redundancy is too high, and so the noise—which includes one's attitude and cultural lens—renders too little of the data certain. This provokes a visceral experience of doubt and confusion. Alternately, music with too much redundancy strikes us as boring and empty. What is it empty of? Information, uncertainty, surprise.

A possible link connecting information content with the subjective experience of musical pleasure is suggested by neurologist Robert Sapolsky in his studies on dopamine.[6] In experiments in which monkeys are given tasks which sometimes produce rewards, he demonstrates that their dopamine level, as well as their engagement with the task, is maximized when the rewards are 50% predictable. In other words, when they are able to correctly anticipate 50% of future outcomes, they are maximally engaged, and enjoying maximum dopamine. In terms of information, in this optimal state they are receiving a signal that is half information/unpredictability and half redundancy/ certainty. (This raises an intriguing parallel with Shannon's approximation that the English language is 50% redundant.) So it may be with humans and music.

---

[6] Robert Sapolsky, "Dopamine Jackpot! Sapolsky on the Science of Pleasure." www.youtube.com/watch?v=axrywDP9Ii0.

What is redundancy in music? Repetition of every kind. Repetition in the time dimension is the most obvious. This includes literal repetition of material as well as repetition of abstractions: pulse, pitch contours, rhythmic patterns, rhetorical devices. Redundancy is also an essential feature of pitch space. Octave equivalence folds an infinite gamut into a small redundant cycle. Limiting music to a scale increases redundancy by helping listeners make somewhat more certain predictions.

Redundancy is also the foundation of *ensemble*, in the general sense of "playing-togetherness." Multiple players reinforcing the same meter and articulation can project a clear affect; their redundancy clarifies the signal. Orchestration is the work of planning these redundancies. Harmonic clarity is created with pitch doubling. Formal clarity is projected when all instruments observe the same cadence points, section boundaries, and phase changes.

The relationship between a piece of music and its stylistic context is also a form of repetition. To gain familiarity with a style is to hear many examples, and register that some features are repeated again and again. Those conventions are the redundancy that reduces uncertainty while listening, while also sensitizing us to what is unique. This is both a technical description of the information and a gut-level experience everyone has listening to music. Uncertainty is a sensation, and in qualitative assessments of music—"do you like it? / do you 'get' it?"—it is frequently a determinative one.

Consonance and dissonance in harmony can also be measured in terms of redundancy. A triad sounds the way it does because the notes within it overlap and reinforce each other's strongest overtones. They reinforce each other the way a checksum reinforces a bar code: they confirm the data. An error—a wrong note or sour tuning—is detectable because of the thick network of checksums woven in. This is where we can begin to answer the question that launched this detour into information theory. How can a random series of pitches create any expectations at all?

The answer is that random notes of a bloom create a harmony. Every pair of notes has its own relationship. The quality of that relationship can be described by the degree of similarity (redundancy) and difference (information) between their overtones. From this observation a line can be drawn all the way to an emotional response. These relationships create tendencies towards resolution in the mind of the listener, who moves from a state of stress when uncertainty (information) is high to a state of ease when certainty (redundancy) is high. Dissonant intervals share fewer overtones, so they contain more information. More information is more uncertainty. More uncertainty is more stress. While some listeners seek out more dissonant, less certain experiences, it is not because their response is different, but because stress can also be exciting.

A bloom of 5 notes contains 120 pairs of these relationships. In that tangle of tendencies there are many possible paths towards resolution, many “hypothetical meanings." Each path is an expectation waiting to be given an “evident meaning” when the consequent is revealed. A mental model containing 120 different hypotheses for a 5 note chord is probably imagining too much. Instead, picture a single hypothesis that gets progressively modified as each new note is played. Some notes confirm a hypothesis, like “we are in D major”—and by the luck of the draw, several may. Then outliers create ambiguity. I present this image not as imaginary speculation about the specific way our mind makes models. But, as successful receivers of information, we know that the basic operations of information theory are in play: we employ a sequential heuristic that processes a signal by using the past to predict the future.

Blooms have been my laboratory for thinking about these things because they offer a simple, crystallized version of the experience of meaning in music. Returning to the case that began this section, a random bloom repeated *exactly*, the experience can be understood in a new way. Without patterns of pitch or rhythm, the first bloom is pure information before it is played. Playing it reveals patterns of resonance, harmonic redundancy. As redundancy increases, our ability to make a meaningful prediction also

increases. From this prediction arises expectation, sensation, and affect. That expectation is for resolution, increased redundancy, and so what happens next is a shock. Every detail of the original bloom is made redundant when it is repeated. The entire experience flips from having “no pattern” to being “all pattern." The pitches flip from sounding like “all random notes” to “no random notes.” The rhythms flip from casual and mushy to laser-etched and diamond-bright. None of our harmonic expectations are resolved, but certainty increases anyway—each note resonating perfectly with its double, in our memory.

Now our expectations are tightly focused on a single hypothesis, which seems very likely: the bloom will be repeated again. We are alert to all of its details, maximally sensitized to any variation. If it is repeated exactly, our theory is confirmed, uncertainty drops, stress drops, the mind relaxes, and we think, “I get this music." It is a satisfaction at the beginning of boredom. We may be entering the land of a thousand zeros. But if it is repeated with even the smallest variation, all of our attention goes to that new information. Our theory must be revised, our uncertainty increases, and we stay engaged with the music.

This is a parable about all music, which beneath its endless surface variety is united by “the psychology of human mental processes— the ways in which the mind… selects and organizes the stimuli that are presented to it.”[7] This story of three blooms is a microcosm of these processes: the cycle of information and confirmation, hypothesis and evidence, certainty and uncertainty.

I have been exploring blooms for 14 years, using software to model, transform, and compose with them. Many pieces of music have resulted but, to my taste, it is hard to top, for sheer beauty, the scene I have just described: a bloom repeated once exactly, and then again with small changes.

---

[7] Meyer, 413.

Project Narrative (2007-2021)

Seed

In 2007 I wrote a piece for piano and computer called *Parable of the Sower*. In it, I played the first 24 notes of a Bach D Minor Violin Allemande very slowly on a piano over two and a half minutes. This was recorded and automatically looped while I recorded another layer: one new note for each note of Bach. Over seven cycles, a cluster of notes slowly grows around each original seed. I composed as carefully as I could, by asking this question of each cluster: “what is its energy and character? What one new pitch can I add, and how should I play it, to reinforce that energy and character?" Iterating seven times, I composed a sequence of chords.

Performing these chords one note at a time, extremely slowly, with no click track made simultaneous events impossible. Unable to anticipate when a note would occur on the previously recorded layer, I could only react. Each chord sprawled out in a loose, imperfect strum that reminded me of a flower blooming. Five aspects of this piece permeate my bloom music in the early phase, until *Bloom Suite* (2013):

- A texture: bloom-shaped gestures, played on a piano, with lots of silence to ring in.
- A grammar: iterative, with small variations that take on a large significance.
- A rhythmic mode: smooth, non-metric time
- A mental texture: contemplative, subtle, patient, hyper-sensitized to detail.
- A metaphor of plants, flowers, organic growth; an image of the composer as a gardener, cultivating music to fulfill its own will to grow.

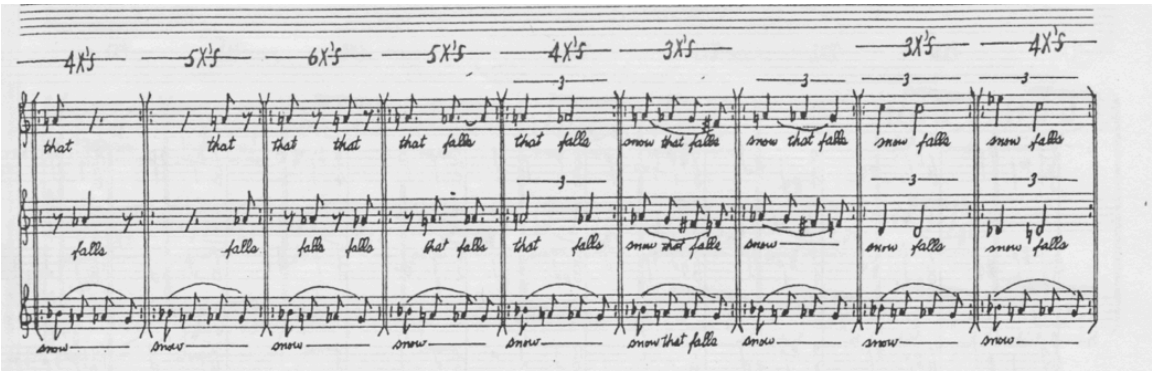
As Salty as Possible

This aesthetics of this music begin with the observation that the *salience* of information is inversely related to *quantity* of information. Salient details are the ones

that leap out at you. While the words aren't etymologically related, I think about this as "saltiness." In music with lots of elements—as an extreme example, consider Stockhausen's *Gruppen*—the salience of any detail on the note level is diluted by the flood of information. But in music with only a few elements, their individual details are more significant, more "salty."

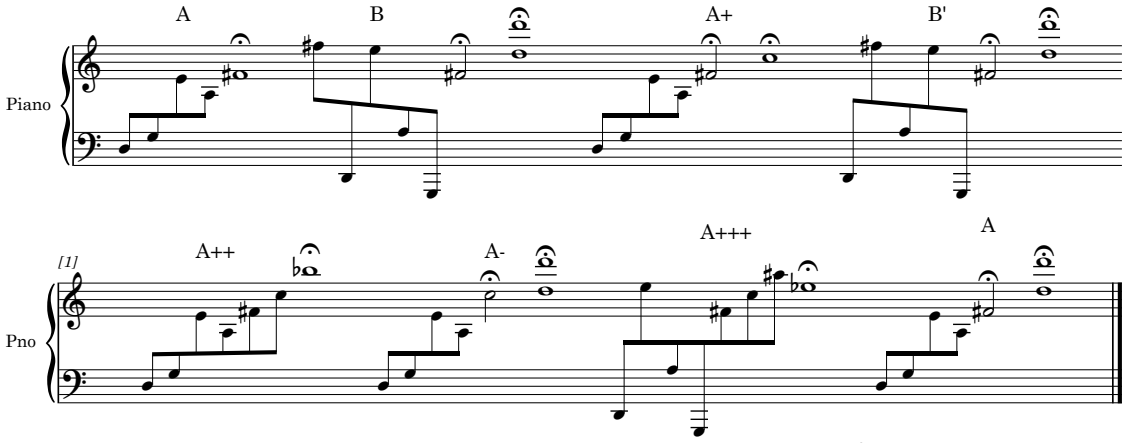
Saltiness can be further magnified through comparison. In a first impression of a musical structure, we don't immediately know what details are salient. Information is high, redundancy is low. But if that structure is repeated with one detail changed, that detail becomes extremely salient. The repeated elements become redundant and all attention can focus on the small piece of new information.

One strategy for composing vivid music is to make the details as salty as possible. Two works that seem to pursue this strategy were very influential for me: Morton Feldman's *Three Voices* and Ann Southam's *Simple Lines of Enquiry*. A characteristic passage from *Three Voices* demonstrates this well. First, the texture is simple enough that every detail can be heard clearly. Second, patterns are repeated, sensitizing attention to even minute, incidental variations. Third, changes are introduced slowly, so that their salience is magnified.



Feldman, Morton. *Three Voices*. Universal Editions, 1982, p.9.

In *Simple Lines of Enquiry,* Ann Southam creates a similar effect with similar means. In the 60 minute work, the pianist plays only one pitch at a time, so each can be given full attention. Patterns of pitches are repeated with simple transformations. This allows the connections between phrases to be easily perceived, which increases the salience of new or altered elements.



Ann Southam, *Simple Lines of Enquiry,* Eve Egoyan, piano. Centrediscs, 2015, CD. Transcription mine.

In this passage from section X., two complementary structures, labeled here A and B, evolve to make a larger phrase. A undergoes extension (+) and contraction (-), while B is re-ordered, and its closing octave D is used to close the larger phrase.

Three Voices and other works by Feldman, particularly *Piano and String Quartet* and *For Bunita Marcus*, were very conscious influences on the quiet, salty, hyper-detailed space of *Parable of the Sower.* But looking again at *Simple Lines* and doing some transcription (the score is not readily available), I see so many aesthetic and grammatical similarities to my future Bloom software that it is clearly has been the greater influence.

Roots

I latched onto the image of a flower without much thought, but it is the right image. A flower can have any number of petals, of any sizes, shapes, textures and colors.

It is recognizable because those details are organized in a way that clearly suggests eruption of a single impulse. The cone of the flower's bloom is the decrescendo of the musical one.

Bloom is a noun that is also a verb, an object and the impulse that creates it. Similarly, my blooms were first a structure defined in numerical arrays in computer code; they then became the creative impulse that has driven my work and animated my imagination for over a decade.

Bloom as a coding project was spurred in 2011 by a classic composer's crisis: the paralysis of trying to make decisions in a new aesthetic space without any established guiding criteria. In other words, trying to speak in a musical language I did not yet know. I was trying to write blooms at the piano by intuition: play a few notes, listen deeply to them, pick a few more, hold that shape in my mind, play another, see how the transition feels, try to pretend I'm hearing it for the first time like the audience, doubting, trying another combination, doubting. It was effortful and slow, but that wasn't the problem—that's composing. What felt incorrect about this way of working was the texture of my mind. I wanted this music to help evince and explore a particular mental state: meditative, receptive, calm. But my mental state at the piano was all frustrated action, doubt and effort, and it felt that while I was in that state, I could not make accurate judgments about the music and the mental experience I wanted to evoke.

I imagined an ideal creative space on a train: headphones on, eyes closed, mind calm, hands on a laptop conjuring blooms and shaping them into beautiful forms, saving the best for my piano music later. That semester my colleague Jascha Narveson helped me start programming in SuperCollider, and that is exactly what I made: blooms modeled in code, methods to generate and shape them, and a storage system, all with an interface I could control with my eyes closed on the train.

Bloom / Datatype / Prototype

A bloom is modeled by three lists: `notes`, `velocities`, `timeIntervals`. `Notes` and `velocities` are MIDI values 0-127 representing what key to play and how loud to play it. `TimeIntervals` represent the time to wait between each item of `notes`. In the prototypical usage, all three values are randomized, within constraints, with the method `.seed`.

Notes are initially limited to C3 and above; limiting low notes limits overall dissonance because low notes have more audible overtones to clash. Within that range, notes are randomized with a function that applies reciprocal distribution, which makes lower values slightly more likely. This lowers the harmonic center of gravity, which makes for more resonant structures. It also makes them more pianistic. Giving blooms a tendency to have more notes closer together discourages very wide spacing, putting the gestures somewhat more "in the hands."

`TimeIntervals` are initially restricted to a range of 0.1 and 0.5 seconds, and are also generated randomly with reciprocal distribution. Smaller values are more likely, encouraging faster flow through the notes, and greater unity of gesture.

`Velocities` are limited between 40-100, roughly ***pp*** to ***mf*** in most software instruments, also generated randomly in reciprocal distribution, and sorted loudest to softest. More soft notes than loud notes helps the loud notes be unique, which helps make the flower-blossom shape, a single source impulse (the first and loudest note) from which clearly subordinate structures emanate.

Shaping Blooms

Now that I could audition one new, never-before-heard bloom after another, I wanted to be able to shape them. I wrote methods for transforming them by performing

arithmetic operations on their three lists. Appendix A provides more complete introduction to these methods; for now it is enough to give a few examples.

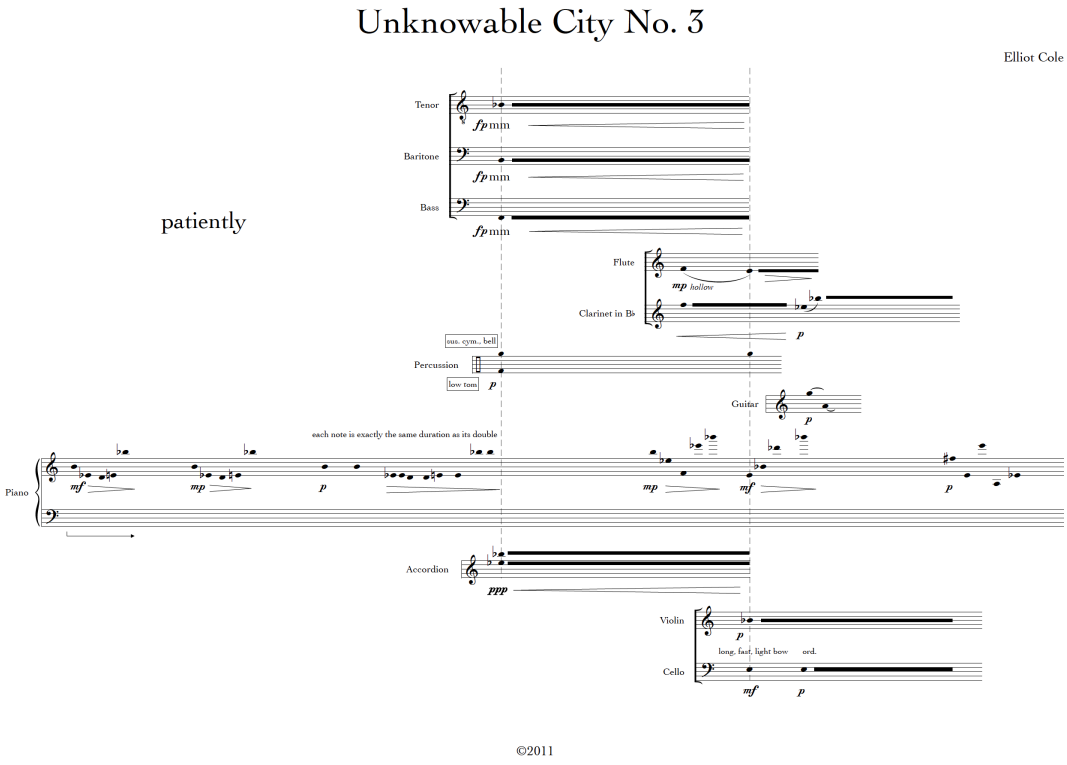
`.faster / .slower`	scale `timeIntervals`
`.rotate`	rotates `note` list
`.shear`	holding lowest note fixed, randomize octaves of pitches above
`.stutter`	double every `note`, `timeInterval` and `velocity`
`.lace(anotherBloom)`	interlace a second bloom
`.applyScale(scale)`	quantize notes into a scale
`.cast(bloom)`	apply the contour, velocities and timings of a another bloom
`.pivot`	a harmonic move that preserves intervallic content while generating new pitches
`.simplifyScale`	adjust `notes` so one fewer pitch class is present
`.report`	print the three lists, `notes`, `velocities`, `timeIntervals`
`.asPChist`	return pitch frequency as a pitch-class histogram

Storage and Interface

I wrote a class *Garden* to let me store, sequence and traverse a list of blooms. Many of the list-transformation methods for individual blooms adapted well to whole Gardens. Mapping these functions to keystrokes gave me the ability to work in the same state I intended for the music to be heard in: eyes closed, calm, receptive.

Problems of Notation / Unknowable Cities

Using this interface, I wrote three pieces that I grouped under the title *Unknowable Cities*. (The full set is 5; I included *Parable of the Sower* (2007) and *O*

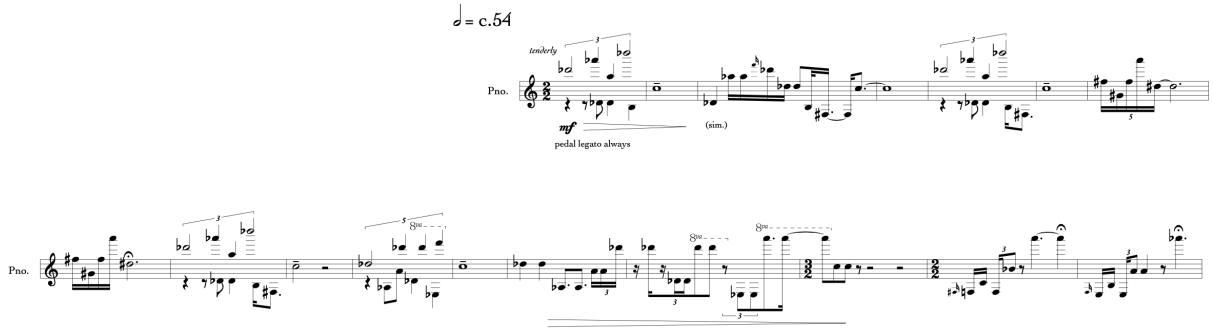


(2008), a 40 minute work for piano and double string quartet based on the chords form *Parable,* in the series as conceptual precursors.)

Unknowable City No. 3 (2011), for piano, flute, clarinet, violin, cello, accordion, percussion, and tenor, baritone and bass singers, was written for a call for scores. It was my first attempt at capturing the bloom world in notation. Rhythm is the obvious challenge. Blooms are not organized metrically, yet their timing is very precise. Repeated structures, however improvisational they sound, must be duplicated exactly. I side-stepped this issue entirely in the first half of the piece by representing no rhythmic information at all. I used an approximate proportional notation, with simultaneities indicated vertically.

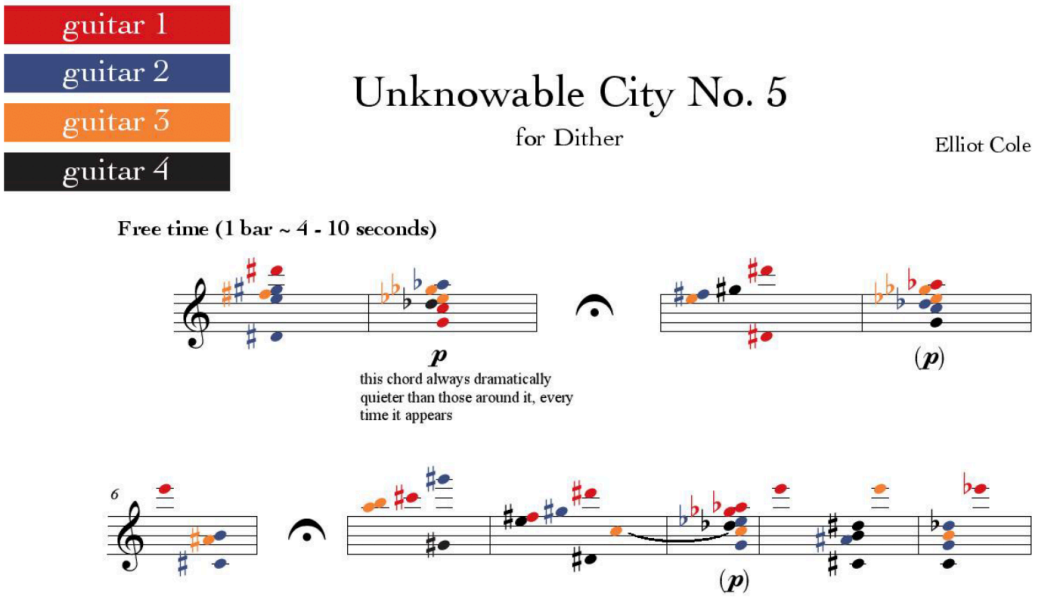
I took another approach in a piano solo later in the work, attempting to render the rhythms in traditional metric terms. Polyrhythm, tuplets, grace notes, and off-the-beat phrasing were my principal strategies for creating the rigorous yet non-metric

timings of a bloom. Its success is difficult to evaluate because the piece was never performed.

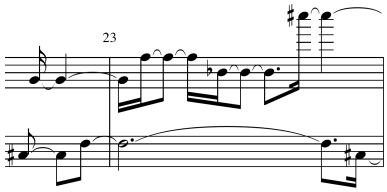


Unknowable City No. 3, piano solo

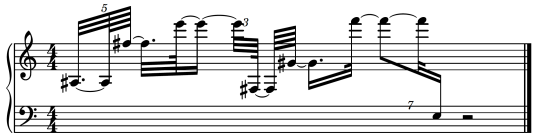
Another work from this period skates over the rhythm problem in a different way. Unknowable City No. 5 calls for sounds incapable of projecting rhythmic definition at all —electric guitars with volume pedals fading in each note. Without clear attacks, any precise alignment of notes is moot, and the score does not need to represent much rhythmic information at all. My Postludes for Bowed Vibraphone no. 2 and 7 are also blooms pieces, also for instruments without clear rhythmic attack.



Neither of these solutions were satisfactory for the precise piano music I was creating with my code. I tried software solutions, first simply using the MIDI transcription function of Finale, and then integrating my software with Lilypond via Fomus.



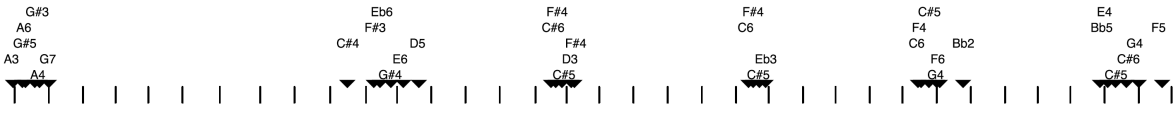
[Transcription by Finale]



[A different bloom, rendered by Lilypond via Fomus]

Both are precise renderings, in higher or lower resolution, but both are impractically over-complicated. I wanted the visual presentation to suggest a calm, clear quality of mind. Stemless notes in proportional notation, as in UC#3, seemed best, if a way to render the details precisely could be retained. The challenge would be to render the proportions with enough precision.

I wrote a program in the visual sketching language Processing to help me do this. It created a precise timeline of MIDI input, using arrows and note names. I could overlay this image on a staff in Adobe Illustrator and arrange noteheads by hand.



Had I continued in this direction, I would have had the satisfaction of knowing I had precisely expressed time, but I knew the practical outcome would not have been much different, because performers interpret proportional notation in an approximate way anyway. In retrospect, the most effective route may have been the simplest, to transcribe the gestures by ear into somewhat-quantized notation, but instead I pursued a precision I couldn't achieve. I set rhythm aside to focus on grammar.

Problems of Grammar / L-Systems

Another question consumed me: how do I make meaningful decisions about sequence? As I created, transformed and sequenced my Gardens by ear, I found my usual compositional instinct to *phrase everything* taking over. Beginning-ness and ending-ness, primary and subordinate phrases, initiation and continuation, digressions and returns—I heard intimations of all of these discursive binaries in my sequences, and I felt obligated to shape those forces into well-formed sentences, but in a language I did not yet speak.

Two domains comprise grammar in language: syntax (order) and morphology (transformation). Phrases were comprised of new blooms, "fresh" to the ear, identical repetitions, and transformed repetitions. New blooms initiate new phrases, and transformations continue them, making the most natural sentence syntax to be:

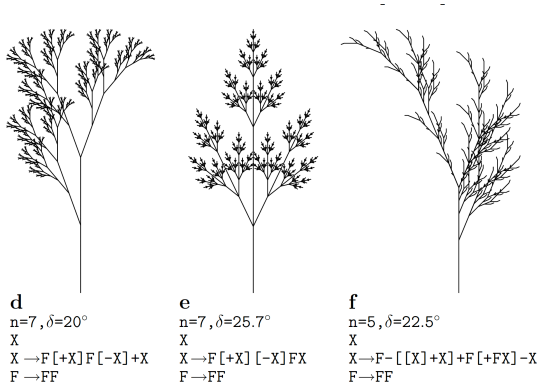
New bloom > sequence of transformations > closure

I often created closure by dissipating energy, slowing down and softening the bloom, and reinforced the boundary with a long pause. This syntax grew boring to me, however, but as I started mixing elements more freely my sense of purposeful phrasing fell apart.

I grew to regard purposeful phrasing as too human an imposition. I was trying to fulfill conventional expectations with sonic objects unsuited for the task. I returned to my initial guiding metaphor, that my job was to garden, to gently shape a process of growth largely out of my control.

Looking for a solution resonant with my core metaphor, I borrowed a model used in both linguistics and botany: Lindenmayer or L-systems.

L-systems are a variation on the idea of a formal grammar developed first by Noam Chomsky.[8] They are used in botany to model plant structures recursively.[9] I implemented a model of a garden in L-systems, depicting an imaginary life cycle of a series of blooms.



In this model, a bloom is represented by a character, signifying one of the following states:

+	growing (adding a few notes)
p	getting pruned (losing a few notes)
8	pollenating (sending copies of a few of its notes to nearby blooms)
|	splitting into two blooms
z	extending roots (starting a sequence of transpositions / transformations)
#	dead

An L-system defines a set of states, each with a set of possible transformations for each state. My "life cycle" imagines each state progressing to another state:

State	**becomes**	**new state**
+		p (probably) or [s s s] or [z z z] (occasionally)
p		+ (probably) or 8 or # (occasionally)
8		p (always)
|		[+ +]
s		+
z		+
#		#

[8] N. Chomsky, "Three Models for the Description of Language," *IEEE Transactions on Information Theory* 2, no. 3 (1956): pp. 113-124, https://doi.org/10.1109/tit.1956.1056813.

[9] Prusinkiewicz, Hanan, and Lindenmayer, *The Algorithmic Beauty of Plants* (New York etc.: Springer, 1996).

Each “season” these transformations are applied again. For example:

Season	Garden
0	+ |
1	p [+ +]
2	8 + +
3	p [z z z z] #
4	+ + + + + #
5	+ p + [s s s] + #
6	+ # [z z z] # p p p #
7	+ # [# #] [z z z z] p p # # p p p #
8	+ # [# #] s s # s p p # # # + 8 [z z z z z z z] #

The listener would hear each generation in order, a loop of blooms that grows longer and thicker each cycle.

The Siren Song of the Instant Music Machine

Conceptually, this was an exciting model because it allowed sequences of blooms to grow in metaphorically meaningful way. However, the more I worked on it, the less I found I cared about what was coming out. Committing to any particular slice of its endless and effortless output to be a “piece” felt arbitrary. My interest faded, and I produced only one finished work based on it, the electric guitar quartet cited above, UC#5. I was at a dead end.

This was an important lesson. The idea of finding an “easy way” to write music is seductive. A composer learning to code naturally hears that siren song, as their nascent powers ring with possibility. I have seen many students with fire in their eyes about the same idea: building an instant music machine.

But I found that, the more I put into a generative system, the more self-sufficient and automatic I made it, the less I cared about the music itself. No particular output mattered more than any other; I could always make more. Tweaking the program was much more interesting than the music that came out, and the moment of committing to

any particular stretch of it to be “a piece” seemed to be a horizon that always receded. Others have reported similar experiences; composer Roger Alsop describes “an endless, almost overwhelming desire to adjust the algorithm in the hope that a better composition will result.”[10]

I discovered that what made music matter *to me* wasn’t primarily its structural elegance or other abstract virtues. Creating meaning for me was the result of effort, investment, care, and intimate involvement with the music, note by note.

This revelation broke with the writing of UC#5. It uses the L-system garden grammar, progressing through several generations of evolving, pollinating and dying blooms. But when it came time to write a final version, I found myself at the piano, replicating my software ideas with a pencil, feeling each chord in my hands. Getting to know and care for each moment of the piece felt good. I stopped trying to make Bloom into an instant music machine, and instead explored ways to use it more like a hand tool.

Braids / *Bloom Suite*

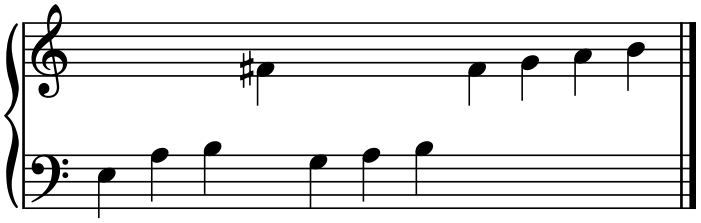
To increase my hands-on involvement with the work, I wrote `.record`, a method that let me play anything in at a keyboard and store and transform it as a bloom. A bloom was no longer necessarily a random, non-metric cluster, but became simply a musical phrase that can be transformed. I also integrated blooms with the SuperCollider `Pattern` library, a vocabulary of composable objects that produce streams of values in different ways. This proved to be a phase change: a bloom had been a bounded gesture, but now it could be spun out into an endless flow of notes. These two developments let me generalize on the ideas and infrastructure I had written, leaving the narrow aesthetic of “ideal blooms” behind.

---

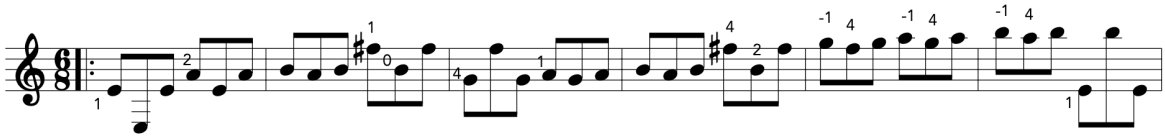
[10] Roger Alsop, “Exploring the Self Through Algorithmic Composition,” *Leonardo Music Journal* 9 (1999): p. 94.

The first work that used these expanded possibilities was *Bloom Suite* for solo guitar (2013). It remains one of my most persuasive and popular pieces. A deeper analysis of the work provides a good point of entry for the new approach as a whole. The core material of Bloom Suite is a sequence of 24 blooms. Some were recorded in by hand at the piano, others are transformations of them. Movements 1, 2 and 4 are the same sequence of pitches read out in different patterns.

In movement 1, the core material is read out in a "sliding window" pattern 3 notes long. In this pattern, the sequence ABCDEF becomes ABC BCD CDE DEF etc.—iterating through the list in windows of 3. (Begin looking at the fifth note; the initial four involve the tail end of the cycle wrapping on itself).

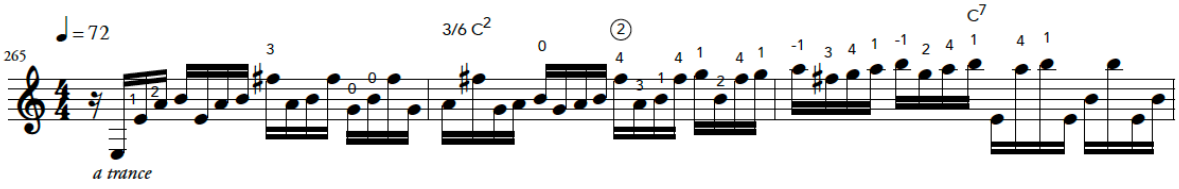


Bloom Suite core material, first 11 notes.



Bloom Suite, movement 1. Core material through sliding window size 3.

Movement 4 is the same pattern but the window is 4 notes long: ABCD BCDE CDEF DEFG.



Bloom Suite, movement 4. Core material through sliding window size 4.

I created movement 2 in a more freewheeling fashion. I wrote patterns that read out notes from the complete sequence of blooms by large leaps, e.g. every 10th note, every 50th note, every 157th note. The sequence wrapped in a circle, so these patterns resulted in unpredictable cross sections of the material. I tried many possibilities, looking for sequences of pitches that caught my ear, recording them to MIDI and editing them later into a loose rondo form. Movement 3 was freely composed for contrast.

The "slide" patterning of movements 1 and 4 was particularly enchanting to me. It is an object lesson in redundancy. In these patterns, every note (after the first two) recurs a fixed number of times at fixed intervals. Consider C in this sequence, a sliding window 3 notes long: ab**C**b**C**d**C**dedef. Its first appearance is echoed twice, interlocked with d and its two echoes, interlocked with e and its two echoes, and so on. It naturally projects triple time and an echo's decay, with the first appearance of a note on the strong beat, its first echo landing on the weaker third partial, and the last echo landing on the weakest second partial.

The effect is striking: a free-flowing stream of notes with many surprising harmonic twists and turns, but the harmony always feels grounded and a wrong note is immediately evident. Harmonic grounding is usually vertical, but this is purely horizontal. Errors are evident because the pattern creates a predictable rhythm of information and redundancy. The ear quickly grasps the pattern: [new note, repeated note, repeated note], or [information, redundancy, redundancy]. A stray note either immediately violates this pattern, or the checksums that follow fail to confirm it.

Above, I drew the insight that musical order is a shared responsibility of the dimensions of music, and that they can share that responsibility unequally. A similar lesson can be drawn from thinking about redundancy. We need a certain amount of redundancy for music to make sense, but where it is located or how it is created can shift. A hundred violins can improvise in C major in total rhythmic and motivic chaos, but the redundancy of timbre and key will more than suffice for it to be beautiful. Each member

of a jazz big band can play random notes, but if they play every note in unison rhythm and articulation, it will be legible (and likely thrilling). The lesson I took away: if my music wants to be wild in one dimension, make it redundant in another.

Mirrors / *Facets*

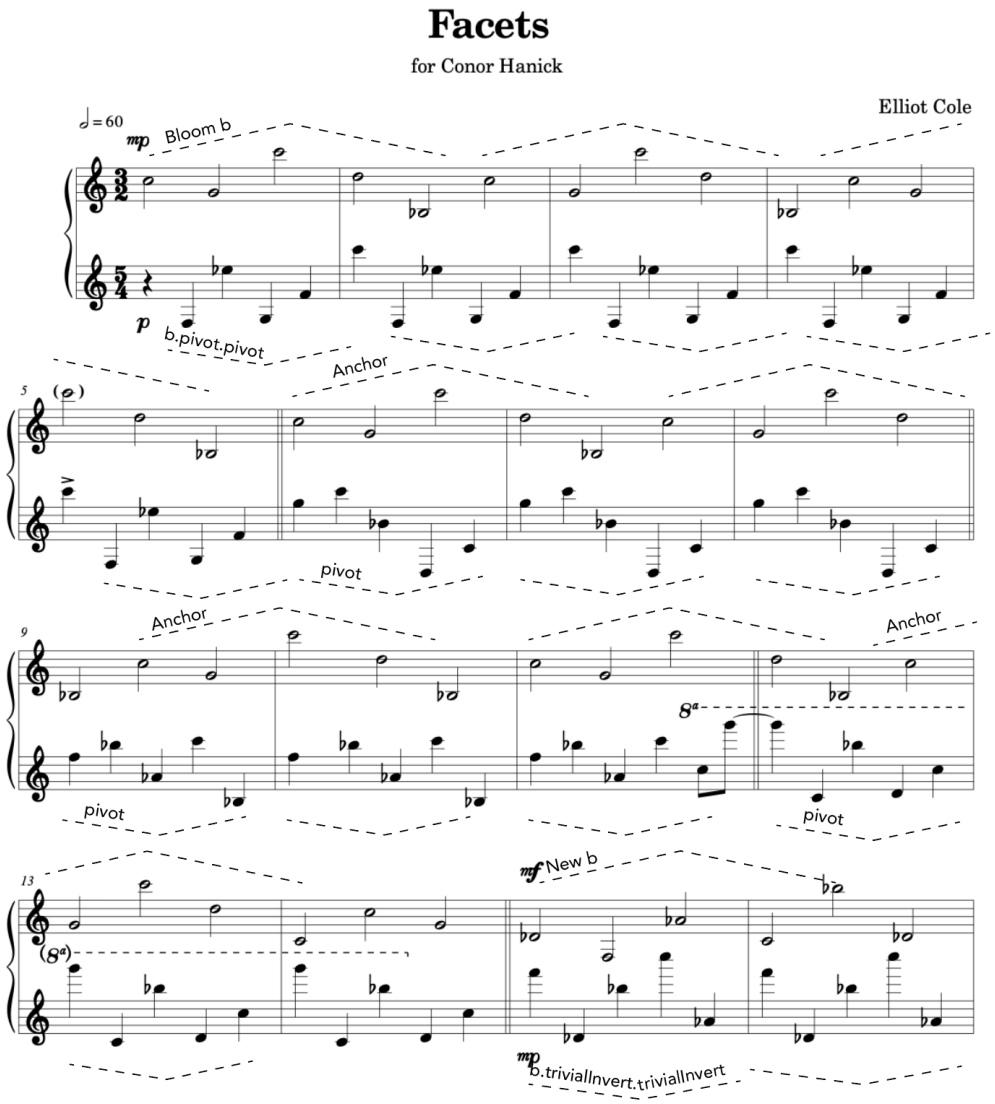
Facets for solo piano (2013) was the second piece I wrote with the expanded Bloom software. I consider it to be the most beautifully constructed piece I have written, and have returned to it twice, first as a chamber orchestra song *Well of Whales*, and later as a percussion quartet.

It is constructed in two layers, one for each "hand" (in practice, the resulting patterns need to be broken up variously between the hands). In each passage, the right hand loops through a bloom, reading every Nth note. The left hand reads through that blooms "pivot-shadow"—a second bloom derived from the first by the `.pivot` method—in a 5:3 polyrhythm.

`.pivot` is a harmonic transformation I learned from the composer Quinn Collins. It works like this: take a chord, invert it (move the bottom pitch to the top), and then transpose the second chord down such that the top note of both chords is the same. The result is a different chord, often with many different notes, but it shares the same top note and the same interval vector. The top note functions as a common-tone to smooth the often distant harmonic move. The interval stack of both chords is the same except that one interval is inverted. This preserved interval vector creates a sense that, even with all new notes, the harmony has in some deep way not changed—another example of how I am charmed by seeming impossibilities. It is like a wormhole to a parallel universe; you travel instantly to nearly identical structure in a distant harmonic space. A chord can be pivoted in this way into as many new chords as it can be uniquely inverted.

The piece begins with one 5-note bloom in the right hand and its pivot-shadow in the left. Of its 5 possible pivot-shadows, I chose by ear what I considered the most

beautiful when combined with the right hand. Then the left hand pivots to another set of notes. One layer is the anchor while the other changes. In this way the piece evolves smoothly, nearly always bridging transitions by anchoring one pattern while pivoting the other. In each case I chose which `.pivot` possibility by ear, and occasionally employed other transformations like mean-inversion, pivoting around the bass instead of the soprano, transposition, and `.trivialInvert`, chord inversion the way we first learn in school, moving the lowest pitch up by octaves until it is the highest.



The patterning is complicated by the fact that each hand often walks through the bloom

in a different order. Imagine the left and right hands, then, taking different walks through a core sequence of pitches and its shadow, each taking turns holding while the other shifts. The result is a tightly-woven fabric of reflections, a tunnel of mirrors.

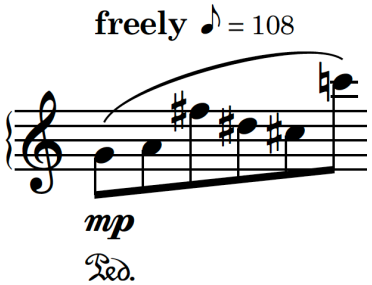
Expanding *Facets* into the orchestral song *Well of Whales* was a study in pareidolia, our tendency to impose meaningful interpretations on ambiguous stimuli, like the way we often see faces in inanimate objects or see animals in clouds. Because each hand's patterns are full of wide leaps, there are no deliberately-composed linear structures. I never wrote a melody. And because of the music's essentially flat and unchanging texture, there is little rhetoric to signal what notes are important or connected, or what a note is intended to mean. And yet the ear cannot help but connect the dots in its own way, bringing out melodies and hierarchies, characters, affects, and meanings. My approach to orchestration was to listen for the connections that my ear drew implicitly and draw them explicitly. Out of the incidental interactions of two patterns I drew a rich polyphony implied voices, emerging and submerging like threads running through a tapestry.

Stutters / *Two Nocturnes*

Two solo piano pieces were directly inspired by sounds that came out of blooms, but I composed them away from the computer. The idea was sparked by the `.stutter` method, which doubles each note along with its velocity and duration. For example:

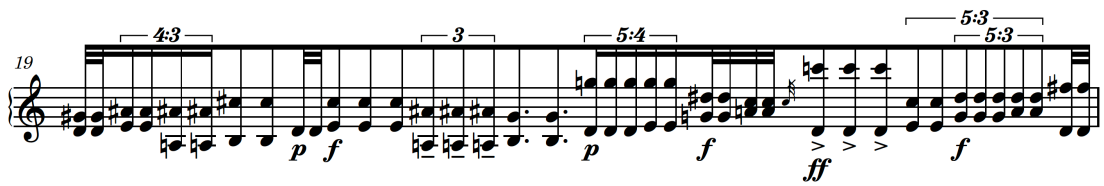
```
b = Bloom.new.seed
notes:            74   95   98   78
velocities:       93   84   76   56
timeIntervals:    0.33 0.21 0.46 0.12

b.stutter
notes:            74   74   95   95   98   98   78   78
velocities:       93   93   84   84   76   76   56   56
timeIntervals:    0.33 0.33 0.21 0.21 0.46 0.46 0.12 0.12
```

Applied several times to blooms with random `timeIntervals`, it results in a striking rhythmic effect. Each new pitch sets a new tempo; the pitch repeats some number of times exactly in that tempo, then switches to an unrelated tempo with the next pitch. It reminds me of an Escher painting; lots of very straight lines (the regularly repeated notes), but at each corner the perspective changes. I approximated these rhythms at the piano, charmed by the tension between the improvisatory, imprecise rhythmic shifts and the rigid reputations of each notes. It's another example of "if you want to do something wild, reinforce it": I could fill the music with frequent, random tempo shifts, but by immediately confirming each with a series of redundant notes, it projects a rigorous approach to time.



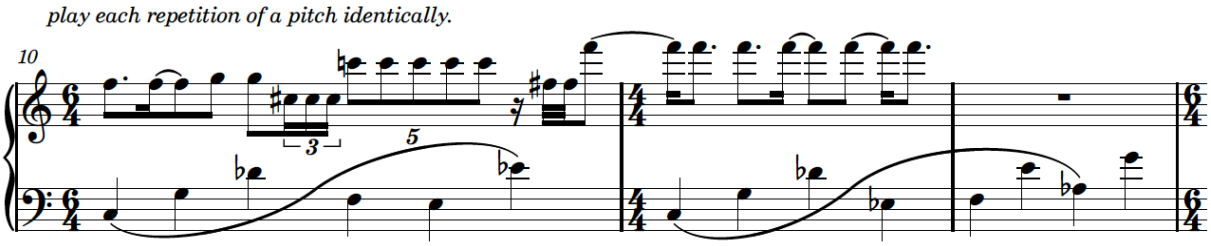
It begins with melodic/harmonic *idée fixe:* a [0, 2, 11] set repeated at the tritone, established in the opening gesture and developed in a bloom-like manner: a series of transformations inspired by `.stutter`.

I approximated the irrational tempo relationships in traditional rhythmic notation, using tuplets, sometimes nested, grace notes, and quick leaps around the subdivision tree (e.g. 32nd notes followed by 8th notes without 16th notes in between). Beaming across whole phrases, rather than by beat, further discourages the performer from relating all notes to a common pulse, as the notes of a bloom are not related to a common pulse. Perfect execution of the changing rhythmic relationships is nearly impossible by design, resulting in the irrational tempo relationships I desired. The



performer is told that rhythmic relationships are approximate except that repeated notes must be identical.

The second *Nocturne* places the same ideas in a different context. A walking bass, reminiscent of the left-hand parts of Chopin, articulates a steady pulse. The `.stutter` inspired right hand, still playing transformations of [0, 2, 11], floats against it. Widely varied rhythms and off-the-beat phrasing (as in the final F6s in this example) contribute to an illusion that the right hand's rhythms are not related to the steady pulse.



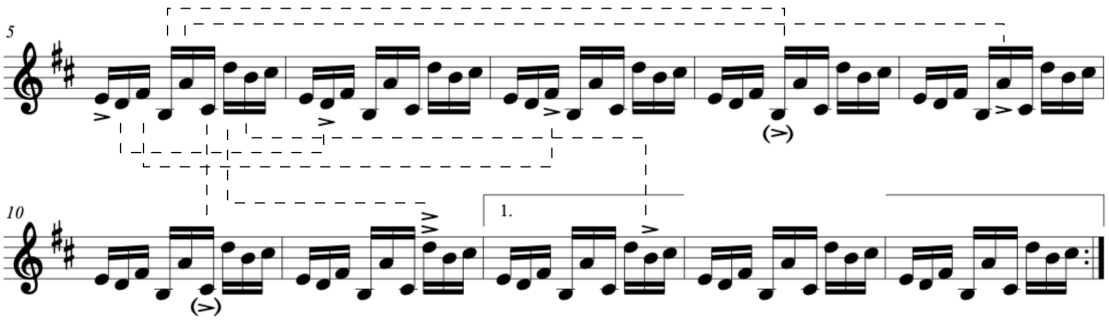
Neither piece was written with the Bloom software, but neither would have been written without it. This underscores my opening assertion that we should think about writing software primarily as a practice for opening new pathways for creative thought. If in the process we make useful tools that are also functionally useful "labor-saving devices" that is a bonus, but not the only value of the work.

Wheels Within Wheels / *new year's day*

Two pieces focus on another rhythmic effect discovered in bloom pattern play. I call it the "wheels within wheels" effect: a set of notes `n` items long is looped, and an accent is placed every `n+1` notes. The result is the same cycle of notes articulated at two different speeds, a wheel within a wheel, as in *attention,* the first movement of my solo vibraphone suite *new year's day:*

Note cycle: 9 notes long (fast level)

Accent cycle: 10 notes long (slow level)

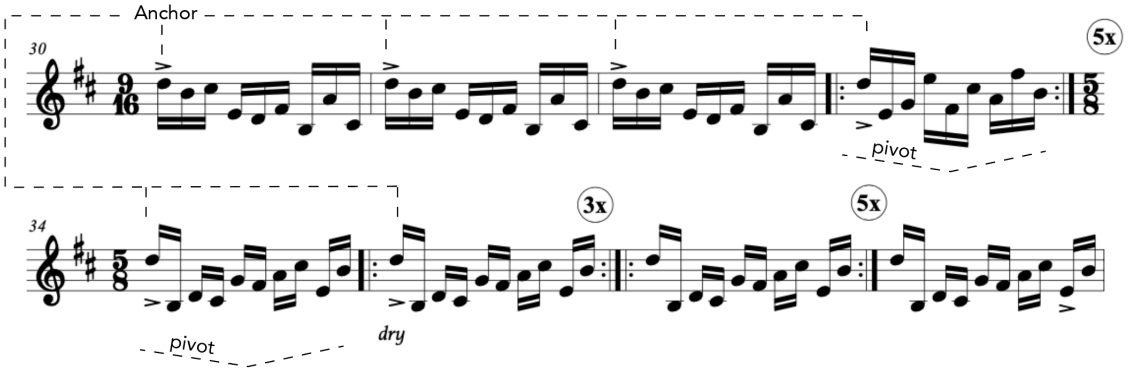


The accented notes play the same melody as the 16th notes, but at a 1/10th the rate. We perceive the accented notes in their own stream, on a separate plane, creating the illusion that there are two lines being played out of one.

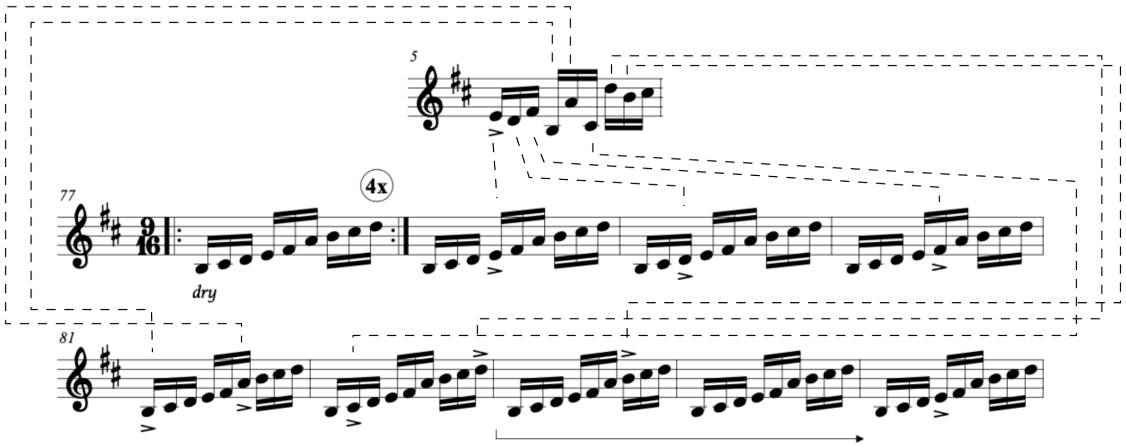
This effect charms me for a similar reason that the `.stutter` effect does. It is an auditory paradox, fusing oppositional forces. `.stutter` creates simultaneous impressions of imprecision and precision. Wheels Within Wheels is a paradox of stasis and change: the pitch sequence does not change, but feels always new. The melody is heard again and again with a constantly shifting focal point. The changing position of the accent re-contextualizes the sequence both rhythmically and harmonically, destabilizing our metabolism of the repetition and refreshing it in our attention.

This movement of *new year's day* was created in my software. Each cycling pitch set is a bloom, and the evolving sequences are created by method transformations: sorting and scrambling, pivots, transpositions.

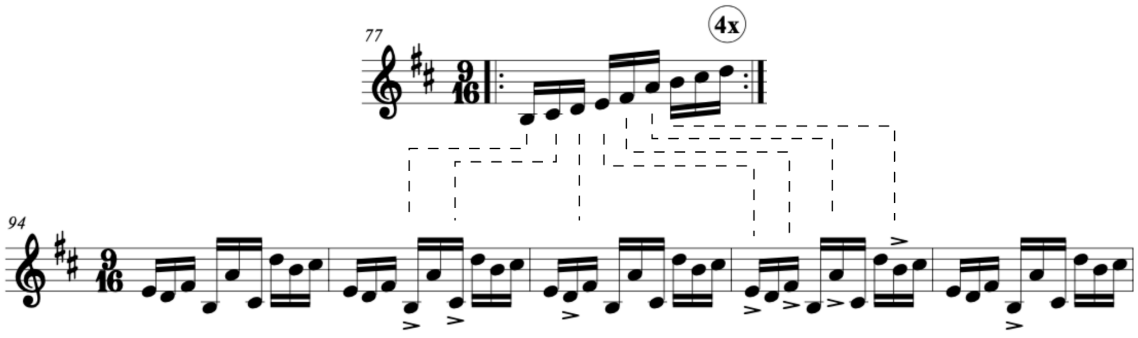
Recalling the bridging technique I found in *Facets* of preserving one pattern while altering another, I use the accent patterns to create bridges, as in this passage where the accented D5 pitches act as a anchor point against which the pattern can shift in mm. 33 and 34.



This idea is abstracted in a later section to pivot around thematic material. In the passage mm. 77-83, the 16th note cycle is comprised of the same pitches as in mm. 1-14, but they are sorted into an ascending scale. The accents are placed in their original order from the start of the piece, E D F# B A C# D B. The slow sequence of accented notes is preserved, the fast melody is re-ordered.



The parts are flipped once again in the final passage, in which the theme is restored on the fast level, but now the ascending scale is brought out on the slow level.

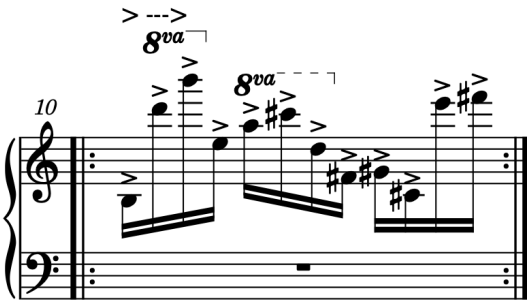


A full map of these strategies in *Attention:*

Measure	5-14	37-48	78-91	94-120
Fast	theme	scrambled	sorted	theme
Slow (accents)	theme	theme	theme	sorted

Wheels Within Wheels / *Flows*

I pursued the same texture more extensively in *Flows* (2017), a large work for solo piano. In this piece, each bar is to be repeated as many times as the performer chooses (the first performance last 46 minutes). I used a shorthand notation for the wheels within wheels pattern: an accent and an arrow above, instructing the performer to loop the bar, playing the first accent on the first loop, the second accent on the second loop and so on:



Changing the accent shifts the point of focus within a pattern, creating an illusion of change. It was entirely composed in software, with all material generated and developed by method transformations, but in a hands-on, improvisatory way. I worked bar by bar, coding transformations live while the patterns played. When I found a form I liked, I committed it to the score.

Outdoor Games / *Flowerpot Music Books*

The *Flowerpot Music Books* are an evolving body of pieces for large groups of people equipped with flowerpots and mallets. Iterations for students at College of South Idaho and New York University in 2018 led to its widespread production on Make Music Day, June 21st, 2021. Over 750 participants in over 40 cities performed a version of the work.

Flowerpot Music Books are collections prose scores for games and activities. There are a variety of approaches, but many of them use the bloom gesture as the building block. The players form a network animated by spreading activation: each only plays one note, quickly after a nearby player plays a note, a bloom rippling through a room like a neural network or the stadium game The Wave. The score is written in prose that is deeply influenced by coding, including methods, variable storage, and a semi-rigorous syntax.

Pitch methods from my software (like `.pivot`) aren't possible because Flowerpot Music is pitch-agnostic; each player brings a pot, and I don't know what note it is. But I found simple methods of time and volume were easy for a group to reproduce. For example:

exactly (attempt to repeat the bloom exactly as it was last heard)
slower and *faster*
softer louder decrescendo crescendo
smoothTime (attempt to even out `timeIntervals`).

The score also features methods for transforming the new dimension, space, for example:

shuffle the map (try to keep reproducing the bloom exactly while you walk to a different part of the room)

traveling unisons (two people strike in unison, move to new positions, repeat)

spread out / *converge* (slowly walk from / toward a central point)

The group also can "store" and "recall" blooms like variables, as in the piece *Undo*. The following instructions (from *NYU Book*) direct the group to develop and refine a bloom, store it in A, and then alternate between A in full form and weekend, thinned versions of the same. [>A] means "remember what just happened and reproduce it exactly when you see [A]." An ellipsis is a long pause. The statements in between modify and trigger a bloom.

Undo

(everyone) new bloom decrescendo… faster…
make more beautiful… make more beautiful… exactly… [>A]

…

10x ||: [A]… softer & some people don't play… :||

Compare another prose score with a similar piece in code:

Unfolding

(player 1) slow toll 3,

do the phrase: [new fast bloom decrescendo... exactly... exactly... slower... softer... ...]

do for: (players 1-5, 1 leads)... (players 1-10)... (players 1-20)... (players 1-20 thick) ...

5x (overlapping) ||: (all sticks! leaders, in order: 1, 7, 11, 13, 19) new fast decrescendo bloom,

:||

4x ||: (players 1-5, all heads! different leader for each) new slow bloom decrescendo... :||

(player 1) toll 3 ... ...

Unfolding in code

```
b = Bloom.new([60, 60, 60], [3]).play   // “toll” 3 middle Cs
b.seed.faster.play;        // new fast bloom (decresc. by default)
b.play;                    // exactly
b.play;                    // exactly
b.slower.play;             // slower
b.softer.play              // softer

b.thin.thin.play;          // only a few notes
b.thicken.play;            // add more
b.thicken.play;            // add more

b.chan_(1);                // switch instrument = “all sticks”, MIDI channel 2
Task {5.do {b.scramble.newShape.chan_(1).faster.play; 2.wait}}.play;

b.chan_(0);                // switch mallet sound back (all heads!)
Task {4.do
       {b.rotate(10.rand).slower.slower}.playWait; 5.wait;
       // rotate is functionally a ‘different leader for each’
}.play;


b = Bloom.new([60, 60, 60], [3]).play   // “toll” 3 middle Cs
```

Like *Nocturnes, Flowerpot Music Books* don’t employ computers at all, but programming opened the creative pathways I followed.

Improvisations / *Hashflowers*

Hashflowers is a body of recordings and the method for creating them that I pursued in 2021. This method involves a kind of virtual prepared piano that combines serial determinism with free improvisation, and is built around the bloom object. Here, "hash" primarily refers to a cryptography operation in which any data can be reduced to a 64-digit hexadecimal signature that can be used to verify that data. The double entendre with hashish is welcome, as the music rewards languid contemplation.

To translate a hash into music, I converted each digit to a scale degree, and used the scale mapping function of the bloom object to create a fixed loop of pitches. Then I wrote a program that, listening for any key on an electric keyboard, passes the next note of the hash row instead. This note is transposed to the nearest octave to the note physically played on the keyboard. The pianist is now not responsible for pitch, but can create gestures in rhythm and register.

Instrumentation is also serialized. I loaded seven software instruments—a mix of contrasting pianos, celeste, and bells—on different MIDI channels. Then I patched each instrument through a different path through the studio, through contrasting amplifiers and compressors, to make their characters even more distinct. Each was also sent through different reverbs and delays, so that each instrument projected a contrasting sonic space. The bloom object holds an array for MIDI channels alongside pitches, velocities and time intervals. This array of values functioned as a switchboard, deciding which note would be routed through which instrument. In practice, this meant that I could play a chord on the piano and each note would be rendered by a different instrument in a different sonic space. Finally, all of the instruments were summed and routed immediately through a mastering chain of analog compression and EQ.

I wanted to create a workflow very different from my usual cycle of drafting and revision. Instead, I aimed to design an instrument so unfailingly attractive that I could improvise directly to a finished product. It's an unpredictable one: when I play a note, I

do not know what pitch will come out, or what instrument will play it. But by carefully tuning all of the elements beforehand I can stay in territory that excites me.

Once set up, I improvised short pieces, mostly between 1-6 minutes long. Some end when the 64 pitches of the hash have been exhausted, and others loop that sequence. The improvisation happened at the piano but also on the hardware effects, including an OTO BIM delay, BAM reverb, a Strymon Volante tape delay, and Chase Bliss Mood and Blooper pedals, all of which I manipulated in real time to alter the character and destiny of the notes. The Blooper is a pedal which records, loops and layers past audio, and it made an important syntactic and rhetorical contribution. My improvisations began with the hash row, performed in improvised gestures at the piano, while also playing the parameters of the delays, reverbs and looper. Then the focus shifts to the contents of the looper itself, an accretion of memories floating in the background. It is an agent of cohesion, a circular time to balance the row's time's arrow. It's also a canvas within the canvas, where beautiful surprises often emerged.

These are not 'classical bloom' pieces for several reasons. First, there are many more modes of gesture and flow than the bloom gesture. Second, there are no method transformations of the bloom, only a linear walk through the pitch set. Third, I worked in fixed diatonic spaces only, rather than the chromatic clusters that attracted me at first. Nevertheless, I feel that these pieces represent my initial aesthetic vision more successfully than any others. For a listener, they are both austere and voluptuous, unified and kaleidoscopic. They create "the charm of impossibilities." And for the creator, they offer a serene and gratifying space to work, where micromanaging is impossible, and unexpected beauties arise endlessly.

Bloom Praxis in Context

The Bloom software "under the hood" unites two major traditions of musical thinking, but "behind the wheel" it prompts a creative process distinct from both.

The Parametric Tradition

The first lacks a common name, but I suggest it can be called the *parametric* tradition, which is at its core the insight that the music is a conjunction of separable parameters.

The most common form of this idea is to separate pitch and rhythm and have one or both repeat in cycles. Independent cycles of rhythms appear in the idea of rhythmic modes that organized the music of ancient Greece and the middle ages, for example in the Notre Dame School in 12th century Paris. Dance music generally is also grounded in repeating loops of characteristic rhythms. Independent cycles of both parameters, *color* (melody) and *talea* (rhythm) form the backbones of French motets of the 13th and 14th century, as well as modern polyrhythmic techno. In India, the independence of *shruthi* and *laya* (tone and rhythm, considered the 'mother' and 'father' of music) is deeply explored in extensive systems of *raga* and *tala*. When these cycles are of different lengths, the result is *variation-through-repetition*, as elements are simultaneously in a familiar context horizontally (in a correct place in sequence) and a novel context vertically.

Parameters are also frequently transformed arithmetically. Addition and subtraction are most common operations in the pitch domain—transposition. Rhythms are more commonly multiplied by simple integers and fractions, as in prolation canons, in which a melody is accompanied by itself played at a different speed. Rhythmic multiplication is endemic in Carnatic music education, which trains musicians to play all phrases in multiples of 2 and 4 from the first exercises.

In the 20th century, many Western composers pursued new approaches to structure. The novelty was superficial, however, as their dominant claim to innovation, serialism, is a simply a generalization of this tradition. The twelve-tone music of Arnold Schoenberg and his disciples organized pitch in cycles that were transposed, inverted, and reversed. Messiaen was the first to extend this organization to parameters beyond pitch and rhythm. In his piano work *Mode de valeurs et d'intensités,* which set the agenda for this approach, four parameters are strictly ordered: pitch, duration, volume, and articulation. The strict orders are then permuted so that every event is a novel conjunction of parameters, so that "no combination of dynamic, duration, touch, or pitch is repeated within this mode."[11] When the possibilities for novel events are exhausted, the piece ends.

A system that optimizes for novel events creates problems for listeners. The fact that audiences found it so hard to understand this music can be explained by Shannon's theory of information. To maximize novel events maximizes information and minimizes redundancy, which maximizes uncertainty. In an interview in 1933, Arnold Schoenberg was asked to explain what makes his music so difficult to understand. He could have pointed to the high level of dissonance, 12-note themes, or hard-to-perceive thematic transformations, but instead he framed it in terms of repetition:

[11] Tu, E. (n.d.). Messiaen and Math: Mode de valeurs et d'intensités. Retrieved February 28, 2021, from https://messiaenandmath.tumblr.com/modedevaleurs

> Schoenberg: The difficulty for the public to understand my music is the conciseness and the shortness.
>
> Interviewer: You mean that you speak musically in a kind of aphorism, epigrams — I almost said enigmas, Mr. Schoenberg.
>
> Schoenberg: Yes, my works are apparently enigmas to many people, but there is an answer to all of them. What I mean is, I never repeat. I say an idea only once.[12]

The desire to maximize variation in a systematic way was not just a preoccupation by the serialists. Lou Harrison, whose aesthetics were in many ways diametrically opposed, describes a similar procedure in his *Music Primer*.[13] He describes a process, used in his *Pacifika Rondo,* of employing repeating cycles of numbers to determine where to operate on a note, for example to end a phrase, apply an articulation, or vary a rhythm. To maximize the distribution of these variation-operations, he chose a sequence of prime numbers and re-ordered them for each operation so the cycles minimally coincide.

Bloom is clearly rooted in this tradition. Pitch, velocity, duration, instruments/ articulations are defined as separate lists, and subject to separate transformations. Rhythmic modes, isorhythm, prolongation canons, and serial operations are easy to reproduce. But it diverges in that it is not designed to maximize variation. It does not prefer information or redundancy. It also diverges in its use of chance.

The Nature Painters / Cage and Xenakis

Bloom unites this parametric tradition with the embrace of chance pioneered by John Cage and Iannis Xenakis. By yielding some compositional decisions to chance, these composers represent a major fork in the tradition of musical authorship.

The standard assumption in the West is that a piece of music is a window into a composer's mind, expressed through their expertise in the language of melodies, rhythms, chords, drama, and the rest. This composer understands the sonic effects of

---

[12] Alfred Lundell: Interview with Arnold Schoenberg, accessed February 27, 2021, https://www.schoenberg.at/index.php/en/alfred-lundell-interview-with-arnold-schoenberg.

[13] Lou Harrison, *Music Primer; Various Items about Music to 1970* (C.F. Peters Corp., 1971), 2-3.

their materials and, in their imagination, crafts a specific experience for a listener by making skillful choices. This model is often associated with the Romantic period, which valorized it in near-religious terms. But employing chance allows sounds to arise without being imagined, and without skillful choosing. The piece is no longer a window into a composer's mind but a window opened by a composer onto something else.

This a break in the musical tradition, but it's not unfamiliar in the visual arts. Compare it to nature painting. A painting of a tree is a window to that tree; whether that particular view actually exists or not, it is a project primarily grounded in observation, not introspection. The person of the painter is expressed in their selection and style of representation, what is often rounded up to an almost mystical quality: "the way the painter sees." The painter filters a view through their taste and skill, but they are putting a frame around nature.

Both Cage and Xenakis connected their music to nature and natural laws. For Cage, the connection is explicit and primary: "the function of art is to imitate nature in her manner of operation."[14] Xenakis' focus was to faithfully emulate nature's "manner of operation" by calculating musical events from fundamental laws of statistics.

For Cage, one essential feature of nature is that it is indifferent to us. Events arise outside of human intention. To imitate this in music, he explored compositional processes grounded in nonintention. He saw it as personal form of Zen practice, "as strict as sitting cross-legged…the shifting of my responsibility from the making of choices to that of asking questions."[15] These methods included: tossing I Ching coins, as in *Music of Changes* (1952) and many other pieces, marking blemishes on a sheet of paper (sections of *Concert for Piano and Orchestra*, 1958), overlaying patterns on transparencies to see where they intersect (*Cartridge Music*, 1960), and overlaying star

---

[14] John Cage, *A Year from Monday New Lectures and Writings* (Middletown, Conn: Wesleyan Univ. Pr, 1985), 31.

[15] Kay Larson, *Where the Heart Beats: John Cage, Zen Buddhism, and the Inner Life of Artists* (New York: Penguin Books, 2013).

charts on score paper (*Atlas Eclipticalis* (1961). “Cage saw [the I Ching] as a means of getting outside the mind altogether, a way of allowing nature, the environment, or what Zen would call Mind with a capital M, to respond to his compositional questions.”[16]

For Xenakis, the essential feature of nature is that its variety unfolds from fundamental, universal laws, which can be discovered through science. He considered laws of chance to be among a trinity of universal primitives: “our universe is formed from grains (of matter) and straight lines (photon radiation) ruled by stochastic laws (probability).”[17] He cites “the collision of hail or rain with hard surfaces” and “the song of cicadas in a summer field” as models for his orchestral textures, in which a large masses of sound are built out of independent points, each behaving according to probabilities.[18]

But for Xenakis, probabilities demand strictness. His chance is never about releasing control, or making choices easier. “Chance is a rare thing and a snare. It can be constructed up to a certain point with great difficulty, by means of complex reasoning which is summarized in mathematical formulae; it can be constructed a little, but never improvised or intellectually imitated.”[19] He scorned the improvisational, free-choice approach to indeterminacy of his peers (I assume he is referring to the mobiles in Boulez’s *Third Sonata* or Stockhausen’s *Klavierstück XI),* stating that they commit “an act of resignation” and it doesn’t “give them the right… to speak of chance, of the aleatory." In this light, Cage’s quest to exert less and less control over his music would be seen as the greatest resignation of all.

The significant differences in their conceptions and uses of chance can be viewed as oppositions, or as complimentary positions within a larger unity. Fundamentally,

---

[16] Christopher Shultis, “Silencing the Sounded Self: John Cage and the Intentionality of Nonintention,” *The Musical Quarterly* 79, no. 2 (1995): pp. 312-350, https://doi.org/10.1093/mq/79.2.312, 318.

[17] Xenakis, Brown & Rahn, “Xenakis on Xenakis,” *Perspectives of New Music, 25,* 1/2 (1987), 36.

[18] Iannis Xenakis and Sharon Kanach, *Formalized Music: Thought and Mathematics in Composition* (Styvesant, NY: Pendragon Press, 1992), 9.

[19] Xenakis, *Formalized Music,* 38.

Xenakis, like Cage, felt called to "imitate nature in her manner of operation," in pursuit of universality/non-duality. Xenakis was convinced that "one can achieve universality… through the sciences," and, as sciences discover and then deduce from fundamental laws, the composer should as well. "In musical composition, construction must stem from originality which can be defined in extreme (perhaps inhuman) cases as the creation of new rules or laws, as far as that is possible." He clarifies "as far as possible" in a way that resonates with Cage: "as far as possible meaning original, not yet known or even foreseeable," a near-perfect echo of Cage's definition of experimental music as one "the outcome of which is unknown."[20] Xenakis continues: "Construct laws therefore from nothing, since without any causality. [sic]" One wonders whether the grammatical error is a sort of pun, since it renders the second clause an effect without a cause, but in any case, the spirit of starting from scratch with invented rules is pure Cage, and, though he may not have intended it this way, negating causality is pure Zen.

Both composers used chance to shift the frame of their compositions away from their subjectivity, taste, and skill, towards a vision of an objective, universal, nature.

From Painting to Foraging

By uniting chance operations with parametric transformations, bloom praxis diverges from the paradigms established by Cage and Xenakis in three areas: taste, control, and procedure. If Cage and Xenakis are nature painters, composing with blooms has made me a forager and a cook.

First, in my approach, taste is valid. It is not a subjectivity poisoning pure nature, but an indispensable part of making it meaningful. Working with blooms involves constant listening, assessment and decision-making. A random bloom presents a shape; the composer reflects on its sound and decides whether to discard it, keep it or 'cook it'

---

[20] John Cage, Experimental Music: Doctrine. In *Silence: Lectures and writings (*Middletown, CT: Wesleyan University Press, 1961), 13.

by transforming it, testing alternate possibilities, spinning it into patterns. Rather than pure expression of self or pure depiction of nature, composing becomes a hunt in the wilds of randomness for particularly enchanting structures, plus the development of particularly tasty recipes. If Cage is a mystic and Xenakis a scientist, this is for epicures.

The second proceeds from the first: to act on taste is to control the sound. Non-control of sounds was central to Cage's desire to be a non-aggressive witness in nature. Xenakis controlled all sounds to make sure they behaved as if they were controlled by natural laws. Blooms work in a loose middle ground, where the material can be steered on a high level but micromanagement is discouraged. Transformations operate exclusively on collections; there is no explicit method for changing a particular note of a bloom. It can be done, but trying to write music that way is painful. Many methods involve random change, like `.shift`, which alters all pitch, time and velocity values slightly, or `.shear`, which randomizes the octaves of the notes. It is not designed to help you be more specific about what you want, it is designed to help you make broad changes and see if you like them.

Third: controlling sounds by taste is an open procedure defined and redefined as it unfolds. Both Cage and Xenakis devised closed procedures: they are defined first and followed strictly to construct the piece. Cage's pieces are frequently 100% the result of this; Xenakis' biographer estimates "75%."[21] These procedures are algorithms, often linear enough to be performed by a computer. Indeed, both used computers to speed their calculations at different times in their careers, although both seemed to enjoy the physical industriousness required by working them out by hand (both composers were expert architectural draftsmen). This is the experimental mindset, a methodology borrowed from scientific research. First design the experiment, then execute it carefully and observe the result. Intervening to change the procedure in the middle of the experiment compromises the experiment.

---

[21] Nouritza Matossian, *Xenakis* (London: Kahn & Averill, 1986).

Xenakis referred to the experiment as the "scheme," saying the modern composer invents schemes where formerly composers only invented "forms."[22] As a summary of history this is debatable, but the distinction is useful. The scheme is an idea, an abstraction, from which a form can arise.

The Bloom software is full of such schemes. Every transformation is an abstract scheme, defining no content but a relation between before and after. But using those schemes invites a process that is itself not a predetermined scheme. It better resembles the improvisatory play of the tinkerer.

Tinkering / Modular Synths / Happy Accidents

The parametric and chance traditions described above underpin how my software works, but they present very different ergodynamics. In his book *Sonic Writing,* Thor Magnusson proposes *ergodynamics* to signify "the expressive power and depth of an instrument"[23] that cannot be summarized in a list of capabilities or limits but only discovered through use. He compares it to the term "gameplay," used to describe the holistic qualities of interaction, challenge, skill-building, and world-exploration in video games. In terms of ergodynamics, bloom reminds me of another musical practice, modular synthesis.

Modular synthesis is a flexible, open-ended, bottom-up approach to synthesizer design. Contrasting with most commercial synthesizers, in which circuits and functions are pre-designed and their connections fixed, modular synths are open circuits that the user can connect and re-connect any way they want. Synthesis primitives like oscillators, amplifiers, and envelope generators can be combined to make sound, or control each other's inputs. Motion and patterning on the note level is achieved in various ways,

---

[22] Xenakis, *Formalized Music,* 133.

[23] Thor Magnusson, *Sonic Writing: Technologies of Material, Symbolic, and Signal Inscriptions* (London: Bloomsbury Academic, 2019), 11.

including sequencers, random value generators, low-frequency oscillators and delays. All modules speak one language, voltage, so they are fully interoperable and composable. Any output can be used to control any input, which means that each element has a fixed behavior but an unfixed function in the larger system. These systems quickly become complex. In practice this frequently leads to unexpected results, including epiphanic moments known as "happy accidents."

These results seem to arise by chance, but it is not Cage's chance or Xenakis' probability. It is the unpredictability of complex systems, and it is explored as blooms are, not in a pre-determined experiments, but through a bricolage of tinkering, listening, reflecting and refining—an improvisor repeatedly asking "what happens if I…?"

From the synthesist's perspective, a bloom is essentially a sequencer. It stores and generates sequences of four-dimensional values (pitch, velocity, duration, instrument channel), that can be modified and looped. The way I've most often worked with blooms—selecting and refining from randomness—resembles a particular type of stochastic sequencer called by synthesists a Turing machine (not the same as the Turing machine of computer science). In its simplest form, this Turing machine is a single dial. On one end, it generates random values. On the other end, it loops a fixed sequence of values that it has recently generated. In between are blends of the two, states of "slippage" where items of a fixed sequence occasionally change. It is an elegant distillation of the essential axis of all music: unity/variety, or stasis/change, or information/redundancy. In use it resembles the foraging model described above. The performer begins in a state of open exploration, with the dial set to "random." At some point a detail catches their attention, as if catching a scent. Ears perk up. They zoom in on that patch of possibility-space by turning the dial toward the other end, preserving the sequence and looping it for closer study. They then explore the immediate neighborhood by allowing some slippage, in case a nearby variation is even more appealing. With a fixed or nearly-fixed sequence in hand, attention then turns to the

question, “what do I do with this?" Other aspects of the sound are developed: tone and duration, oscillations and arcs of change, counterpoint. The raw sound of pure oscillators is carved with envelopes, tenderized by filters and sweetened by effects, until the chef pronounces it complete. These ergodynamics resemble blooms’ in that they reward bricoleurs, foragers, cooks: they invite into unknown aesthetic territories, suggest iterative, informal experiments, create lots of opportunities for happy accidents, and present technical and conceptual challenges for people who like to read manuals.

Composer as Gardener

Though he wasn’t an influence on my project, I later discovered many points of resonance with the work of Brian Eno. He also used seeds, plants, and cultivation as a metaphor for his compositional process. (Coincidentally, he also has a piece of software called Bloom, a visual synthesizer looping game for a smart phone.) He describes a radical shift in his thinking occurring in the early 1970s, that changed “the idea of the composer from somebody who stood at the top of a process and dictated precisely how it was carried out, to somebody who stood at the bottom of a process who carefully planted rather well-selected seeds, hopefully, and watched them turn into something.”[24] Gardening involves blending wild nature with human control, and the degree of control that should be taken is part of a gardener’s philosophy, ethic, and style. Between the uncultivated wild and the proto-industrial total control of, for example, the gardens of Versailles, Eno describes the middle ground as a “collaboration with the complex and unpredictable forces of nature.”

He drew his inspiration from the contemporary ideas emerging from the sciences, including cybernetics, nonlinear systems theory, and chaos theory, the key insight of which is that some simple systems can generate complex behaviors. A set of rules, tuned *just right,* can have a generative, creative potential that is deep and

[24] Brian Eno, “Composer as Gardiner,” *The Serpentine Gallery Garden Marathon* (October 16, 2011).

unforeseeable. He cites a classic example, John Conway's cellular automaton The Game of Life, in which a few rules about which cells of a grid are on or off generates a dynamic world of shapes that seem to respirate, travel, establish trade, give birth, and wage war.[25]

Eno focused on generative automatons in the genre he is best known for, ambient music. Automatons are particularly well-suited for the long-form, constrained variation of the style. Today, much ambient music is composed by modular synthesists, whose tangles of patch cables can be understood as a form of nonlinear systems design.

Although the understanding and appreciation for these dynamics may be new, their operation in music is not. Traditional counterpoint can be seen as just such a system: complex, dynamic structures arising from a few interacting rules. The rules don't constitute an automaton—the composer must generate and solve and choose—but the compounding constraints prompt choices in ways that can feel, as any composer knows, like a line is writing itself. This, too, is "collaboration with complex and unpredictable forces."

As an abstract and flexible object for storing and transforming musical phrases and spinning them into patterns, the potential for building such systems around bloom objects is immense. Many small demos are included in the Generative Music section of the Code Examples on Github. However, my disenchantment with musical automatons has been described above. I am not so attracted to designing closed algorithms for automatic music as I am to open algorithms that invite me to participate, as forager and cook.

On the issue of control, however, Bloom praxis resonates deeply with Eno's image of creation as cultivation. The frequent chance operations in bloom transformations means that I have only a loose control over my materials. Like a gardener managing water, light and pruning, I have many high-level tools for shaping blooms on a macro

---

[25] Imagining a musical equivalent is a thrill I know firsthand; in college I programmed the Game of Life in Max/MSP and used it to control the pitch and position of sine tones: https://www.youtube.com/watch?v=avK-BmL2KZ4

level, but can't directly control how each leaf grows. Ceding control puts me in a collaborative posture.

In sum, Bloom integrates serial and chance approaches with an interface that is designed to frustrate Romantic authorship and reward curiosity, exploration, and experiment.

Bloom 2.0 Overview

In 2020 I began to revise the core `Bloom` class code, extend it for more uses, design a physical interface around the Manta controller, write sample pieces, and write documentation so that it could be used by others. The current version can be downloaded from GitHub.[26] This version retains the core structure as described above, with the following major additions.

Storage

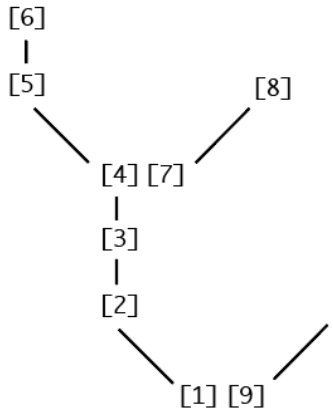
Implementing methods of storage was essential because all bloom transformations are destructive; they permanently change the state of the bloom. I made this choice because it affords a streamlined syntax, but it introduces a bias towards musical phrases that constantly evolve but never return. Musical recurrence requires storage, and Bloom now has three approaches:

The first is the Global Log. This is a collection of blooms that are saved to disk with the method `.log`, so they persist between sessions. The intended use is to build, over time, a catalog of blooms chosen by ear. Instead of generating truly random blooms, one can randomly draw from the list for curated results.

[26] http://github.com/elliotcole/bloom

The second is a simple 1 stage undo/redo function, with `.save` and `.restore`. The third is a 'stack' stored in each bloom. A stack is a list of states saved with the method `.push`. The most recently pushed state can be restored with `.pop`. This is useful for tree-like paths, because `.pop` returns you to branch from a previous point. For example:

```
[1] b.seed.play.push       // branch point 1
[2] b.stutter.play         // transformation
[3] b.negHarmony.play      // transformation
[4] b.push                 // branch point 2
[5] b.thin.play            // transformation
[6] b.chooseScale.thicken.play // transformation
[7] b.pop.play             // return to point 2
[8] b.shuffle.shear        // transformation
[9] b.pop.play             // return to point 1
```



Diatonicism

Blooms can now move easily in diatonic space. Tools include `.applyScale`, for forcing the notes of a bloom into a given scale, and `.dTranspose` for movement within it. `.chooseScale` searches Supercollider's `Scale.directory` for scales that nearest match the current notes of a bloom. `.slantScale` searches for scales that mostly-but-not-too-closely fit the current notes, to shift to nearby harmonic neighborhoods. `.simplifyScale` drops one pitch from the current scale and adjusts the notes to match, allowing you to progressively clarify complex harmonies. `.fixedScale` is a toggle that locks a bloom into a scale, no matter how it is transformed.

New harmonic transformations have been added: `.negHarmony` reflects pitches across the midpoint between the dominant and the tonic of a key. `.addSharp` and `.addFlat` move the scale through the circle of fifths.

Quantization

A rhythmic grid can now be enforced with `.quantize`. Two grids with different denominators can be applied, for example `.quantize(3,5)` divides the beat into thirds and fifths and snaps notes to whichever is closer. `.fixedGrid` is a toggle that locks all rhythms the grid, no matter how it is transformed. Control of total phrase duration is now possible with `.trimToDur`, `.scaleToDur` and `.fixedDur`.

All playing is now done to a clock that can easily be linked with Ableton Live, allowing for synchronized integration with loop-based music.

Chords

Chords within blooms are now possible, using nested arrays. For example, the second item of `[60, [61,65,68,72], 55]` is a Db maj7 chord. The methods `.chord`, `.chords`, `.chordsShorten`, `.chordsRand`, and `.chordsRandShorten` all group the notes of a bloom into chords in different ways. `.flattenChords` reverses these groupings.

Chords can be spaced, roughly, like 'good' piano voicings with `.spaceChords`, and the chords of a bloom can be connected efficiently with `.efficientChordVoicings`.

Phrases

Blooms can be conveniently concatenated into phrases with ++. This is the only method that is non-destructive; `a ++ b` leaves both blooms unchanged, for easy recombination.

Looping / Pulsing

Turning blooms into streams is now easier with the methods `.loop` and `.pulse`. `.loop` loops each list, notes, timeIntervals, velocities, channels, a stream of events that changes as the bloom changes. It creates `Pdef` and `Pdefn` objects, which are SuperCollider pattern definitions that allow for on-the-fly changes. `.pulse` plays the bloom at fixed intervals. It can be passed a function to be evaluated each cycle, enabling automatic transformations with each iteration.

Manta Interface

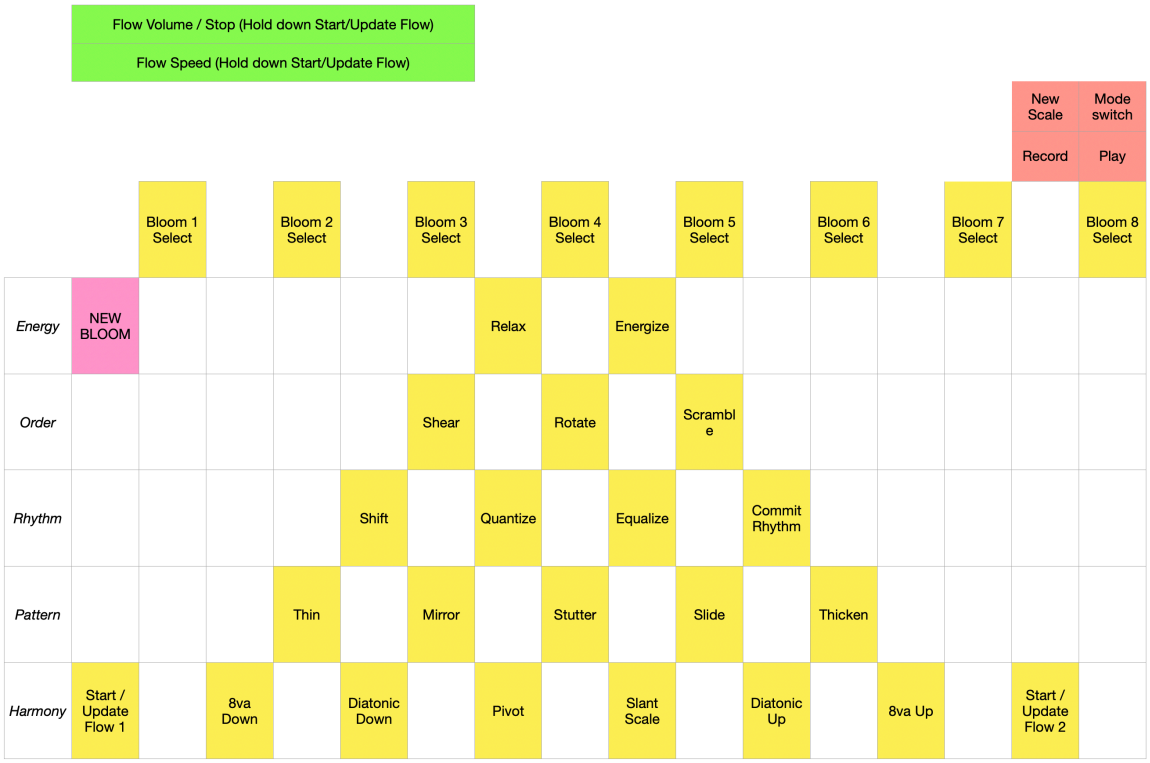
The Manta Interface program turns my colleague Jeff Snyder's Manta controller into an interface for recording and performing with blooms. A full diagram of functions is included as Appendix B. In the current version, there are two modes of interaction. In both modes, there is a single bloom 'in focus' at a time.

The first is Keyboard Mode. Like a keyboard, the 48 pads play one note each. The notes are the notes of `bloom.scale`. This is a complete scale if one has been applied, or, if not, the intrinsic scale of the bloom. These notes are laid out in two panels of 4 columns each, with the left panel transposed below the right. This allows two-handed playing to approximate a piano, with the left hand playing in a lower register. One slider controls the octave, and the other lets the user set a fixed velocity for each note, or, in its leftmost position, makes the velocities variable by the expression dimension of the pads. (Manta pads do not register key velocity in the usual way, the speed of attack of the finger, but rather by the surface area of the finger in contact with the pad. Loud sounds are articulated by the pads of the fingers, soft sounds by the smaller tips.)

Of the four buttons, one switches between modes. Another chooses a neighbor scale, using the `.slantScale` method described above, allowing for smooth shifts in harmonic space around common tones. The remaining two control a bloom recorder:

one starts and stops recording, the other plays back the buffer. This allows the performer to improvise in a scale and record a gesture. This bloom is then the bloom in focus, which can be transformed and saved in Bloom Mode.

In Bloom Mode, a lit pyramid of pads each play the current bloom with a different transformation applied. These transformations are organized into categories by row: energy (velocity+speed), order, rhythm, pattern and harmony. The top row of pads each store and recall a different bloom. Special pads in the bottom left and right start the bloom looping, creating a continuous flow of tones that can be transformed on the fly, with adjustable rate controlled by the slider. The remaining unlit pads each play notes from the current blooms scale, allowing convenient improvisation and intervention.



Manta layout, Bloom mode.

Model Code

I have written code for a collection of example pieces to give new users models to explore. These fall into three categories that I see as the obvious paradigms for use.

Live-coding

A unique feature of SuperCollider is that code can be executed piecemeal, in single lines or blocks. This is distinct from many programming environments, where entire programs must be compiled at once. This allows for a very flexible, hands-on approach to coding, where individual statements can be tested as you go.

When working with blooms, this allows an improvisatory, free-form, ad hoc style of work that resembles live-coding languages like ChucK and TidalCycles. While it could be used in a live-coding performance, it is more obviously useful as a flexible sketch pad for developing compositional ideas. The most direct workflow is: sketch in SuperCollider, record MIDI output in a DAW like Logic or Ableton. From there, it can be edited and converted to notation in Finale, Sibelius or Dorico.

Integrating blooms with SuperCollider's native `Pattern` library offers endless possibilities for transformation and realtime control, and I provide examples for this as well.

Generative Music

Using the SuperCollider `Task` object, sequences of commands can be scheduled and looped for potentially endless iteration. Conditionals like `if`, `while`, and `case` open the possibility of context-dependent evolution. These are the ingredients of autonomous generative music compositions, and I include several examples to help others get started.

Interactive Environments

The last examples show how a MIDI keyboard can be used as an interface for a bloom-based play environment.

Conclusion / “What we need is a computer that isn't labor saving but which increases the work for us to do”

Let’s return to Cage’s provocation. How did he come to this thought, and what advice may he be giving? It comes from the first handwritten draft of *Diary: Audience 1966*. His process for writing it is a perfect example of what I described earlier as a kind of foraging algorithm. He describes the diary’s composition this way:

> This text was written on the highways while driving from an audience in Rochester, New York, to one in Philadelphia. Following the writing plan I had used for *Diary: Emma Lake,* I formulated in my mind while driving a statement having a given number of words. When it had jelled and I could repeat it, I drew up somewhere along the road, wrote it down, and then drove on. When I arrived in Philadelphia, the text was finished.[27]

His procedural experiments like *Music of Changes* were designed to remove his own subjectivity, but in this experiment his subjectivity—his mental chatter on the road, and the experience of “jelling”—is the algorithm. And, as Christopher Shultis has discovered by comparing drafts,[28] he observed the procedure loosely, allowing himself to intervene and revise as he saw fit, as in my image of a forager and cook.

As I survey my years down this bloom rabbit-hole, his recommendation that we need “a computer that isn’t labor saving but which increases the work for us to do” strikes at the irony of my project. The initial goal was to build a labor-saving device, but the labor that went into it far exceeded the labor it saved. What was Cage getting at?

---

[27] Cage, *A Year from Monday,* 50.

[28] Shultis, ibid.

What benefit can come from *increasing* the work for us to do? I can begin to answer that question from my own experience.

First, creating a computer program requires us to think clearly. To translate knowledge and ideas into code requires that they be disassembled, cleaned, and laid out in a careful and comprehensive way. Rarely are we required to do this deep work on our ideas, especially in the arts where our work is frequently guided by hunches and instincts.

Second, it requires us to think abstractly. This is the leap, in Xenakis' terms, from "forms" to "schemes." Finding a pleasing harmonic shift is a project of forms; understanding what is pleasing about it, and using that to generate other possibilities, requires abstraction. Programming continually challenges us to develop that skill. This deepens our understanding of everything we do, and increases the creative potential of that understanding. It can also help us see tools, strategies, and assumptions at play when we create in unstructured and intuitive ways.

Third, it reveals how our ideas, our instruments, and our worldview are connected. Designing a program or instrument encapsulates a way of thinking, as both a representative model of knowledge and as functional, working knowledge.[29] An instrument is also a pathway to new knowledge. Writing about scientific instruments, Thomas Hankin and Robert Silverman assert that "they do not merely follow theory, they often determine theory, because instruments determine what is possible, and what is possible determines to a large extent what can be thought."[30] Using any tool shapes our perspective on the task and angle of approach. This is always both empowering and limiting, as in adage "when all you have is a hammer, everything looks like a nail." It is

---

[29] Davis Baird, in *Thing Knowledge: A Philosophy of Scientific Instruments* (Berkeley, CA: University of California Press, 2004), 131.

[30] Thomas L. Hankins and Robert J. Silverman, *Instruments and the Imagination* (Princeton, NJ: Princeton University Press, 2014), 5.

frequently easier to see the new power a tool gives us than to see ways it limits us. In our techno-centric age, this may be our Achilles' heel.

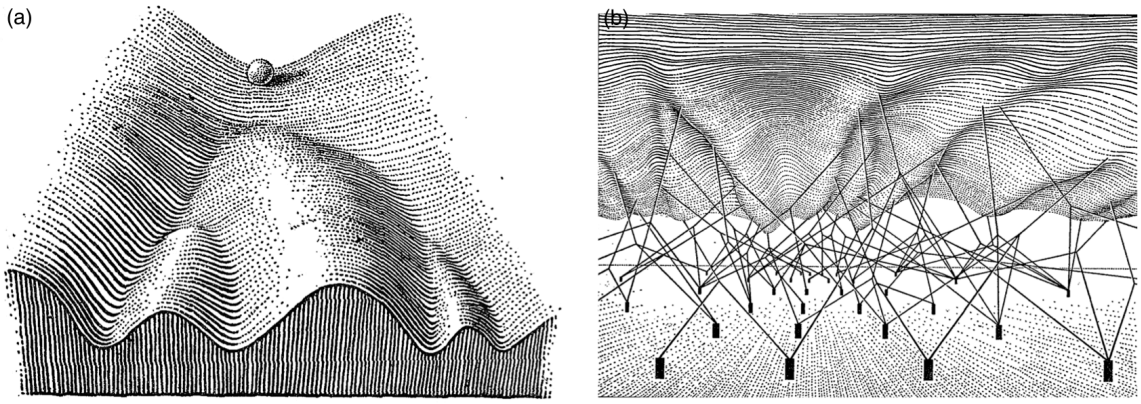
When we create our own technology, we can watch this dynamic play out in real time. We can watch our assumptions become embedded and then seem to disappear as we build around them, and habituate to them. We decide for ourselves what is important and what isn't, which features of reality we want to amplify and which to reduce, to use Don Ihde's useful distillation of what all technology does.[31] Then we deal with the consequences of those choices. We watch our theories, priorities, and biases become design decisions, features, limitations, and flaws. We see how our ideologies help or hurt our intentions. This helps us have a more critical perspective on all the technologies we use. If we can learn that features are always trade-offs, that doors are closed when doors are opened, and that technology is never neutral, we will be in a better position to evaluate, utilize, or resist the technologies that are sold to us.

Fourth, and the heart of my case, is that programming is a powerful way to create and explore new musical languages. I hope that I have demonstrated this by example, but now let's ask: what is a musical language? How do we create one, and how are computers particularly useful in that project? First, the term "musical language" is not an assertion that music is itself a language. Instead it reflects our experience that music can be coherent like language can be coherent. A language can produce an infinite variety of statements that all make sense because they share a consistent grammar, syntax, and vocabulary. Music also seems to "make sense" when it reveals consistent—redundant—underlying structures. Established musical languages benefit from large bodies of literature that reinforce these structures, forming networks of embodied meaning, to return to Meyer's phrase. Without this advantage, there is greater burden on new musical languages to reinforce their underlying structures internally, rigorously and systemically.

---

[31] Don Ihde, *Technics and Praxis* (Dordrecht, Holland: D. Reidel, 1979), 56.

The composer Thor Magnusson writes that a "'new systematicity,' or a shift from composing a work to inventing a system, is characteristic of twenty-first century music."[32] System here transcends and includes the ideas of tools, instruments, and theories; it is the total environment that constrains and stimulates the creative act. Magnusson describes how systems "'contain' the music through their affordances and constraints." This metaphor of containment can be refined. More precisely, the music is not located in the system; the system is ontologically prior. Instead I visualize these containers as forms—basins, channels, contours—in the space of possibility, as in biologist Conrad Waddington's illustration of an epigenetic landscape:[33]



This visual metaphor helps us imagine how genes (the pegs in the ground in the second image) interact with each other (the cables) to shape possible paths of development for a cell (the ball), without dictating its precise destiny. Mapping this metaphor to computer music systems, the pegs are the code and the ball will be music, once it has rolled into being. Between the two is a network of interactions that shape a generative topography of possibility.

---

[32] Thor Magnusson, "Designing the Threnoscope or, How I Wrote One of My Pieces," *Sound Work*, 2021, pp. 219-230, https://doi.org/10.2307/j.ctv1ccbg96.14.

[33] C. H. Waddington, *The Strategy of Genes* (London: Allen & Unwin, 1957).

A musical language can also be imagined as a generative topography of possibility. Some forms are more likely to emerge than others. Some are impossible. Certain pathways, once entered, converge on likely destinies. Coherent musical languages emerge from self-consistent topographies. Music generated within one is more coherent because all of its elements are shaped by common tools, and common rules. Some rules may be explicitly defined but many emerge from the interactions of deeper forces. This is where computers are an enormous help. We can easily re-arrange our pegs —our ideas—and the computer can work out their implications and interactions—the cabling—far beyond our capacity, forming a consistent, stable possibility-space for us to explore.

In most of history, a composer's topography of possibility was shaped by many forces over which the composer had little control: the tools, taste and theory of the times, the capabilities of performers, the mechanics of their instruments. In these domains, composers largely traversed a shared and inherited landscape. What's exciting about creating music today is the power we have to shape our own.

Appendix A: Bloom API Reference

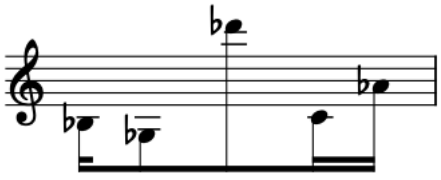
Data Structure

A Bloom is *a musical phrase that can be transformed*. The phrase is represented by 4 arrays: `notes, velocities, timeIntervals and chans`.

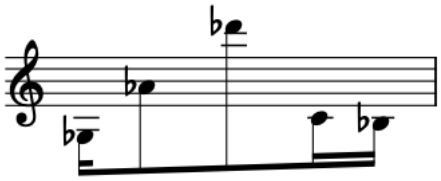
- `Notes` and `velocities` are MIDI values (0-127) specifying pitch and volume,
- `timeIntervals` are measured in beats of the current clock, and
- `chans` are MIDI output channels for each note, which can be used as an instrument, articulation, or timbral parameter

Introduction:

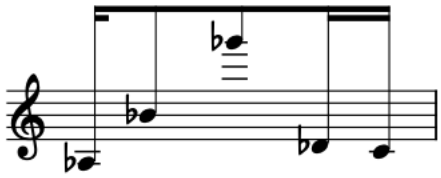
```
// initialize MIDI
(
MIDIClient.init;
MIDIIn.connectAll;
Bloom.midiOut_(MIDIOut.newByName("IAC Driver", "Bus 1"));
Bloom.midiIn_(MIDIIn.findPort(“Your Keyboard”,"Your Keyboard"));
)

b = Bloom.new.seed; // create a new bloom and fill it with
random* notes, velocities, and timeIntervals
b.verbose_(true); // print contents whenever it is played

b.play; // sends midi to
Bloom.midiOut. A DAW or synth is
necessary to listen, or a
SuperCollider SynthDef can be
employed with .synthPlay. Note:
rhythmic notation is quantized for
simplicity
```



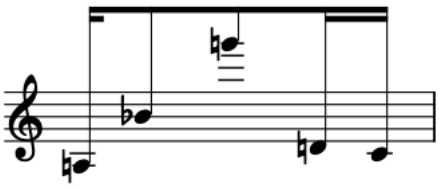
```
b.scramble; // randomize note order
while preserving shape
```



```
b.dTranspose(1).play // transpose up a
diatonic step. If no scale has been
applied, the scale is intrinsic scale
of the collection
```

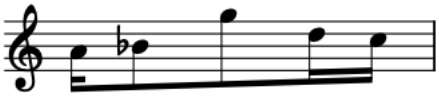


```
b.applyScale(Scale.dorian).play; //
scales default to a C root; this can
be changed with .root()
```

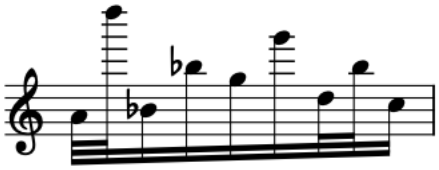


```
b.push // saves current state into memory stack
```

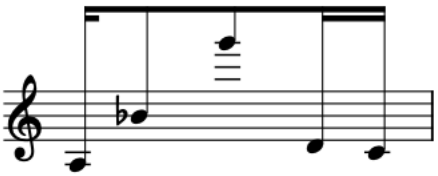
```
b.compass(67,79) // compress within
midi note range, preserving pitch
classes
```



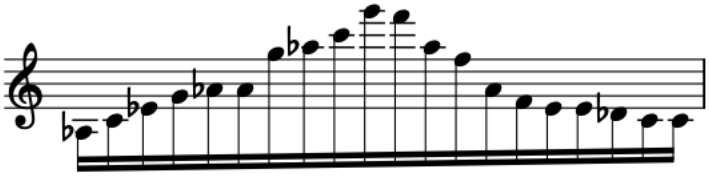
```
b.thicken // increase the density of
the bloom without changing its tonal
character
```



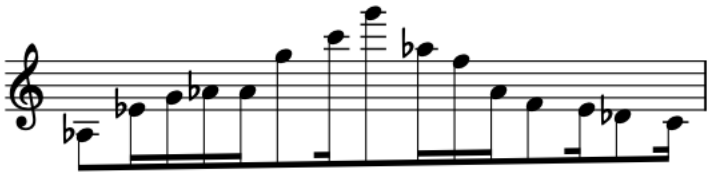
```
b.pop // restore the last bloom that
was .pushed
```



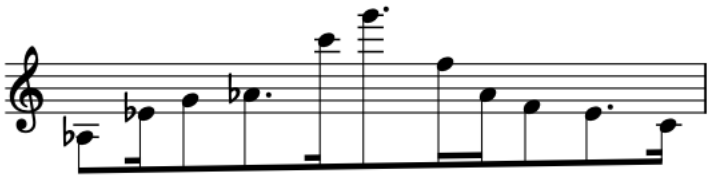
```
b.applyScale(Scale.phrygian).curvesD(20)
 // shift to Phrygian
mode, and then create a
20 note bloom that
'connects the dots' of
the original bloom
```



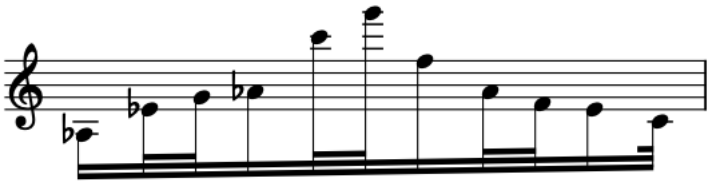
b.thin // drop some notes at random



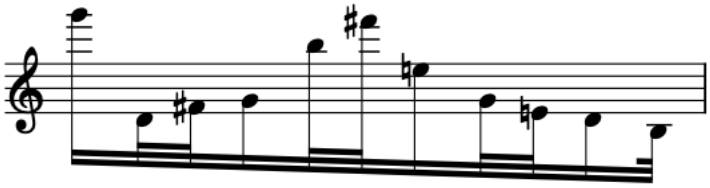
b.thin // drop some more. Note how the overall duration stays the same.



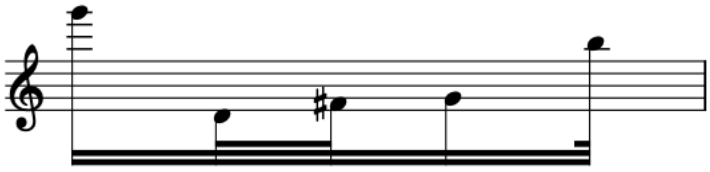
b.wrapTime([1/4, 1/8, 1/8]) // apply a rhythmic pattern (durations are fractions of a beat)



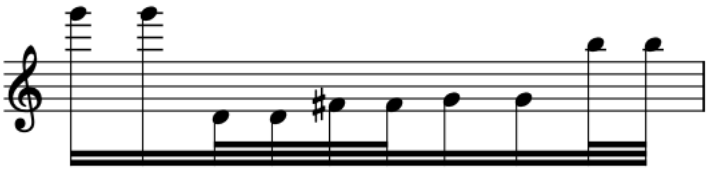
b.pivot // harmonic transformation: invert chord, then transpose down so the new highest note is the same as the old highest note. Preserves interval vector while moving to new harmonic area.



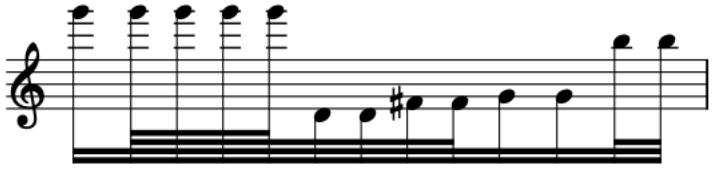
b.trimTo(5) // keep only first 5 notes



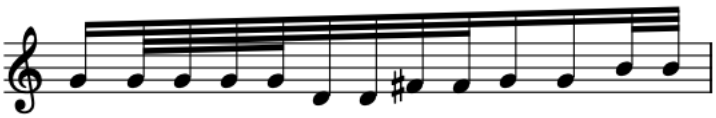
b.stutter // double each note and duration



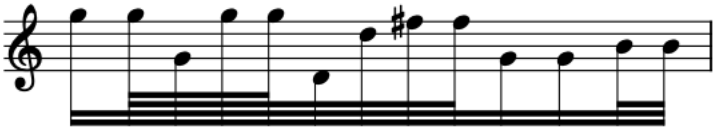
b.ratchet(4, 1) // repeat second note (index 1) 4 times within the original duration



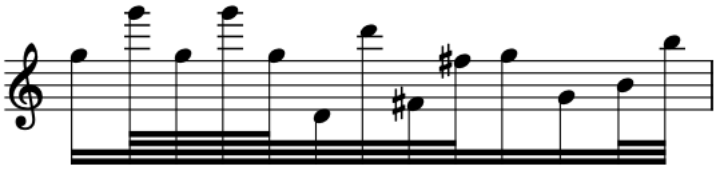
b.compress // within an octave



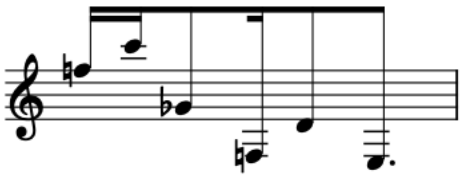
b.shear // expand range and randomize contour by moving some notes up or down by an octave; lowest note remains the lowest note



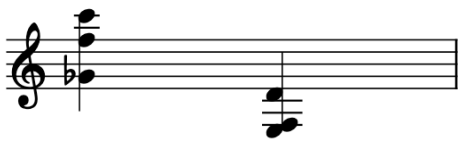
b.shear



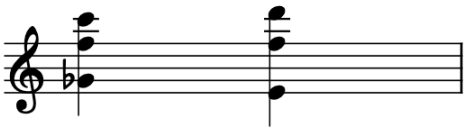
b.seed // new random bloom



b.chords(3) // group into 3 note chords



b.efficientChordVoicings // re-arrange chords for smooth voice-leading



Class Methods

`Bloom.new(notes, vels, times, chans)`

Create a new Bloom. Its contents can be specified at instantiation, but more often it is filled with `.seed`. Lists do not need to be the same length: `velocities`, `timeIntervals` and `chans` will wrap to the length of `notes`.

Examples:

```
b = Bloom.new; // default Bloom
b.report;
b = Bloom.new([46,28,43,24,67],[65,23,45,11],[1],[0]);
b.report
b.play
```

`Bloom.clock`
`Bloom.clock = value`

Clock is by default `TempoClock.default`, but this can be set to another clock or to a `LinkClock` to sync with Ableton, as in the following example, which also sets a 1 bar quant with an offset to compensate for latency. All play methods (`.play, .loop, .pulse`) use this `clock` and the instance variable `quant`.

NOTE: Command-period will kill your LinkClock's connection to Ableton. Rerun the following lines to re-establish it. It is useful to include them and your MIDI initialization in `startup.scd`, for easy resetting with Recompile Class Library (command-L).

```
l = LinkClock(1).latency_(Server.default.latency);
Bloom.clock_(l).quant_([4,0,0]);
```

Instance Methods

storage

All bloom transformations change the internal state of the bloom destructively, so storage is important for creating musical return. There are three approaches to storage built into each bloom.

- **the log**: a globally-available library that is written to disk to persist between sessions and recompiles. Suggested use: when you come across something

really beautiful, `.log` it. Instead of using `.seed` to get random blooms, draw randomly `.fromLog` for curated results. Multiple logs can be kept. Items in the log are located by number.

- **save and restore**: a single 'undo' for returning to a previous state.
- **the stack**: an ordered sequence of states saved by `.push` and retrieved by `.pop`. Useful for exploring forking paths of change.

`.log(index)`

Save this bloom for future use (in `Archive.global`). Logged blooms persist between sessions. If no `index` is specified, the next available will be used.

`.fromLog(index)`

Restore a bloom from the log by number. If no number is specified, one will be chosen at random.

`.clearLog`

Resets the log. Careful!

`.logName`

Multiple logs can be maintained by name. Set the instance variable `logName` to a `\symbol`.

Examples:

```
b.logName = \blooms;
b.seed.applyScale.play;
b.log;
b.seed.play;
b.fromLog.play;
b.logName = \diads;
b.seed(2).play.log;
b.seed(2).play.log;
b.fromLog.play;
b.logName = \blooms;
b.fromLog.play;
b.logName = \diads; b.clearLog; b.logName = \blooms;
```

`.logAsArray`

Return the log as an Array. Modifications to this array will not affect the log. Replace the log with `Archive.global.put(logName, array);`

`.save`

save a copy of the current state of the bloom. Saved bloom is accessable via instance variable `saved`.

Examples:

```
b.seed.chooseScale.play;
b.save;
b.stutter.play;
b.saved.report;
b.restore.play;
b.meanInvert.play;
b.restore.play;
```

`.restore`

Restore the saved bloom.

`.push`

Save the current state to the stack

`.pop`

Return to the last saved state in the stack, and remove it from the stack.

`.popAny`

Return to a random previous state.

`.stack`

Return the stack as an Array.

random variation

`.seed(numNotes, low, hi)`

New random notes / velocities / timeIntervals in prototypical Bloom shape: 4-10 notes, pitches between `low` and `hi`, velocities between 40-100 sorted loud to soft, timeIntervals between 0.1-0.5. All randomization uses `exprand` to privilege lower values.

`.newNotes`

New random notes (timeIntervals and velocities stay the same)

`.newShape`

New random timeIntervals and velocities (notes stay the same)

`.shiftNotes(probability: 0.3)`

Adjust some notes upward or downward one semitone

`.shiftNotesD(probability: 0.3)`

Adjust some notes upward or downward one diatonic step, using the current scal

`.shiftShape(spread: 0.4)`

Adjust timeIntervals and velocities by random percentages within spread

`.shift`

Convenience method: shifts notes and shape the default amount.

`.shiftD`

Convenience method: shifts notes diatonically and shape the default amount.

`.compass(lo: 60, hi: 90)`

Restricts the pitches to a range. notes are transposed by octave; pitch classes are preserved

`.scramble`

Randomize order of notes; shape is maintained; notes in chords stay together

`.deepScramble`

randomize order of notes; shape is maintained; notes can leave their chords

`.shuffle`

shuffle all dimensions like a perfect card shuffle (second half interleaved with first half). Notes in chords stay together.

`.deepShuffle`

shuffle all dimensions; notes leave their chords

expanding / contracting

`.remove(index: 0)`

remove note at index method:index

`.addOne(noteOrList)`

appends one new note, random if nil method:midinote

`.addOneInScale`

appends one new note with the bloom's scale

`.dropLast`

drop the last note of the bloom

`.curvesExtend(newSize: 10)`

"connects the dots" of the bloom (notes, timeIntervals and velocities) with a chromatic line of the desired length. Channel array is wrapped.

`.curvesD(newSize: 10)`

"connects the dots" of the bloom (notes, timeIntervals and velocities) with a diatonic line of the desired length. Uses the existing diatonic space or uses `.chooseScale` to establish one. Channel array is wrapped

`.thicken(percentNew: 0.5)`

Creates a new Bloom and interlace its notes in this bloom. If no scale has been applied, it draws from the pitches present (but randomizes octave). If a scale has been applied, it draws from the scale. A new note is added between each note, with a lowered velocity. Because it uses `.lace`, the total duration doesn't change.

`.thickenLengthen(percentNew: 0.5)`

Same as thicken but uses .laceLengthen, so the total duration is lengthened

`.thin(probability: 0.3)`

Remove some notes but preserve the shape (a removed note leaves a 'hole' in the timeInterval).

`.thinShorten(probability: 0.3)`

Remove some notes along with their timeIntervals.

`.gap(probability: 0.3)`

Silence some notes (velocity to 0).

`.unGap`

Restore all silent notes to the mean velocity of the bloom.

`.removeDoubles`

Remove any doubled notes (adjacent duplicates)

`.trimTo(length)`

Trim the bloom to a certain number of items

`.trimToDur``(dur: 4)`

Trim the bloom to an exact duration

.scaleToDur(dur: 4)
Proportionally compress the bloom to an exact duration

rotations

.rotate(n: 1)
Rotate all lists by n positions.

.rotateNotes(n: 1)
Rotate just the notes by n.

.rotateVelocities(n: 1)
Rotate velocity list by n

.rotateTime(n: 1)
Rotate timeInterval list by n

shaping

.slower(multiplier: 1.2)
Scales all timeIntervals by multiplier

.faster(multiplier: 1.2)

.softer(multiplier: 1.2)
Scales all velocities by multiplier

.louder(multiplier: 1.2)

.fan
Sort timeIntervals fast to slow.

.avgTime
Equally space all notes within the total duration.

.addComma(dur: 1)
Add dur to the final timeInterval; useful in looping patterns.

.decrescendo

Sort velocities in descending order.

.crescendo

Sort velocities in ascending order.

fission

.curdle(probability: 0.2)

divide the Bloom into a few Blooms at random

.split(splitAt)

divide the Bloom into two Blooms at a certain index

fusion

.add(newNotesOrBloom: 60, newVels: 60, newTimes: 0.25, newChans: 0)

append entries to any list, or append an entire bloom.

++(aBloom)

Concatenate Blooms. **NOTE:** this notation returns a new bloom, and does not modify the receiver.

Examples:

```
b.seed;
c.seed;
z = b ++ c;
b.play; // unchanged
c.play; // unchanged
```

.lace(bloom)

Interlace the notes of one bloom into the shape of another. Total duration doesn't change; each new note is slotted between the originals,

.laceLengthen(bloom)

Interlace the notes of one bloom into the shape of another. Total duration does change,

`.cast(bloom)`

Apply the shape (timing, velocities and registral contour) of another bloom to this one. The sequence of pitch classes is preserved; each note is moved to the octave nearest the applied note. Blooms of different lengths will wrap.

patterning

`.stutter(repetitions: 2)`

Repeat each note of a bloom.

`.sputter(probability: 0.3)`

Repeat some notes of a bloom, determined by chance. notes, timeIntervals and velocities are linked.

`.sputterAll(probability: 0.25)`

Sputter notes, timeIntervals and velocities separately.

`.ratchet(repetitions: 2, index)`

Repeat a note `repetitions` number of times, subdividing the duration eg. a quarter note becomes two identical eighth notes.

`.mirror`

Append the bloom's reverse; a palindrome results.

`.pyramid(patternType: 1)`

Apply one of Supercollider's pyramid counting algorithms. See Array.pyramid.

`.slide(windowLength: 3)`

Traverse the bloom in repeated subsequences of length windowLength.
See `Array.slide`

`.quantize(grid: 4, strength: 1)`

Quantize the durations to fractions of the beat. Two grids can be used to allow for polyrhythmic relationships. Grids are defined in fractions of a beat. Beat duration is set via `TempoClock.default`.

Examples:

```
b.quantize(4) // quantize to quarters of a beat (16th
notes)
b.quantize([5,3]) // quantize to a 5 against 3 polyrhythm
b.quantize(4, 0.5) // quantize to quarter-beats but snap
only halfway to ideal position
```

.wrapTime(list)

Apply a rhythmic pattern across the Bloom by wrapping a list of timeIntervals.

.wrapVel(list)

Apply an accent pattern across the Bloom by wrapping a list of velocities.

.wrapChan(list)

Wrap a list across the midi channels.

Examples:

```
b.wrapChan([0,1,2]) // note 1 goes out channel 0, 2 out 1,
etc. argument list a list of channels
```

pitch operations

.compass(lo: 60, hi: 90)

Compress all pitches within a range.

.compress

Compress all pitches within an octave.

.shear

Randomize octaves of pitches; lowest pitch is held.

.invertMean

Set-theory inversion: inversion around mean. Bloom is inverted as a whole, not each chord separately.

.invert(n: 1)

Chord inversion: move lowest note to the top. Bloom is inverted as a whole, not each chord separately.

.transpose(semitones: 0)

Chromatic transposition.

.dTranspose(steps: 0)

Diatonic transposition, using the applied scale if a scale has been applied, or using the pitches present in the bloom as a scale if not.

`.resolve(percentToResolve: 0.2)`

Analyze the notes for their intervals, choose a few (percentToResolve) at random, and resolve them to more consonant intervals. As the resolutions of each interval produces new intervals with other pitches in the bloom, the results are unpredictable but produces a winding path toward harmonic simplification.

interval	resolves to
M7	M6
P4	M3
TT	P5
m7	M6
M2	m3
m2	U
m6	P5

`.pivot(i: 1)`

A pivot is a two step pitch operation consisting of

- chord inversion (lowest pitch moved to highest pitch)
- transposition down so that the highest pitch of the new chord is the same as the highest pitch of the old chord

A pivot holds one note (top note) in place and shifts all others around it. The interval vectors of the two chords are the same, so they have very similar qualities while in distant keys. The aural effect is a pleasing paradox of similarity and difference.

`.pivotBass``(i: 1)`

Pivot around the bass; bass stays the same, other notes move.

`.pivotLoudest``(i: 1)`

Pivot around the loudest note

`.negHarmony``(root)`

Negative harmonic relationship: a reflection across the midpoint between the root and fifth of a key.

`.flatten(howManyToFlatten: 1)`

Choose notes from the scale and lower them a semitone.

diatonicism

`.applyScale(input, root)`

Fits notes into a scale. Input can be a Bloom, a List or Array of semitones, a Scale, or will choose a random scale if nil. Root is a pitch class 0-11 or a note name as a String.

`.chooseScale`

searches Scale.directory for the scales that best contain the notes of the bloom. If there are multiple scales that match it equally well, it chooses one at random. The scale is applied to the Bloom and stored for diatonic transpositions.

`.slantScale`

searches Scale.directory for neighbor scales -- scales that contain the notes of the bloom nearly but not exactly. Choose one at random and apply it to the bloom.

`.root(root)`

Sets the root of the scale and adjusts the notes to fit. Root can be a pitch class 0-11 or a note name as a String.

`.addSharp`

adds one sharp in the circle of 5ths (sets the root up a fifth)

`.addFlat`

adds one flat in the circle of 5ths (sets the root down a fifth)

`.scale`

return the current diatonic space. If a scale has been applied, it will be that; if not, it will be a scale made up of the pitches present in the bloom.

`.simplifyScale`

removes one note from the scale and adjusts notes to fit

chords

`.chord`

Group the whole bloom into a chord.

`.chords(notesPerChord: 3)`

Group notes into chords of a given size. The timeIntervals of each note in a chord are summed, so the total duration doesn't change.

`.chordsShorten(notesPerChord: 3)`

Group notes into chords of a given size. The timeIntervals of each note in a chord are not summed, so chords play in the original, faster rhythm.

.chordsRand(probability: 0.3333)

Group some notes into chords; probability is the chance that a note will begin a new chord. Lower numbers producer larger chords. The timeIntervals of each note in a chord are summed, so the total duration doesn't change.

.chordsRandShorten(probability: 0.3333)

group notes into chords by probability. The timeIntervals of each note in a chord are not summed, so chords play in the initial rhythm.

.chordsByInterval(interval: 3)

Group notes into chords if they make a stack of a given interval. If no scale has been applied, `interval` is chromatic, otherwise it will be diatonic (a value of 3, for example, may return minor or major 3rds in a major scale). Notes are sorted.

.harmonizeEfficiently(chordTones)

Adds a diatonic chord below each note of the bloom based on the applied scale. Progression is optimized for efficient voice leading. [1,3,5] will create triads, like the 1 3 and 5 of a chord.

.flattenChords

Undo chords, restore a sequence of single notes.

.pivotChords

Pivot each chord around its highest note separately.

.spaceChords

Attempt pianistic spacing of each chord separately.

.efficientChordVoicings

All chords will have notes rearranged by octave to move as efficiently as possible from one to the next. Singleton notes are skipped.

playing

.play(channels, trace: false)

Play the Bloom with MIDI. Specifying `channels` overrides the existing channel array `b.chans`. Use `trace` for debugging.

.playWait(trace: false)

Play the Bloom and then `.wait` its duration. Intended for use in `Tasks`, so the next statement of a `Task` is not run until the Bloom is complete.

`.quant = value`

The `Quant` value for `.play` and for any spawned `.loop` and `.pulse`.

`.asPbind`

Return a `Pbind` of the Bloom with a full complement of `Pdefns` for intervention. Each `Pdefn` is created with a unique name; check post window for yours.

Pdefn functions and defaults

Pdefn	what it does	default behavior
\i	“playhead position": index of next note	Pseries(0,1,inf) - forward through `notes` (looping)
\dur	event dur	forward through timeIntervals (looping) * \durScale
\durScale	duration multiplier	1
\vel	sets velocity	forward through velocities (looping)
\chan	sets midi output channel	forward through chans (looping)

`.asPdef(name)`

Return a `Pdef` of the `Pbind`. You must provide a name, which will be the prefix for all created `Pdefns` in this format: `name_i`, `name_dur`, etc.

`.loop(name)`

Loop the bloom. Returns a Pdef (see above).

`.pulse(dur: 8, action)`

Loop the bloom at regular intervals, running `action` at each iteration. Returns a `BloomPulsar`, whose rate is set with `.rate` for on-the-fly change. `Action` can be a `Function` or a `Routine`.

Examples

```
b.fixedDur_(4);
~pulsar = b.seed.pulse(4,
   {b.negHarmony(11.rand).newShape.softer});
~pulsar.rate_(6);
~pulsar.stop;
```

`.setSustain(value: 1)`

Sets the duration each note sustains.

`.setLegato(value: 1)`

turns off sustain mode, note durations are determined by `timeInterval * value`

recording and importing

`.record(metronome: false)`

records MIDI input and saves it into the Bloom. Recording begins with the first note played. The metronome plays at `TempoClock.default.tempo` and must be terminated with `.rstop`

Arguments:

metronome	if `true`, it will use the \default synth, or you can specify your own synth

`.rstop`

stops recoring and saves the contents in the Bloom

`.fromMIDI(simpleMIDIFile, whichTrack: 0)`

takes a `SimpleMIDIFile` (extension in quart wslib) and imports a given track of MIDI into the Bloom. argument:simpleMIDIfile argument:whichTrack

fixed modes

`.fixedScale` `= value`

`true` or `false`. When `true`, the applied `Scale` is enforced every time the bloom is played.

`.fixedDur = value`

Enforces a total duration for the bloom at the time of play.
Uses `.trimToDur` or `.scaleToDur` depending on `fixedDurMode`.

Example

```
b.fixedDur_(4);
z = b.pulse(4);
b.newShape;
b.mirror;
b.fixedDurMode_(\trim);
```

```
b.fixedDurMode_(\scale); // try each mode with the above
transformations
z.stop;
```

.fixedDurMode = value

Specify either \trim or \scale. \trim chops notes beyond the fixed duration, \scale compresses all notes to fixed duration.

.fixedGrid = value

Enforces a rhythmic quantization for the bloom at the time of play. Pass it a single division value or list of two values as you would .quantize.

reporting and conversion

.at(index: 0)

Given an index or array of indices, return the notes as playable Events.

.report

Print the note, velocity, timeInterval, and channel arrays of the bloom.

.rp

Print the arrays in easy-to-copy form.

.asScale(notesPerOctave: 12)

Return a Scale object containing the pitch classes of the current notes.

.asPChist

Return a pitch-class histogram.

.asPCs

Return the notes as pitch classes.

.asMIDIhist

Return a histogram of all MIDI notes 0-127.

.duration

Return total duration.

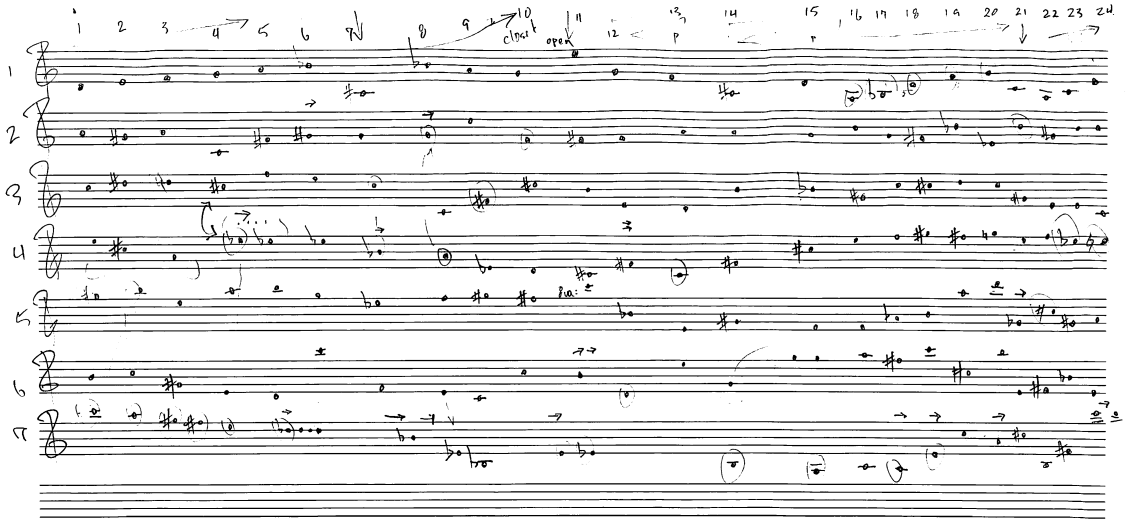
.absTime

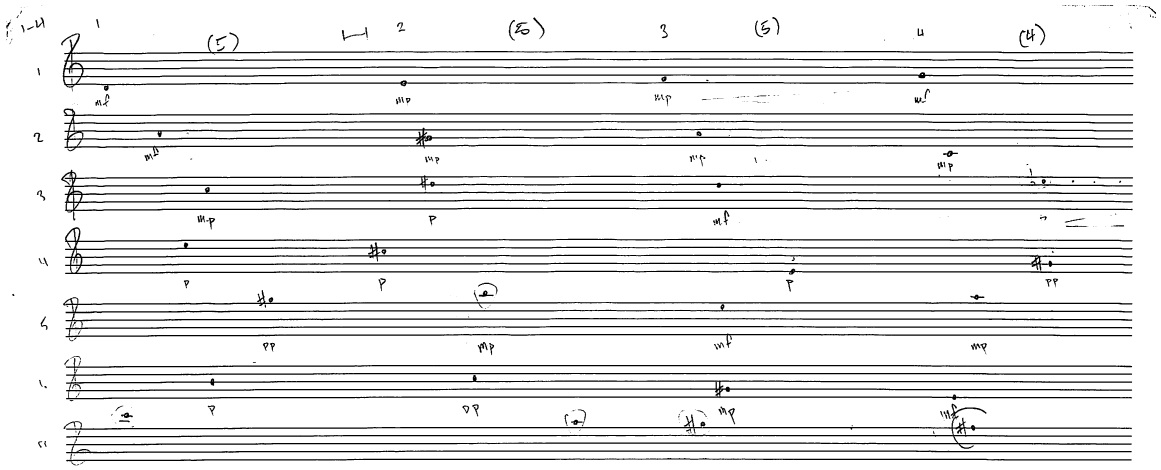
Return an Array of the absolute time positions of each note.

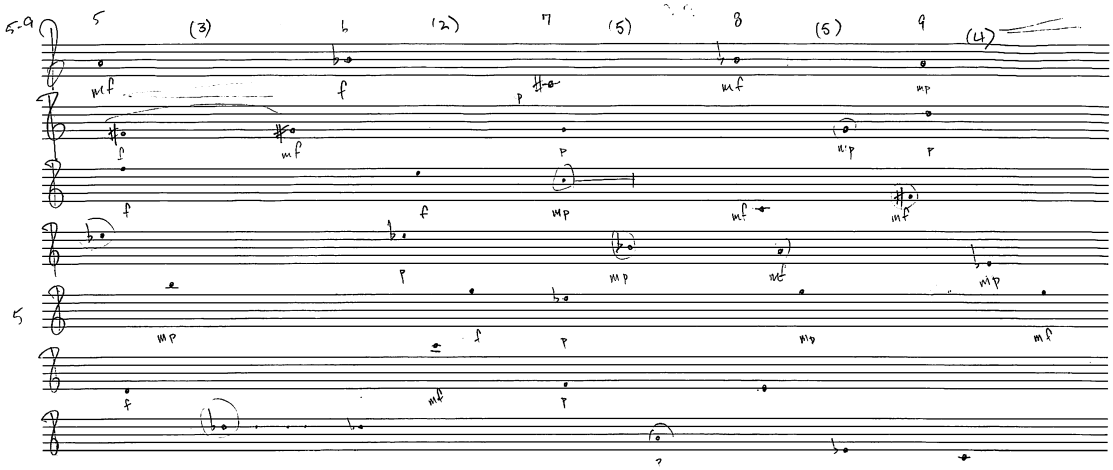
Appendix B: Scores

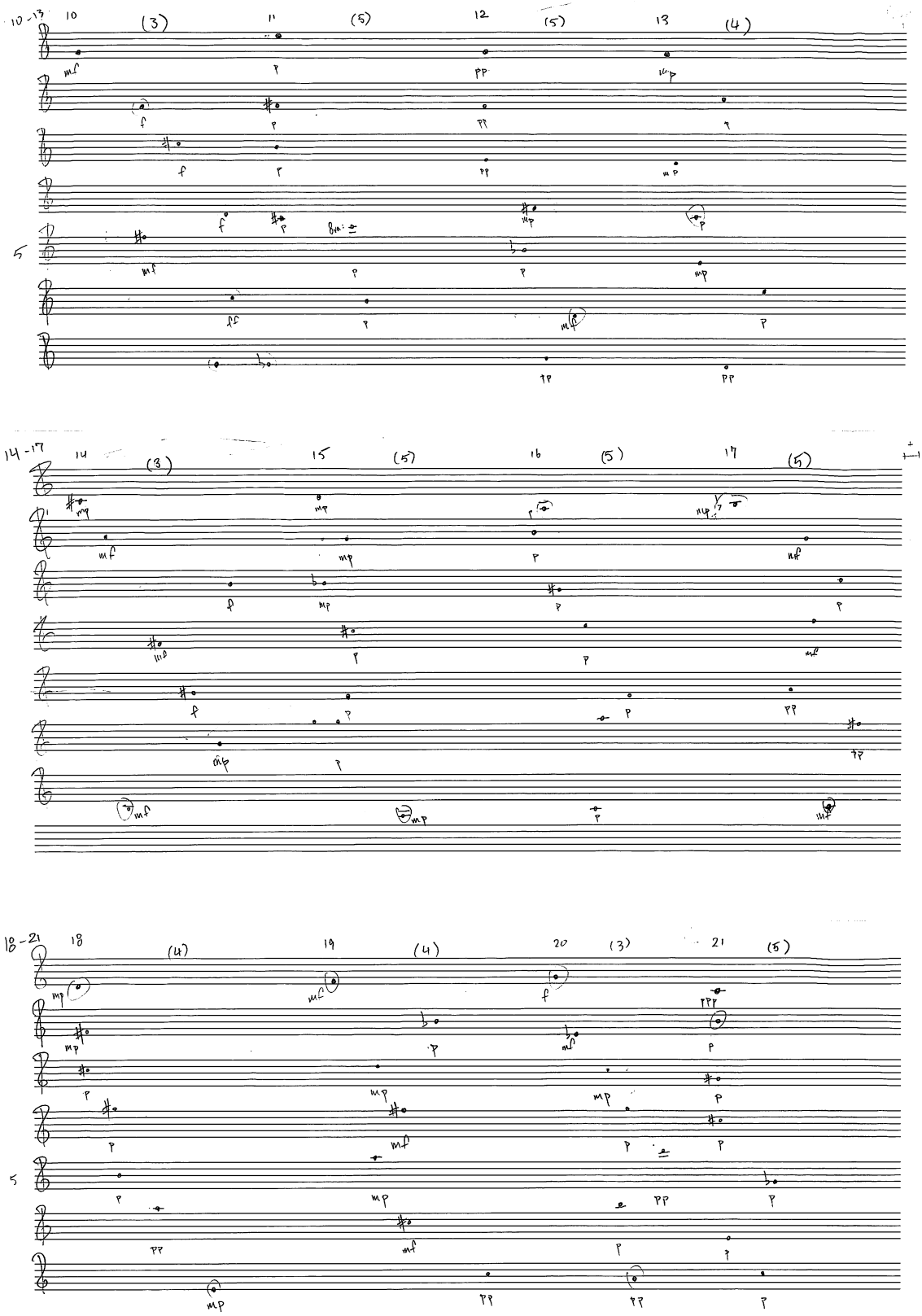
1. Parable of the Sower
2. Unknowable City no. 3
3. Unknowable City no. 5
4. Bloom Suite
5. Facets
6. Two Nocturnes
7. New Year's Day
8. Flows
9. Flowerpot Music (NYU Book)

Parable of the Sower (2007)

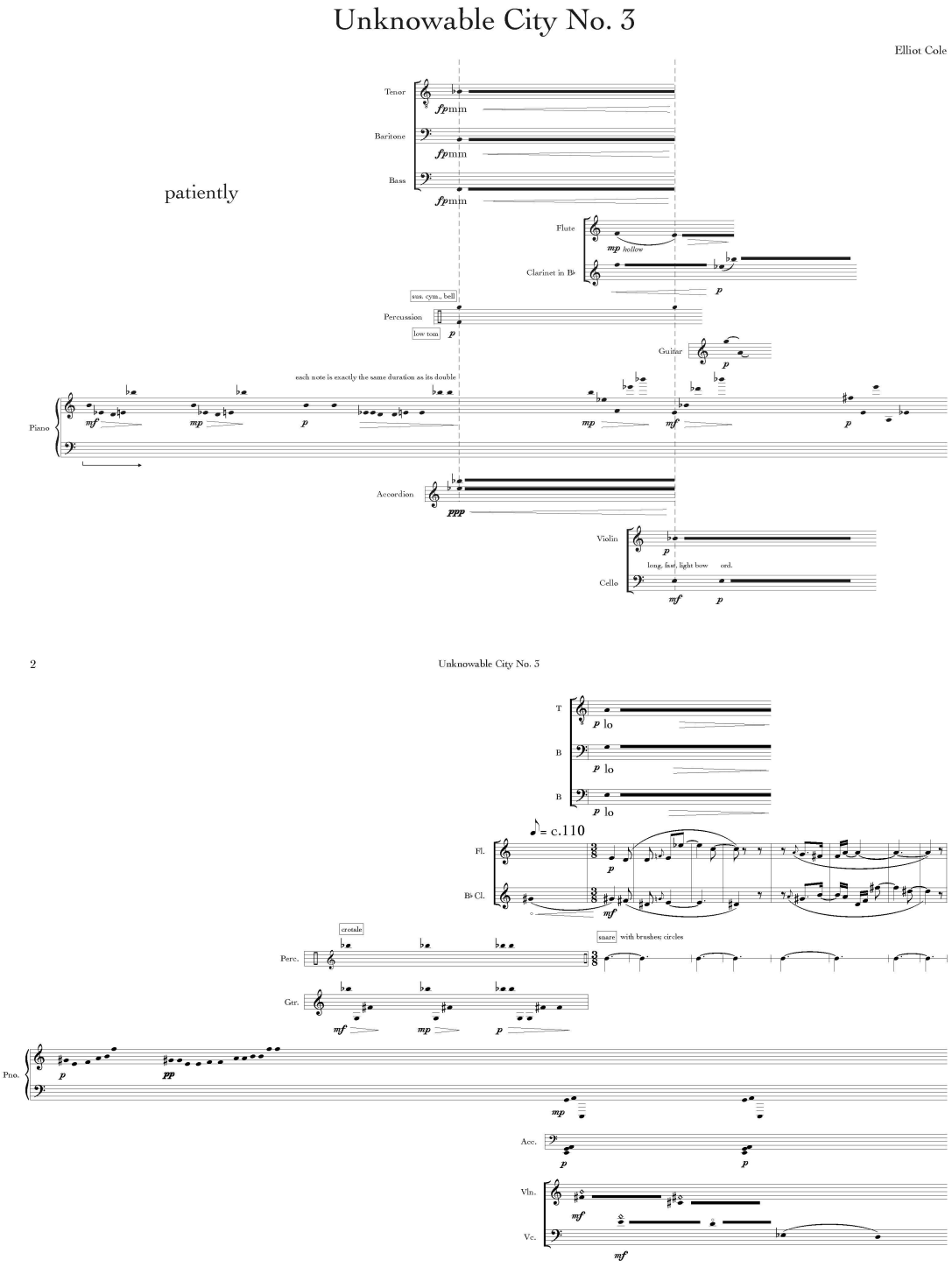


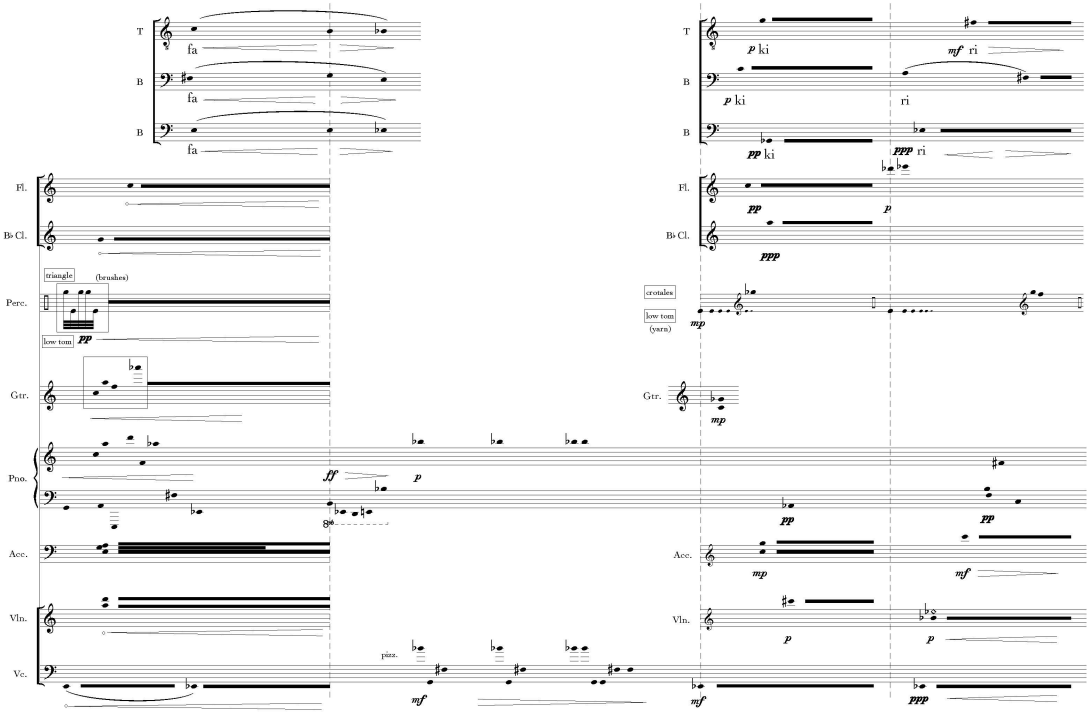


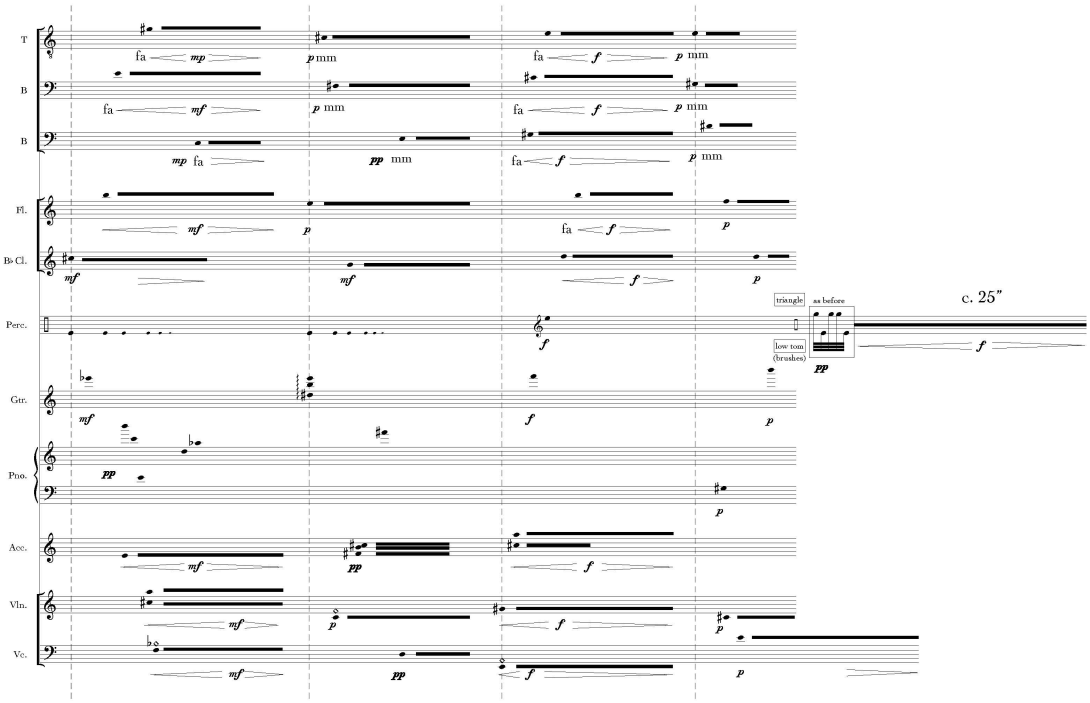


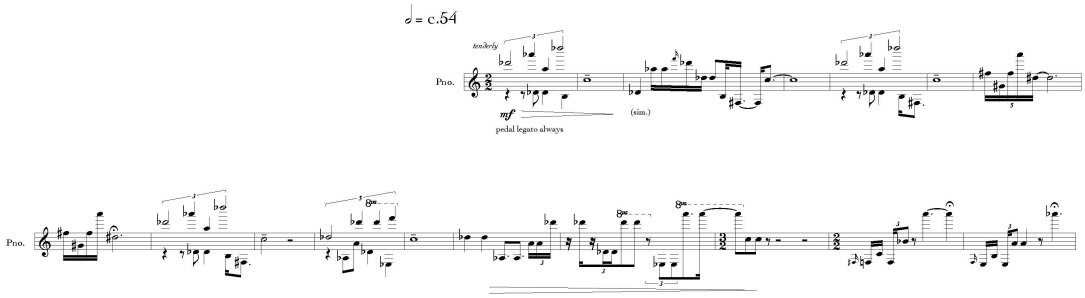


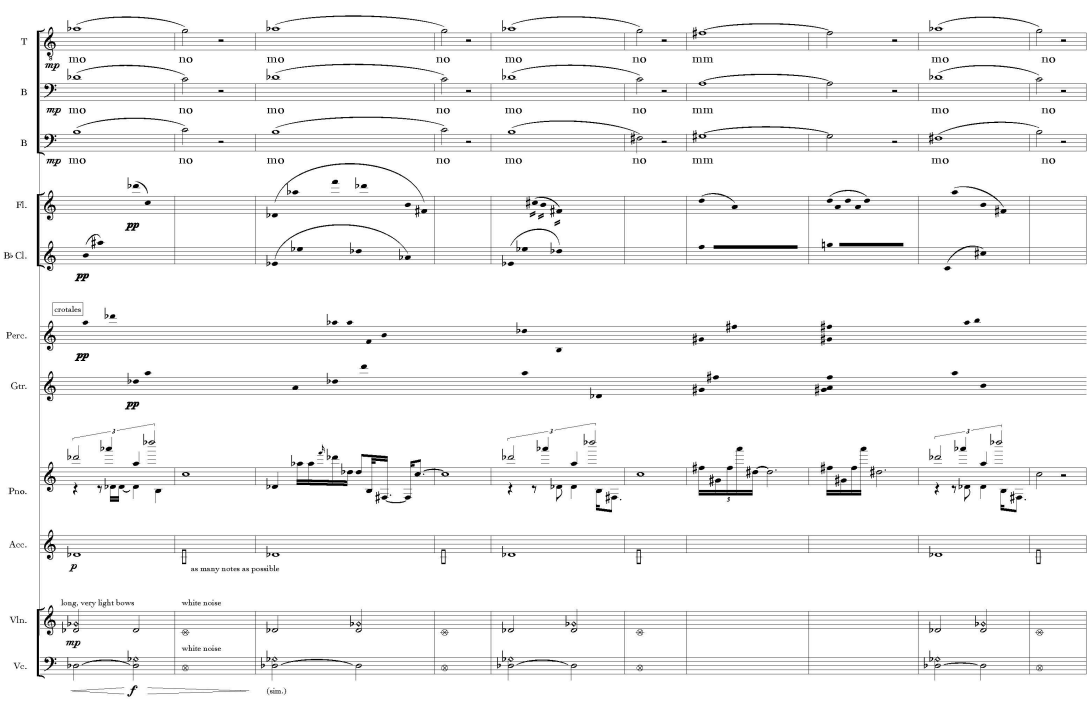
Unknowable City no. 3 (2011)

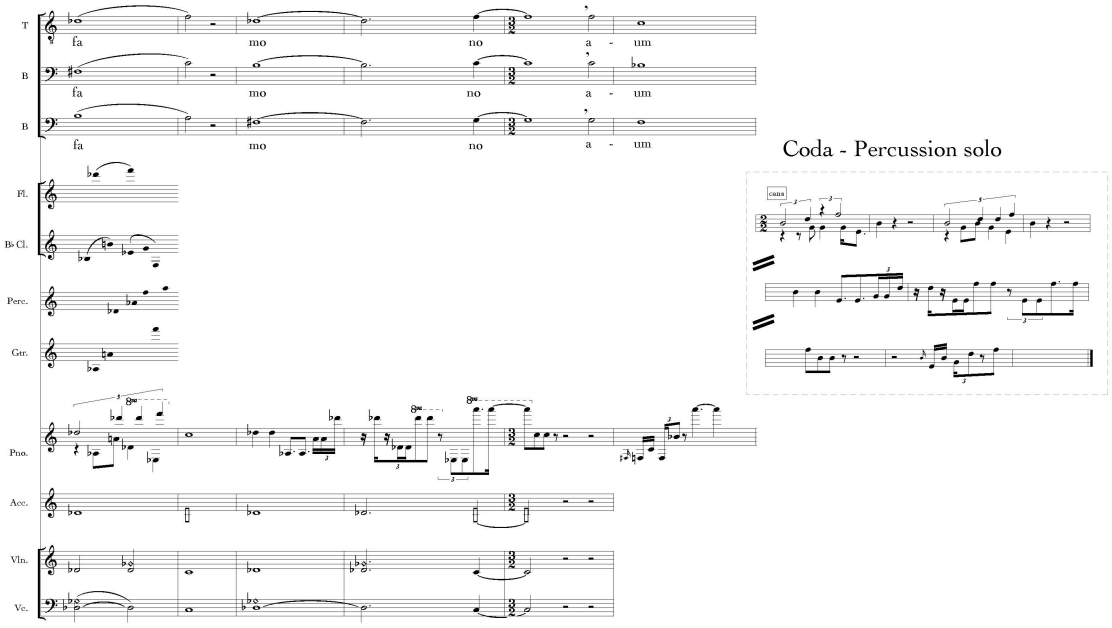



T
B
B
fa
fa
fa
p ki
mf ri
p ki
ri
pp ki
ppp ri
Fl.
B♭ Cl.
pp
p
ppp
triangle
(brushes)
Perc.
low tom
pp
crotales
low tom
(yarn)
mp
Gtr.
mp
Pno.
ff
p
pp
pp
Acc.
mp
mf
Vln.
p
p
pizz.
Vc.
mf
mf
ppp

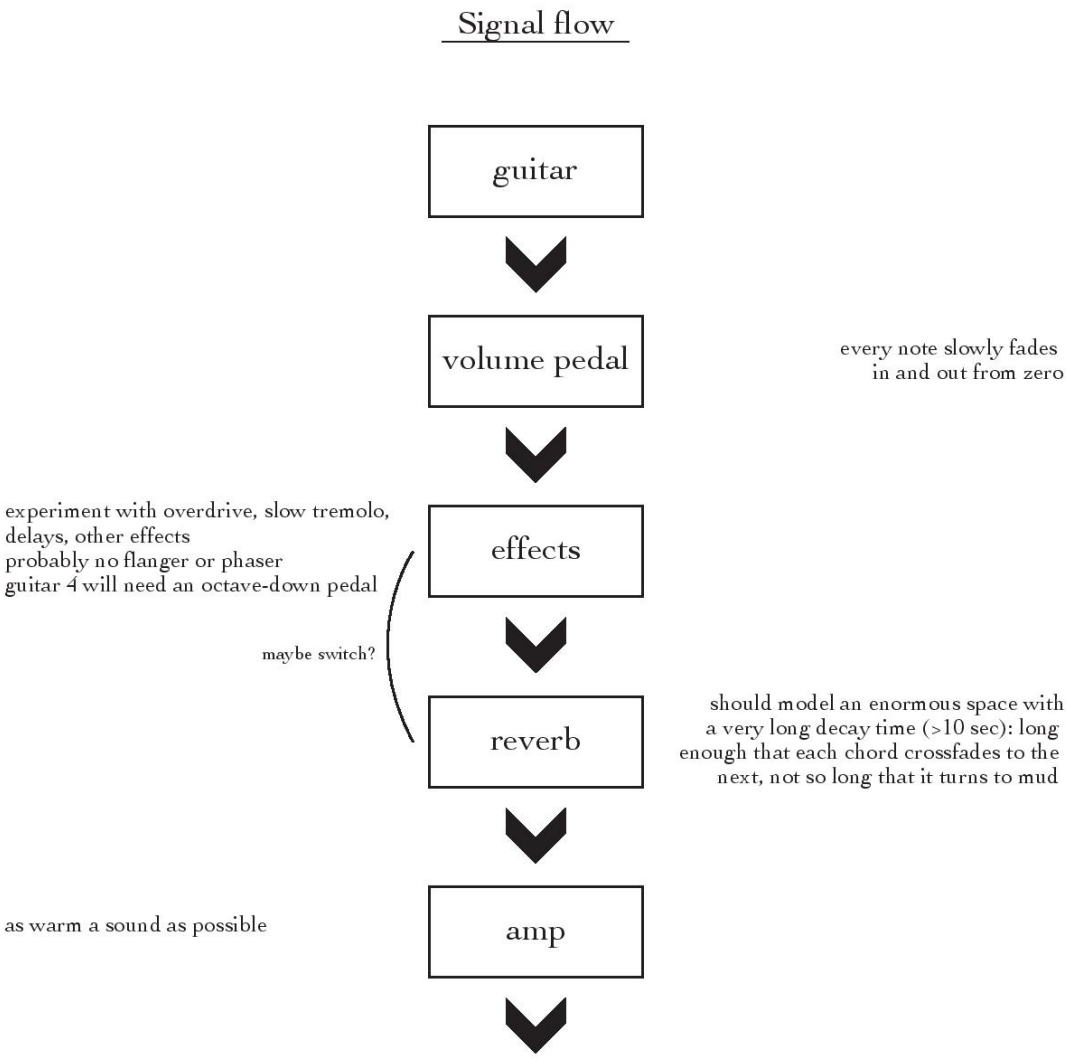

T
B
B
fa mp
p mm
fa f p mm
fa mf
p mm
fa f p mm
mp fa
pp mm
fa f p mm
Fl.
B♭ Cl.
mf
p
fa f
p
mf
mf
f
p
triangle
as before
c. 25″
low tom
(brushes)
pp
Perc.
f
f
Gtr.
mf
f
p
Pno.
pp
p
Acc.
mf
pp
f
Vln.
mf
p
f
p
Vc.
mf
pp
f
p


♩ = c.54
tenderly
Pno.
mf
pedal legato always
(sim.)
Pno.


T
B
B
mp
mo no mo no mo no mm mo no
Fl.
B♭ Cl.
pp
crotales
Perc.
Gtr.
Pno.
Acc.
p
as many notes as possible
long, very light bows
white noise
Vln.
mp
white noise
Vc.
f
(sim.)

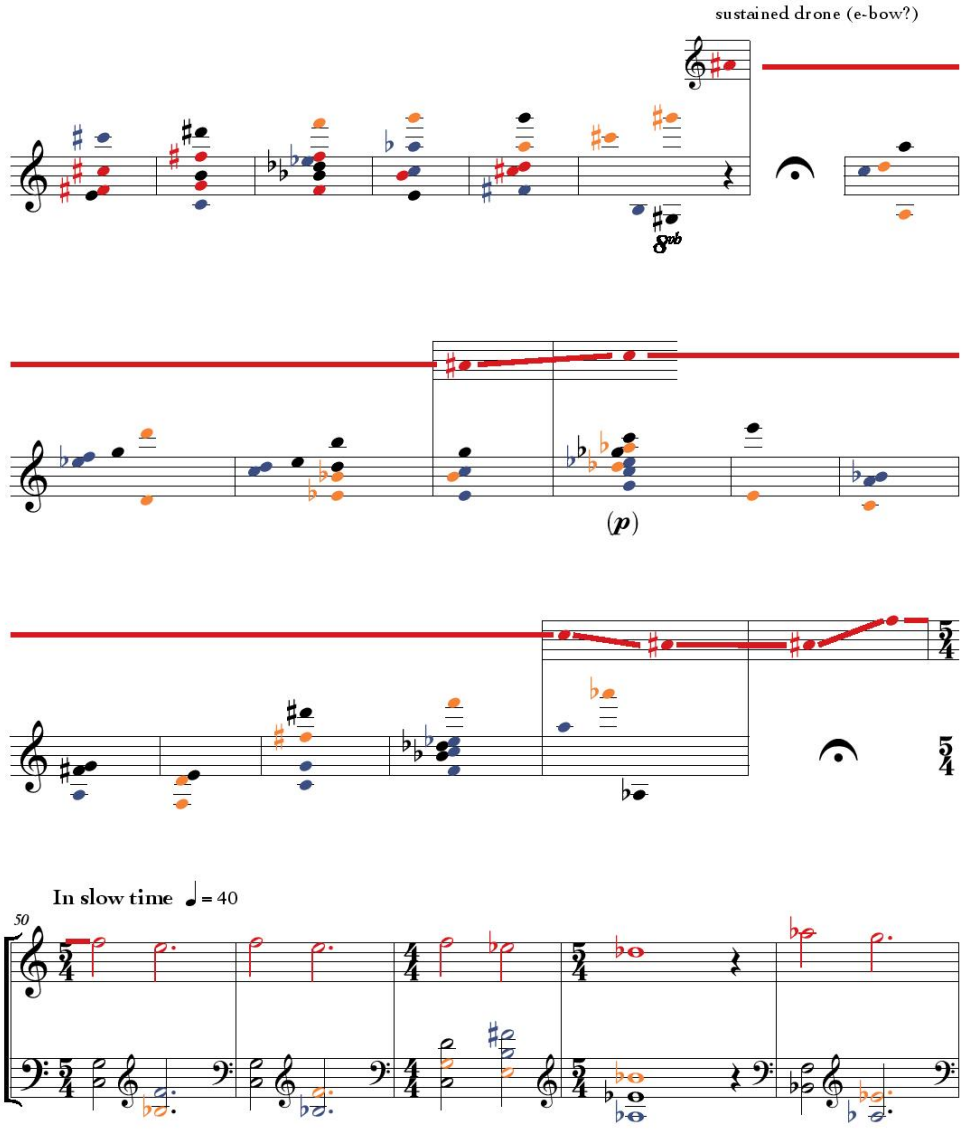

T
fa mo no a - um
B
fa mo no a - um
B
fa mo no a - um
Fl.
B♭ Cl.
Perc.
Gtr.
Pno.
Acc.
Vln.
Vc.
Coda - Percussion solo
cans

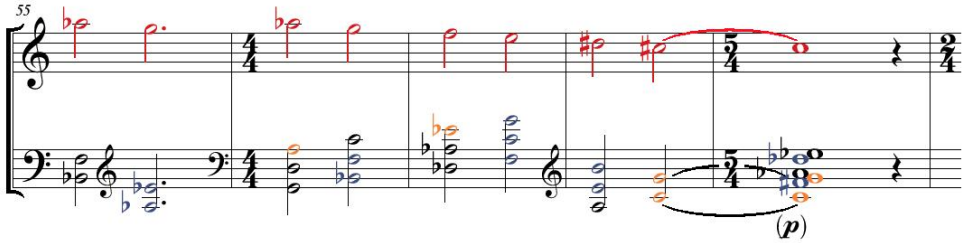
Unknowable City no. 5 (2011)



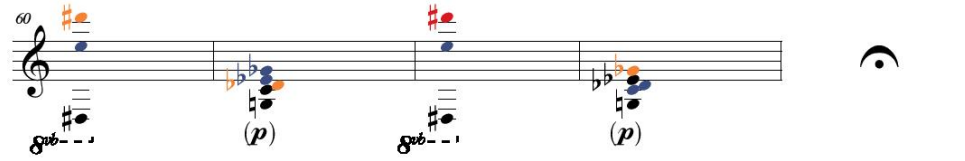
we'll develop much of the detail of the piece in rehearsal
I'm open to both clean/etherial and fuzzy/wall of noise surfaces
in general, each guitar should have a different color of sound,
but not so different that they don't blend

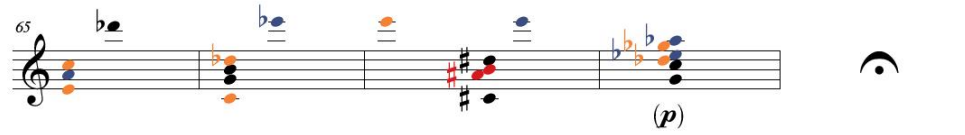
guitar 1 will need to sustain a note somehow -
an e-bow or special effects set up will be necessary

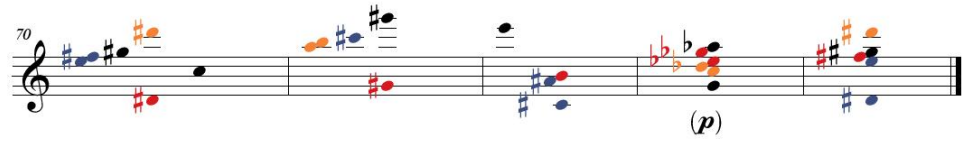

sustained drone (e-bow?)
8vb
(p)
In slow time ♩= 40
50



Tempo I

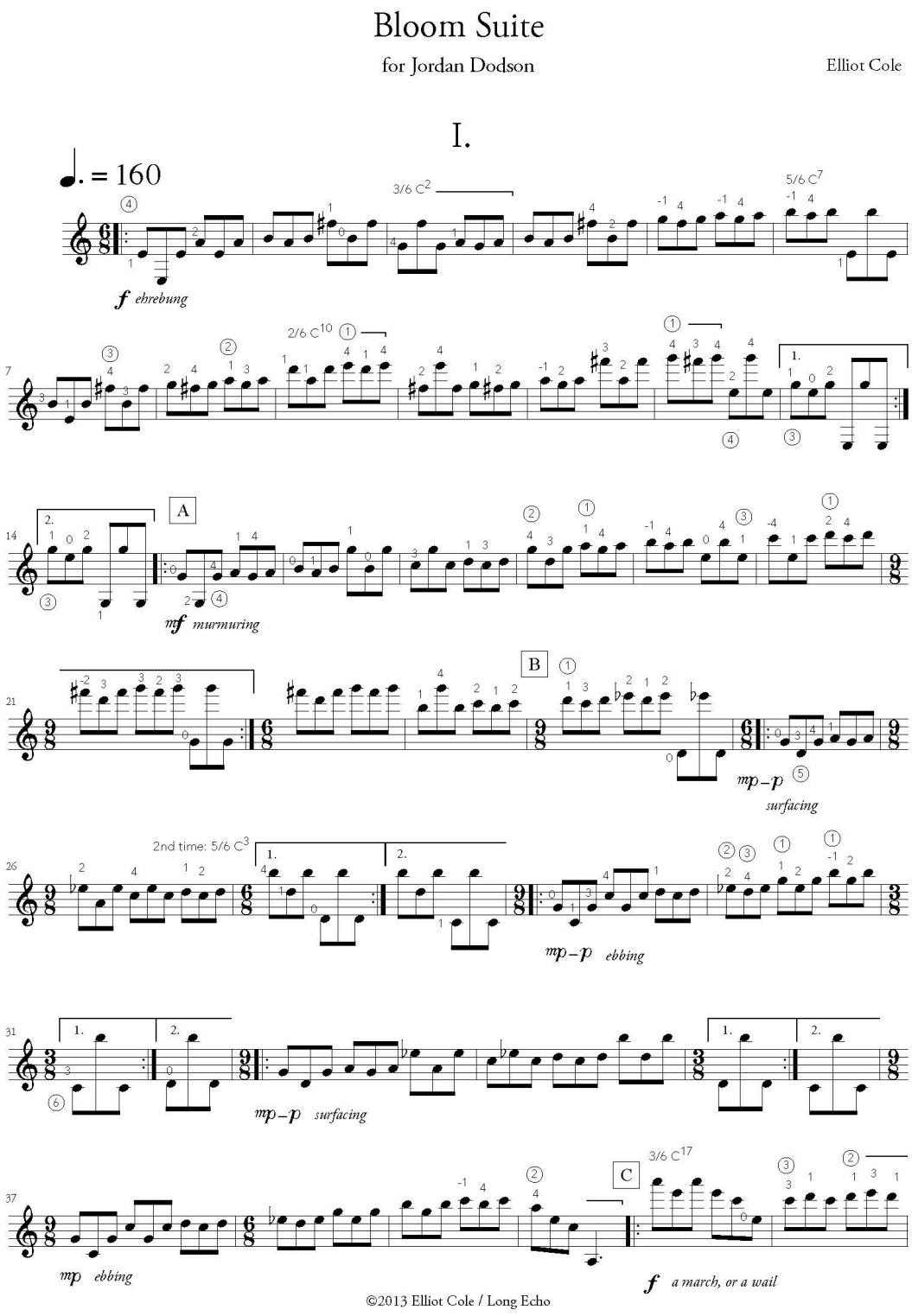


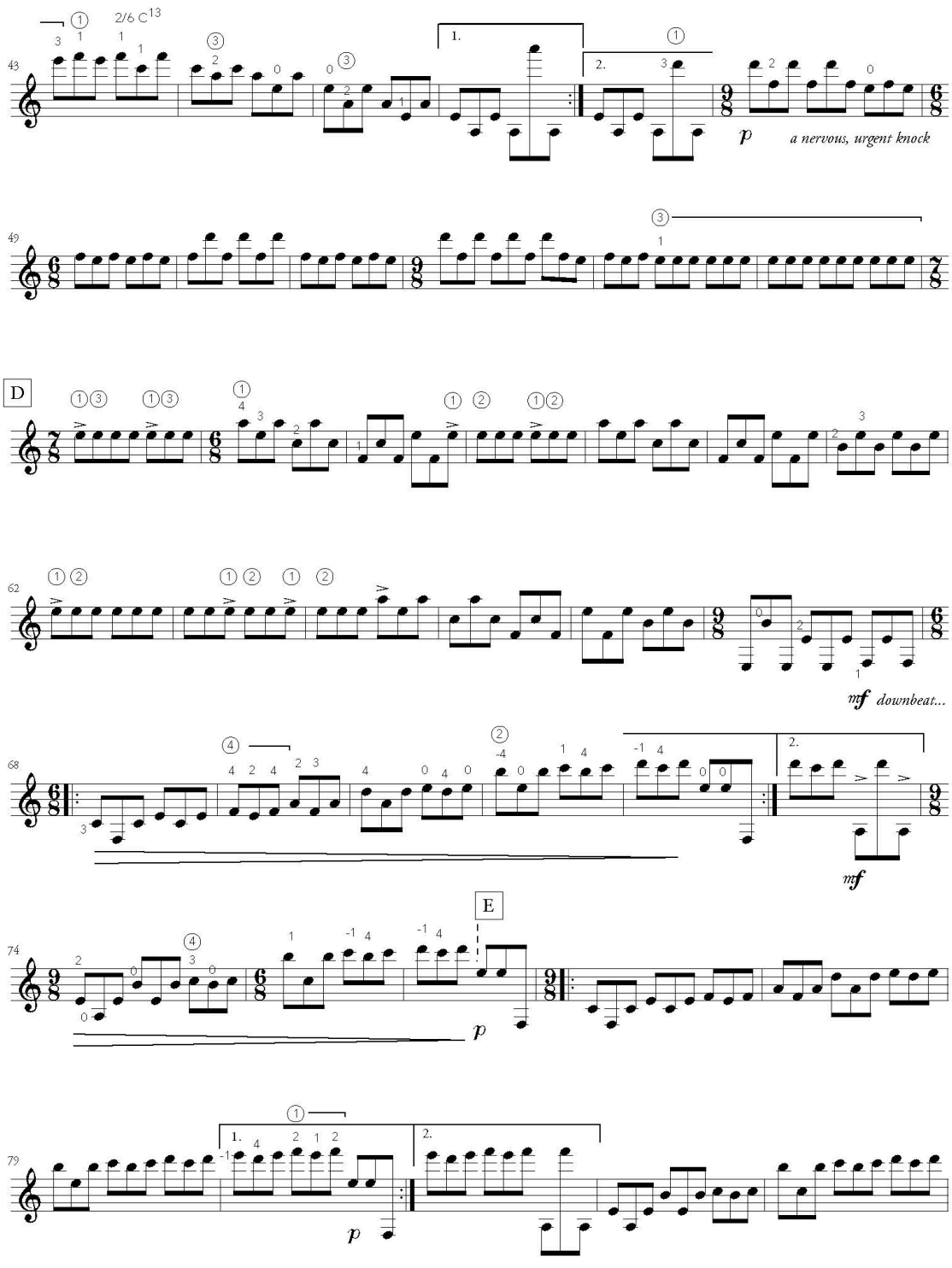


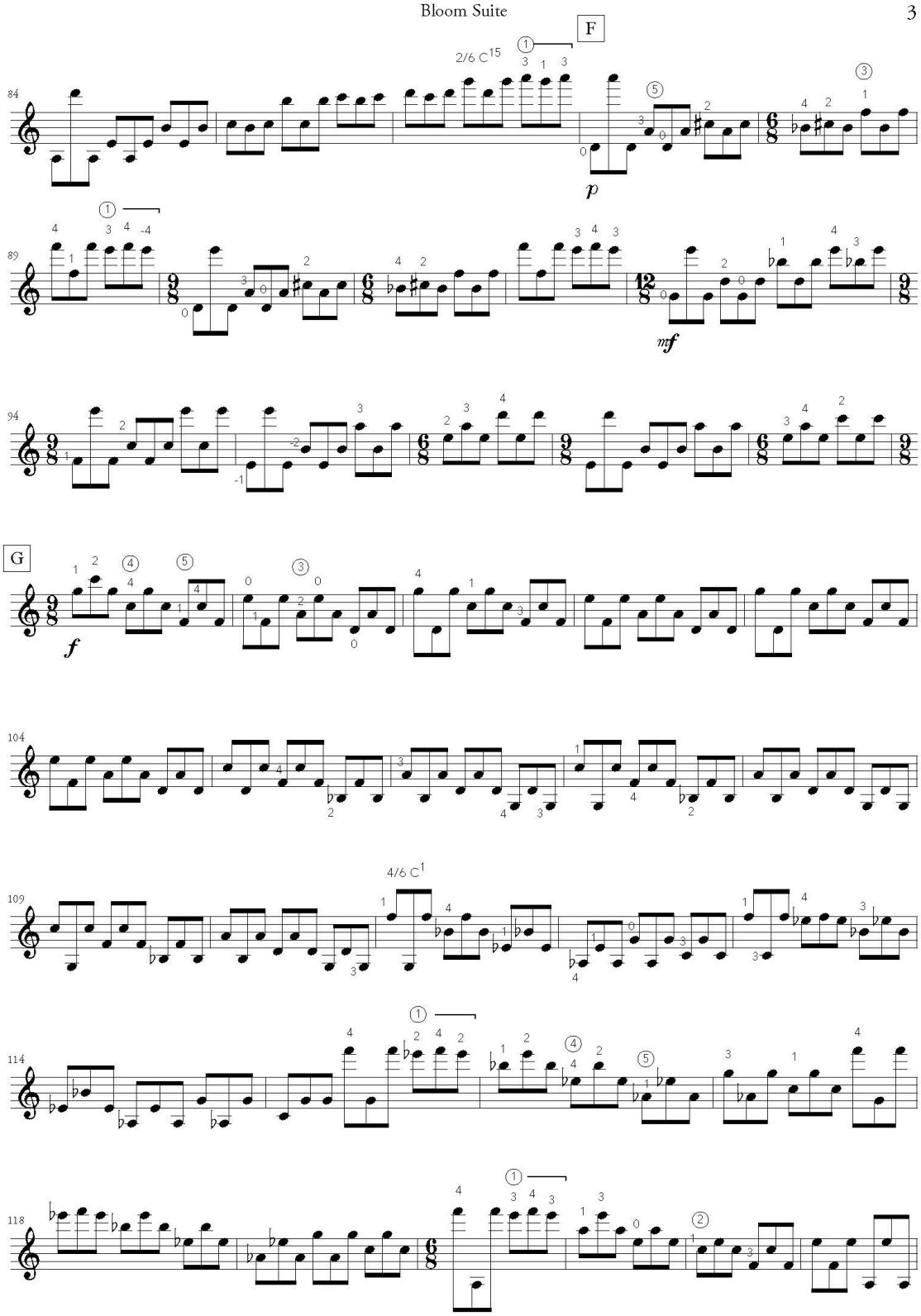


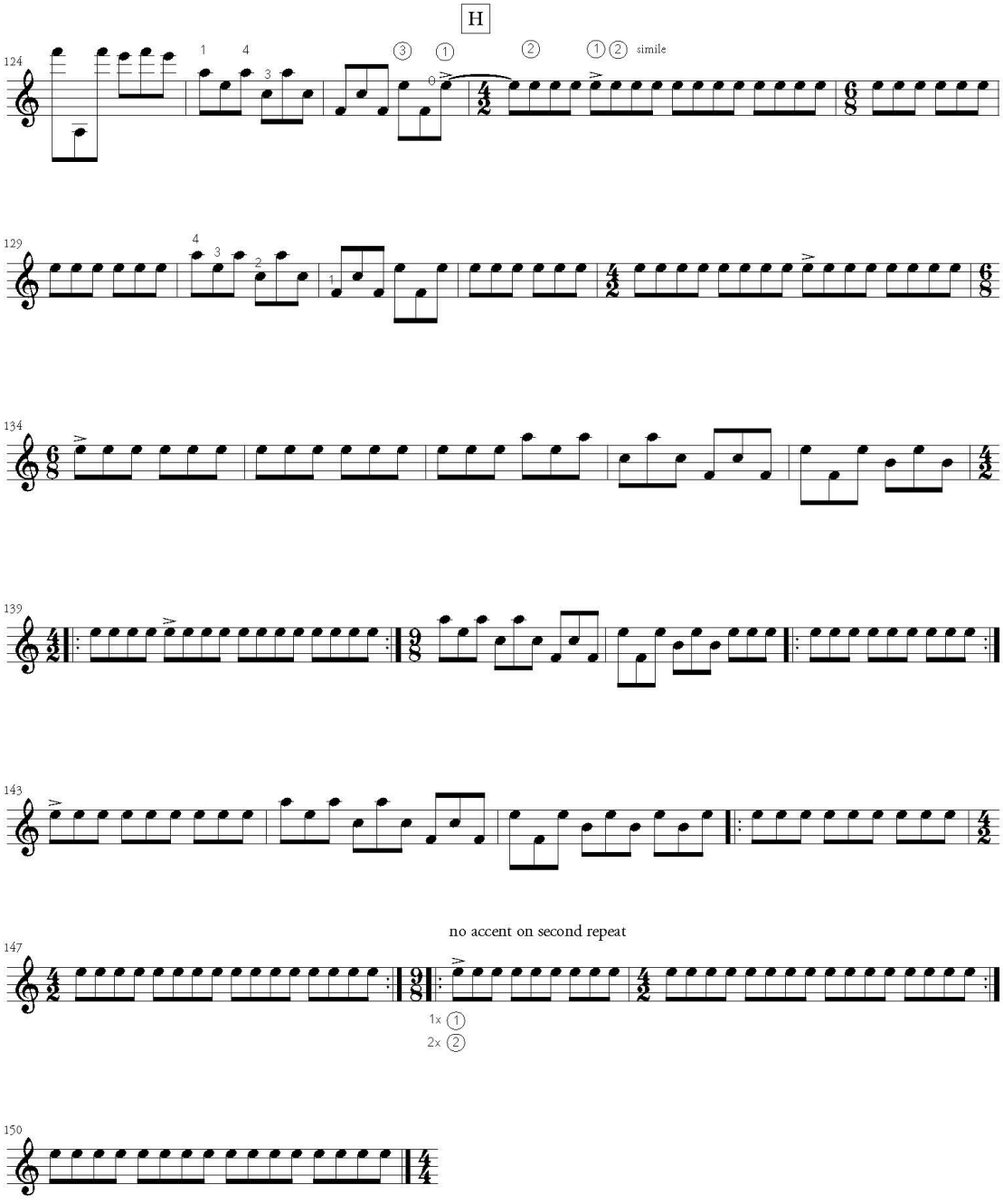
Total time: c. 10-12'

Bloom Suite (2013)

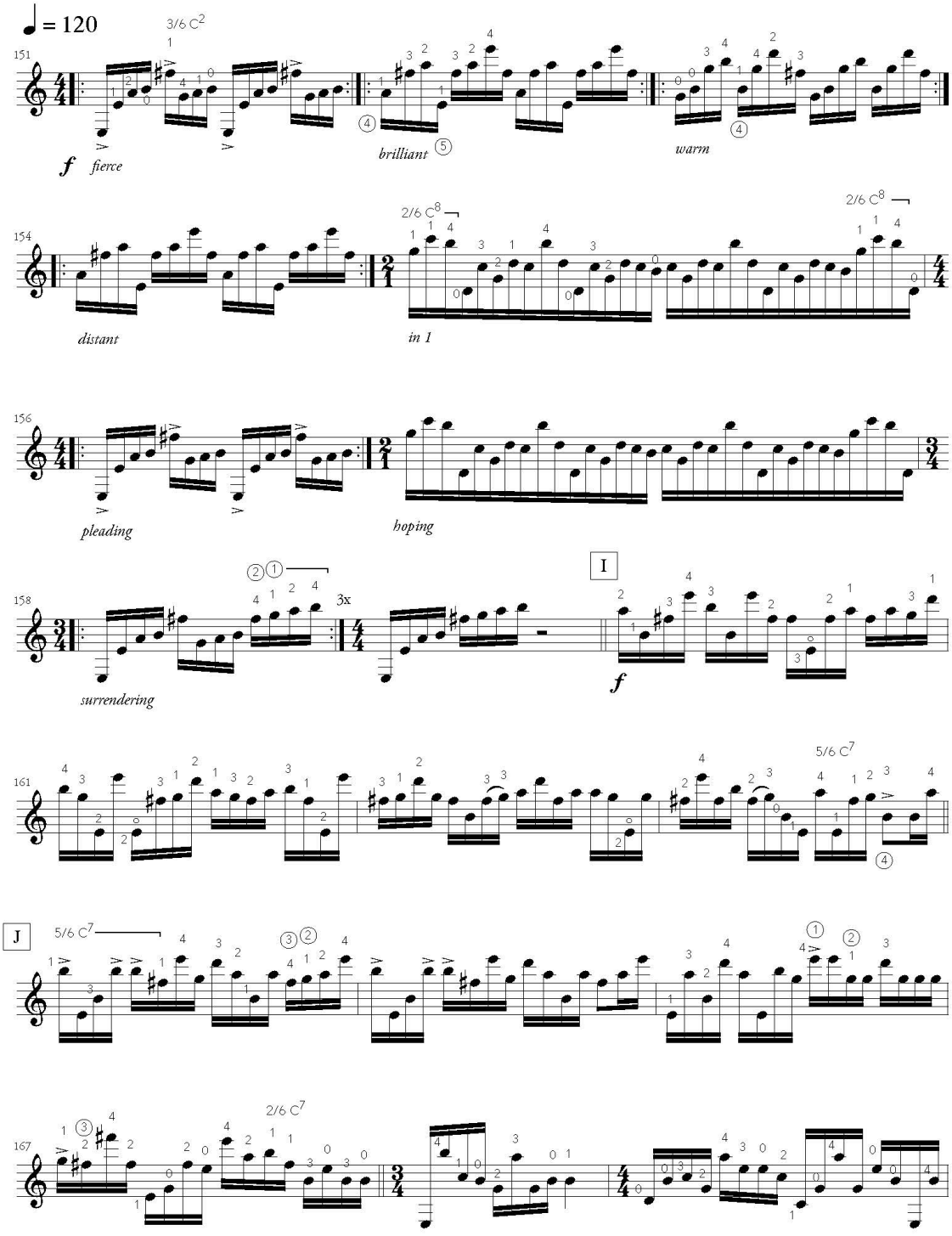


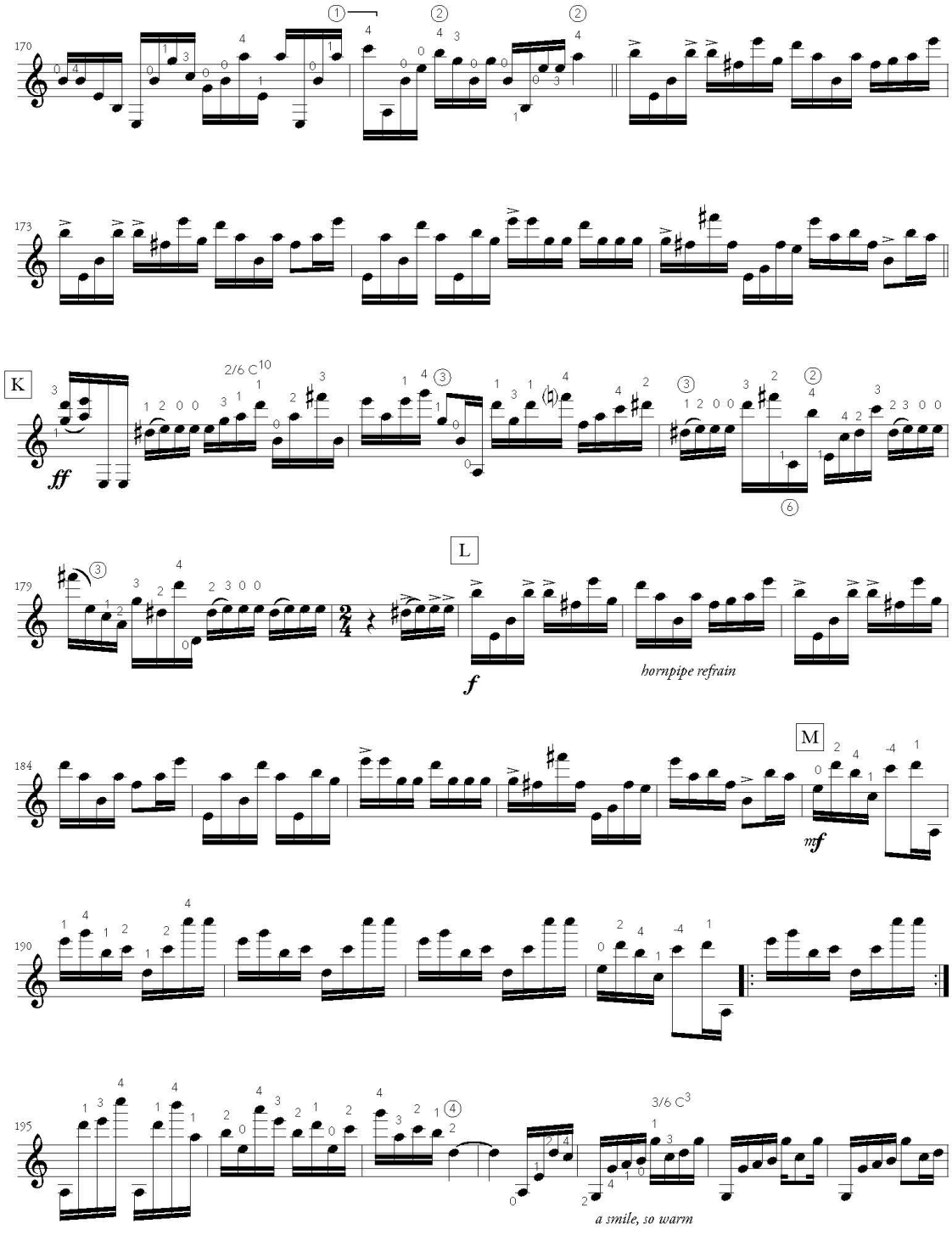

43
2/6 C13
1.
2.
p
a nervous, urgent knock
49
D
62
mf
downbeat...
68
mf
E
74
p
79
p

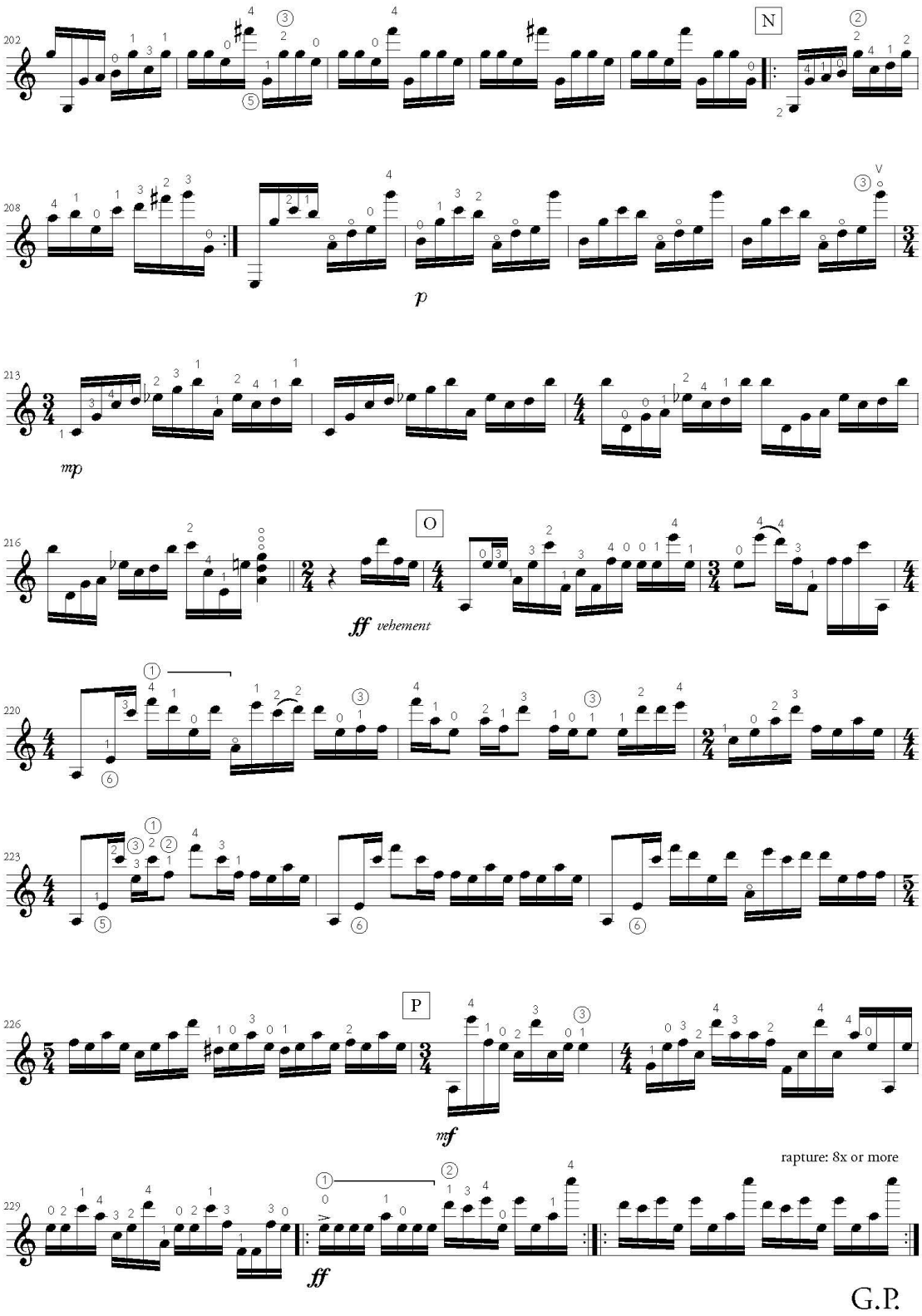

F
2/6 C15
84
p
89
mf
94
G
f
104
109
4/6 C1
114
118


H
124
simile
129
134
139
143
no accent on second repeat
147
1x
2x
150

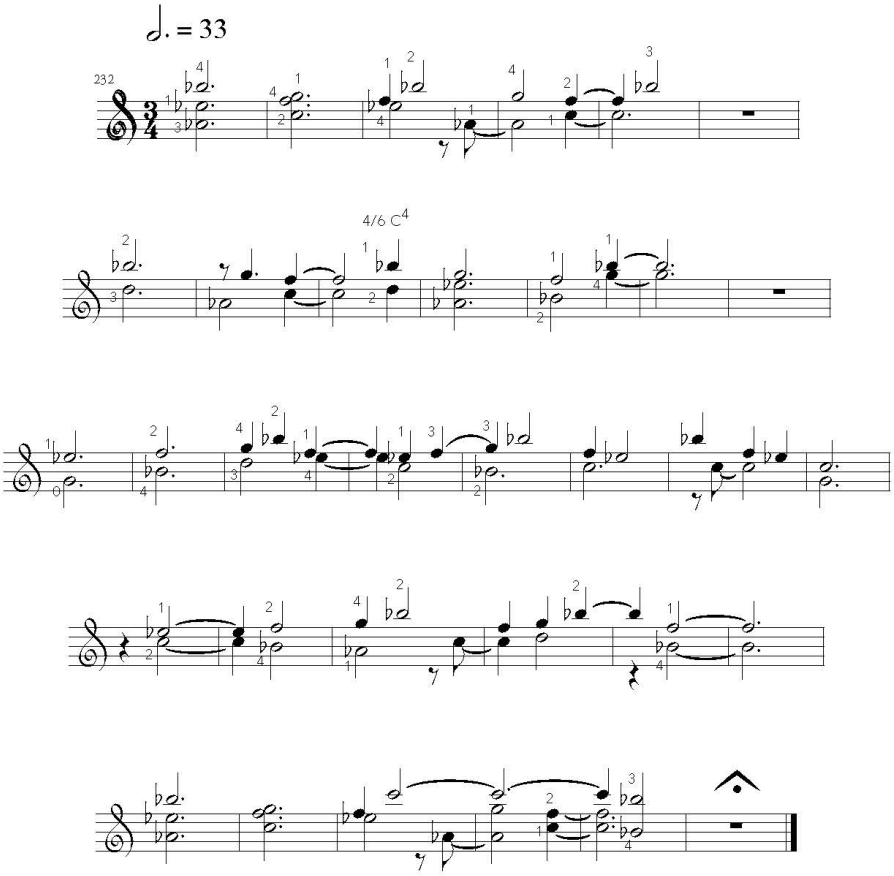
II.


♩ = 120
3/6 C2
151
f fierce
brilliant
warm
154
distant
2/6 C8
in 1
2/6 C8
156
pleading
hoping
158
3x
surrendering
I
f
161
5/6 C7
J
5/6 C7
167
2/6 C7

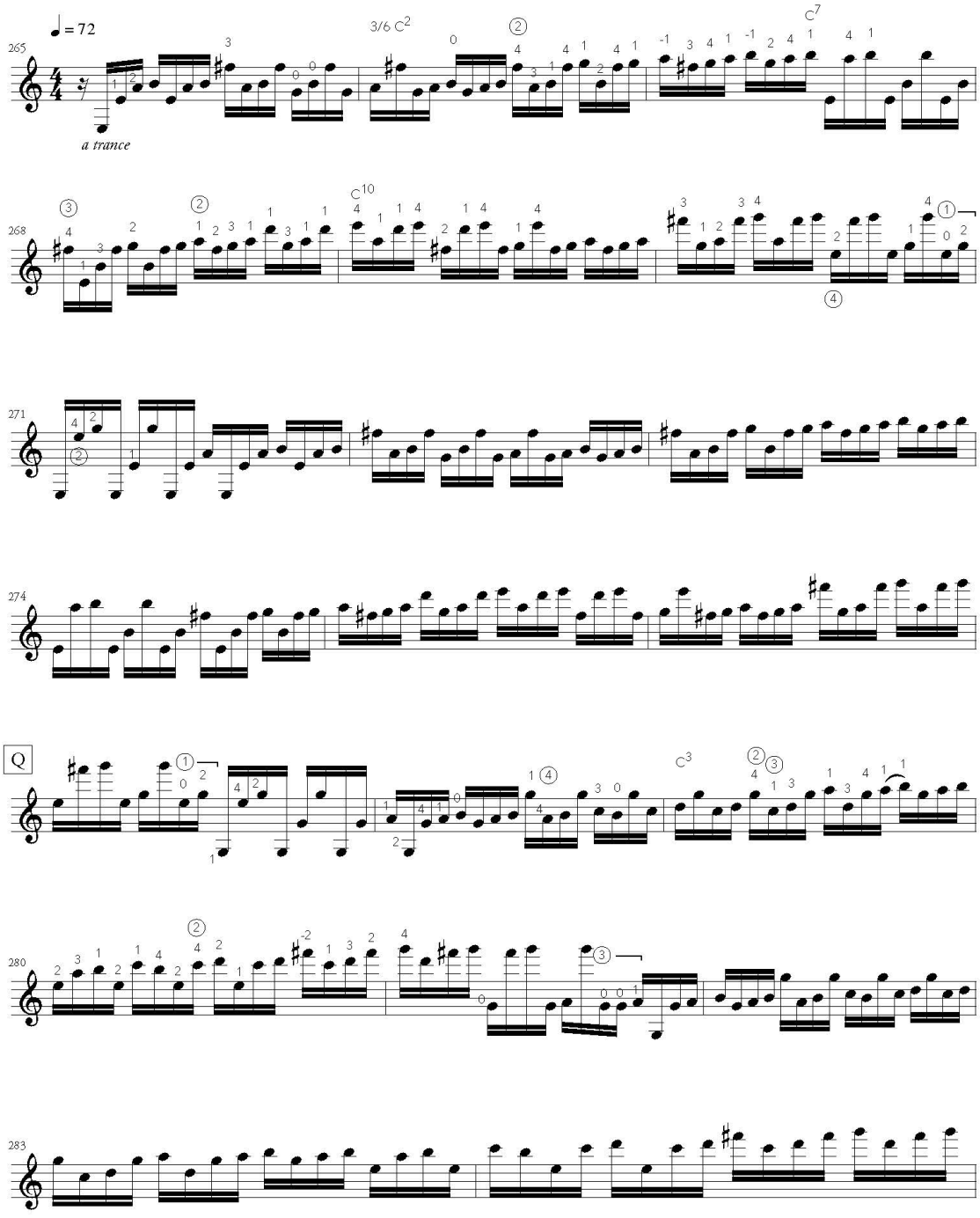

170
173
K
2/6 C10
ff
L
179
f
hornpipe refrain
M
184
mf
190
195
3/6 C3
a smile, so warm

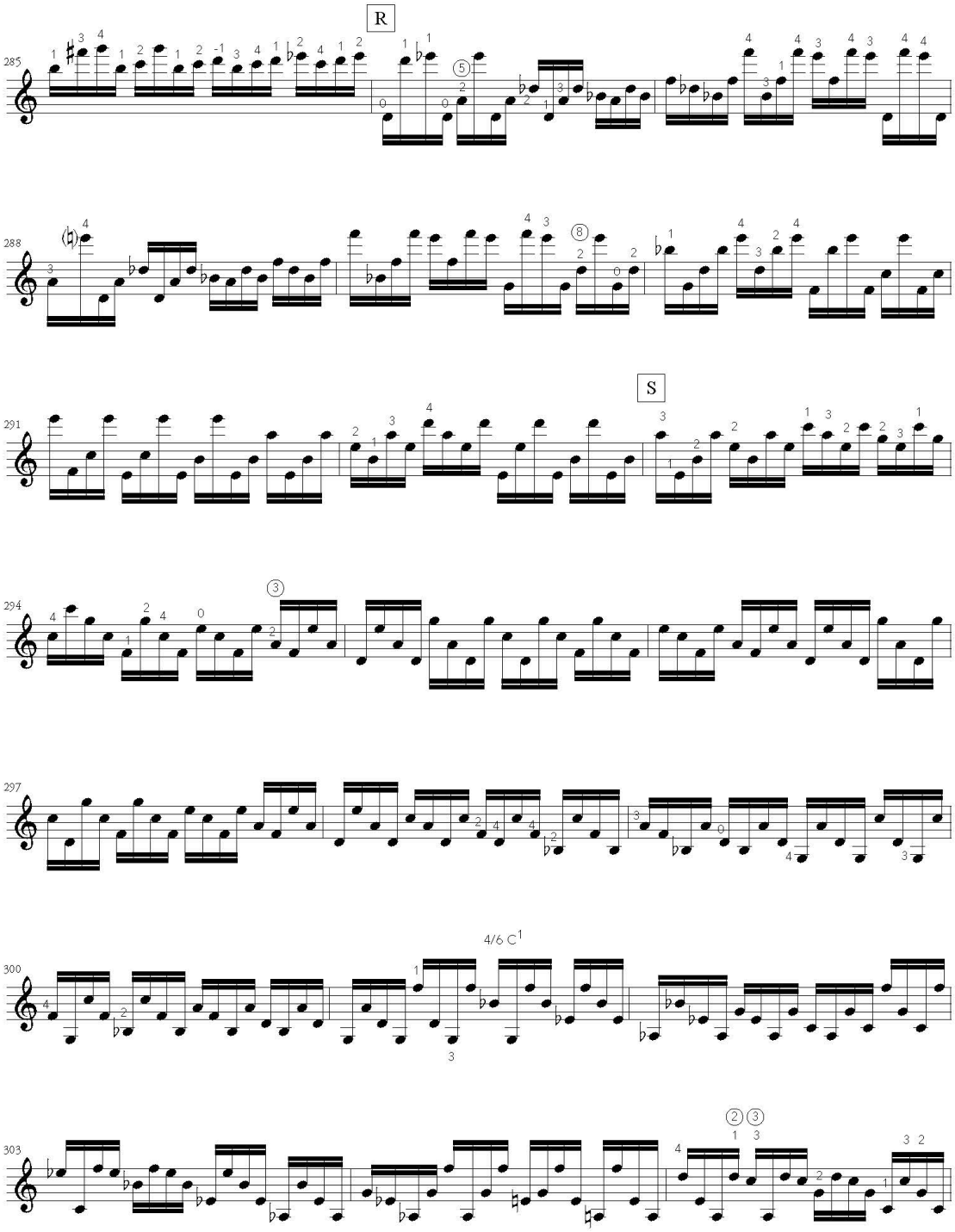

202
N
208
p
213
mp
216
O
ff vehement
220
223
226
P
mf
rapture: 8x or more
229
ff
G.P.

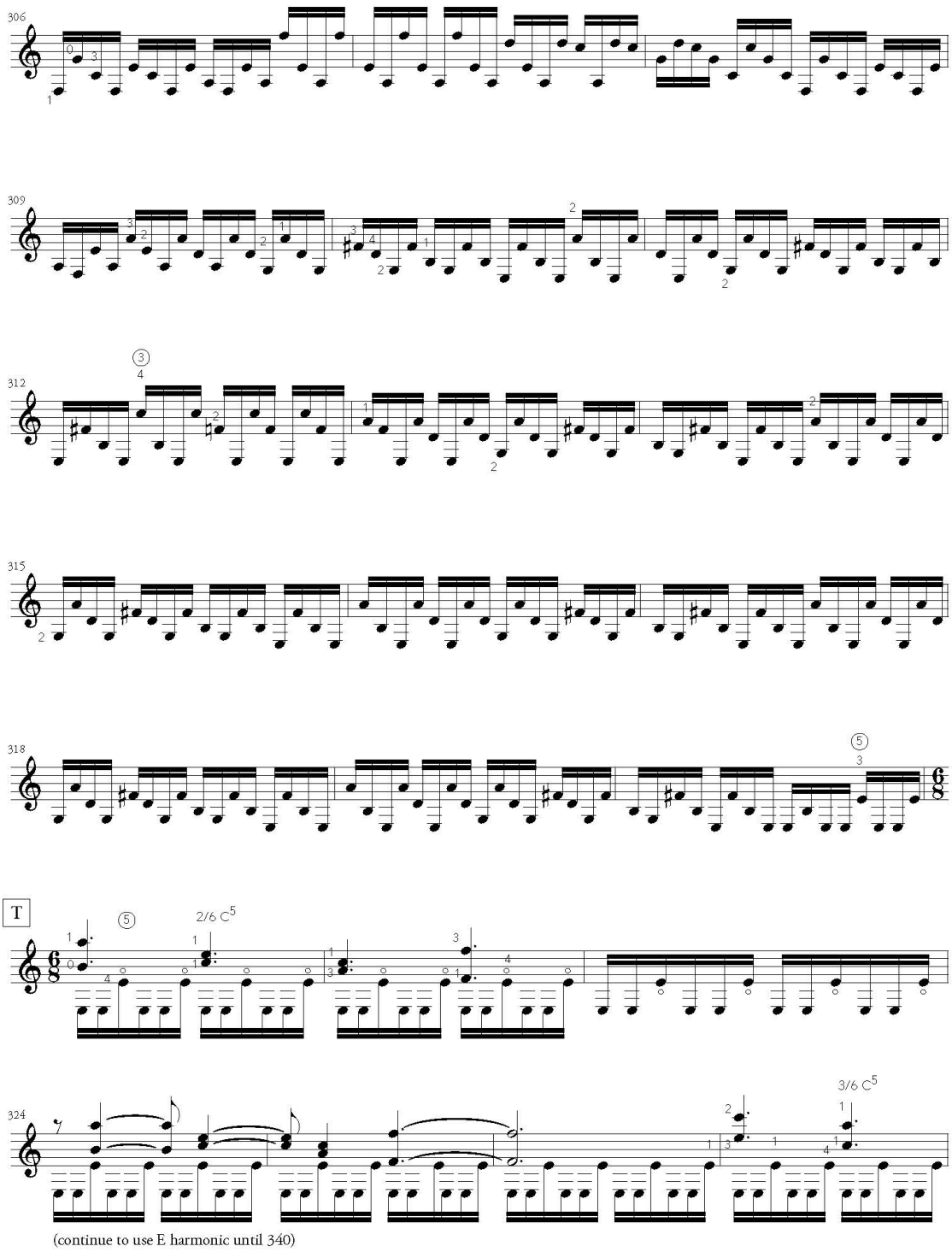
III.



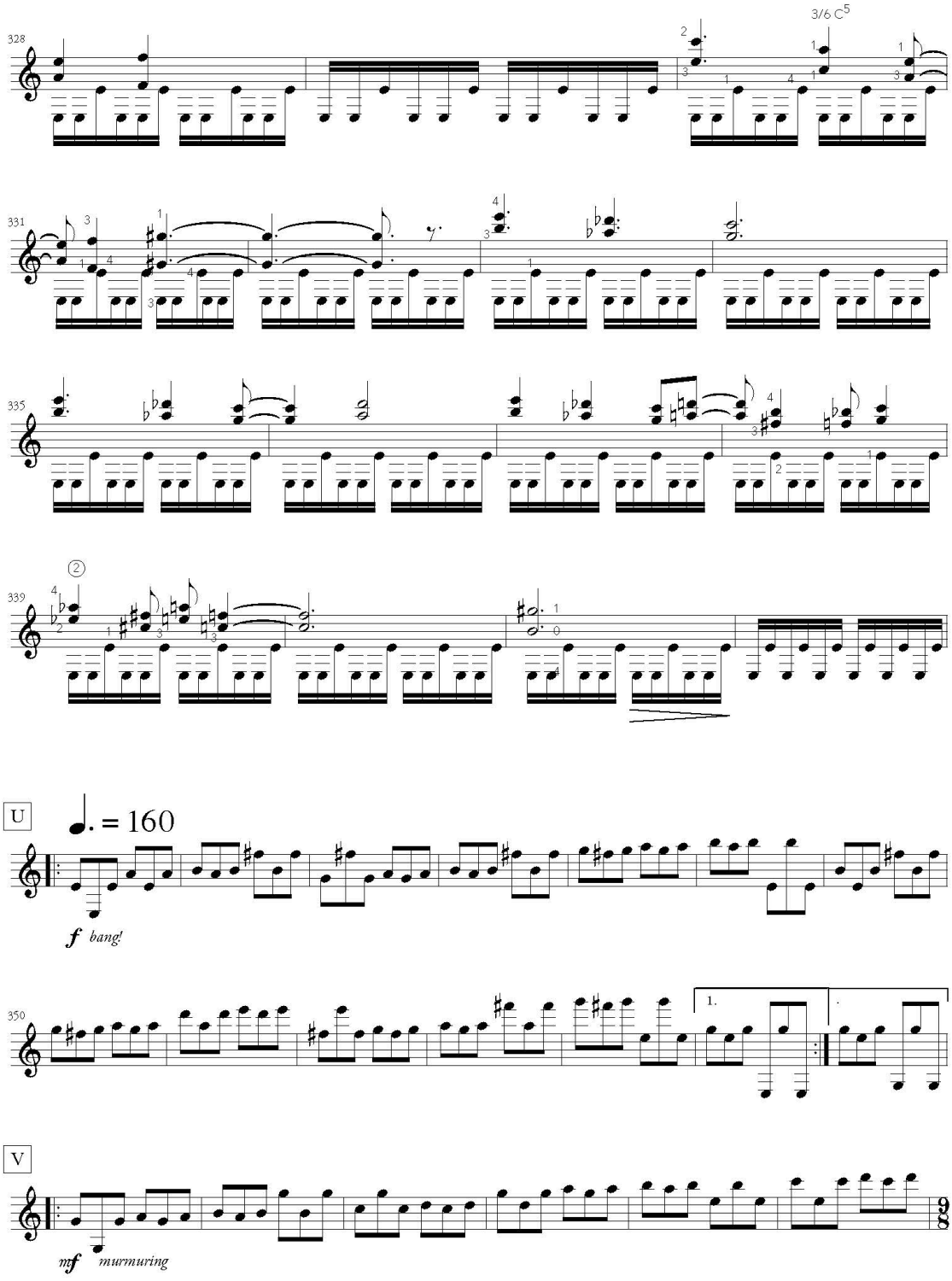
IV.

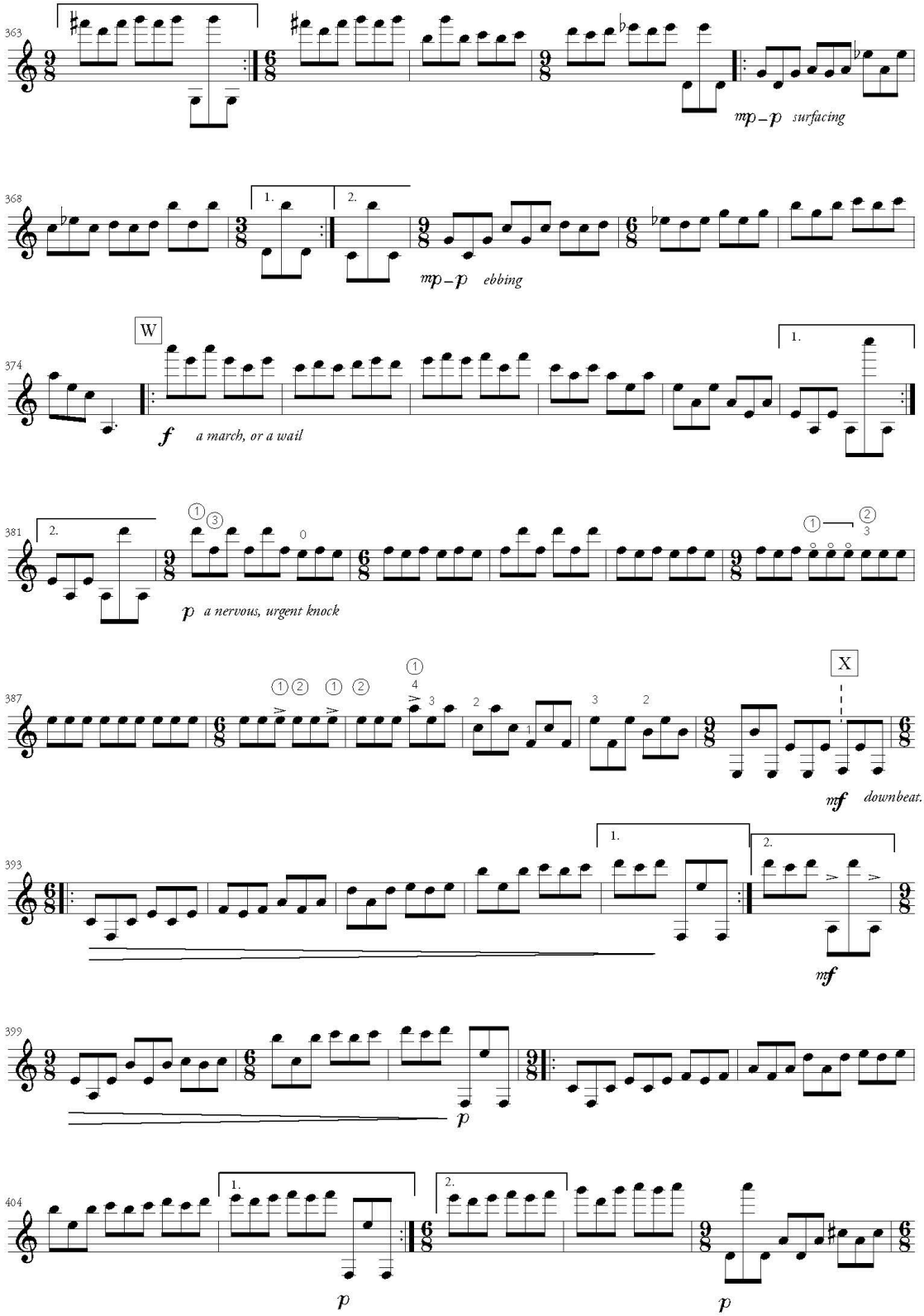


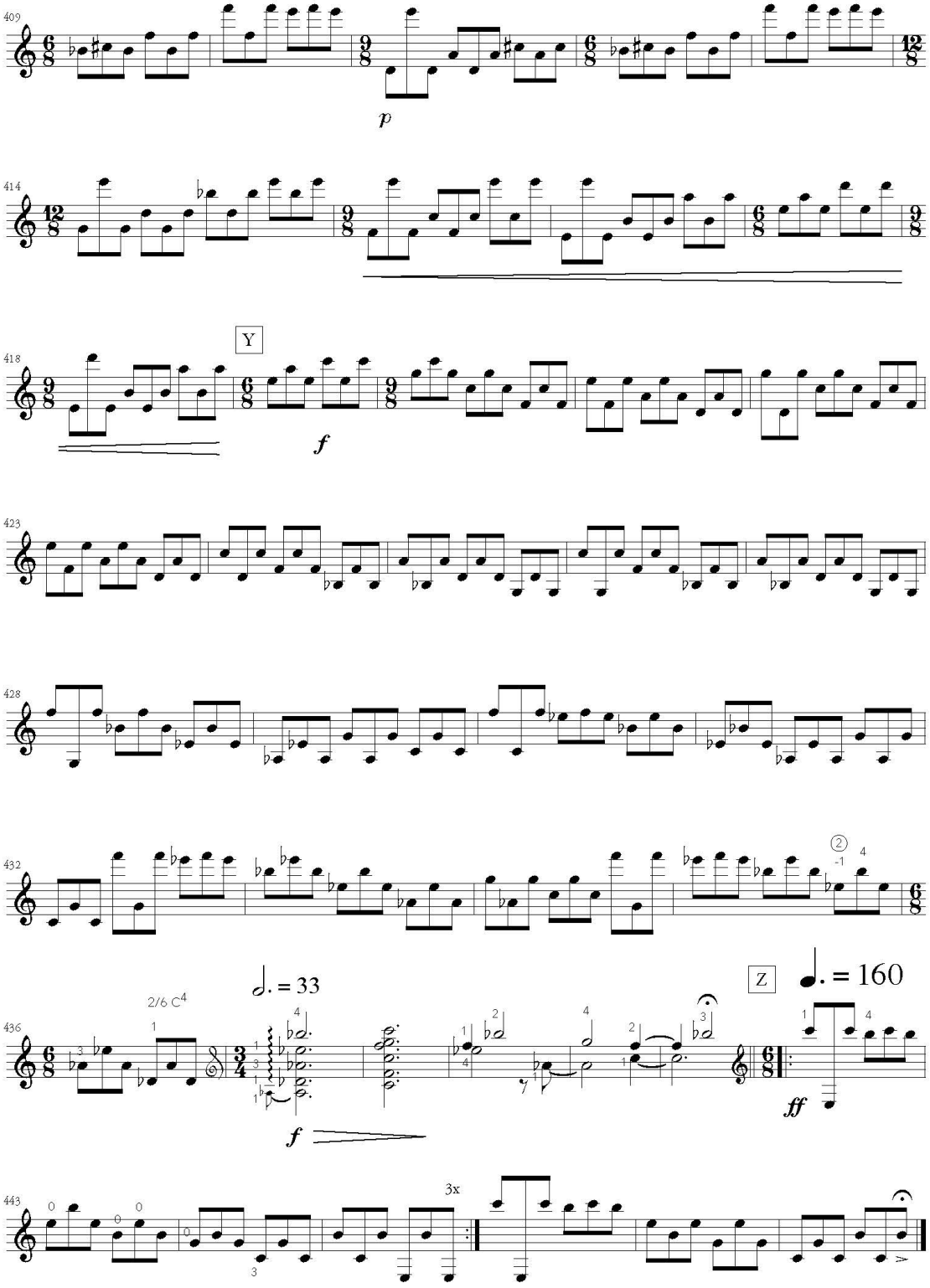

R
285
S
291
294
297
4/6 C1
300
303



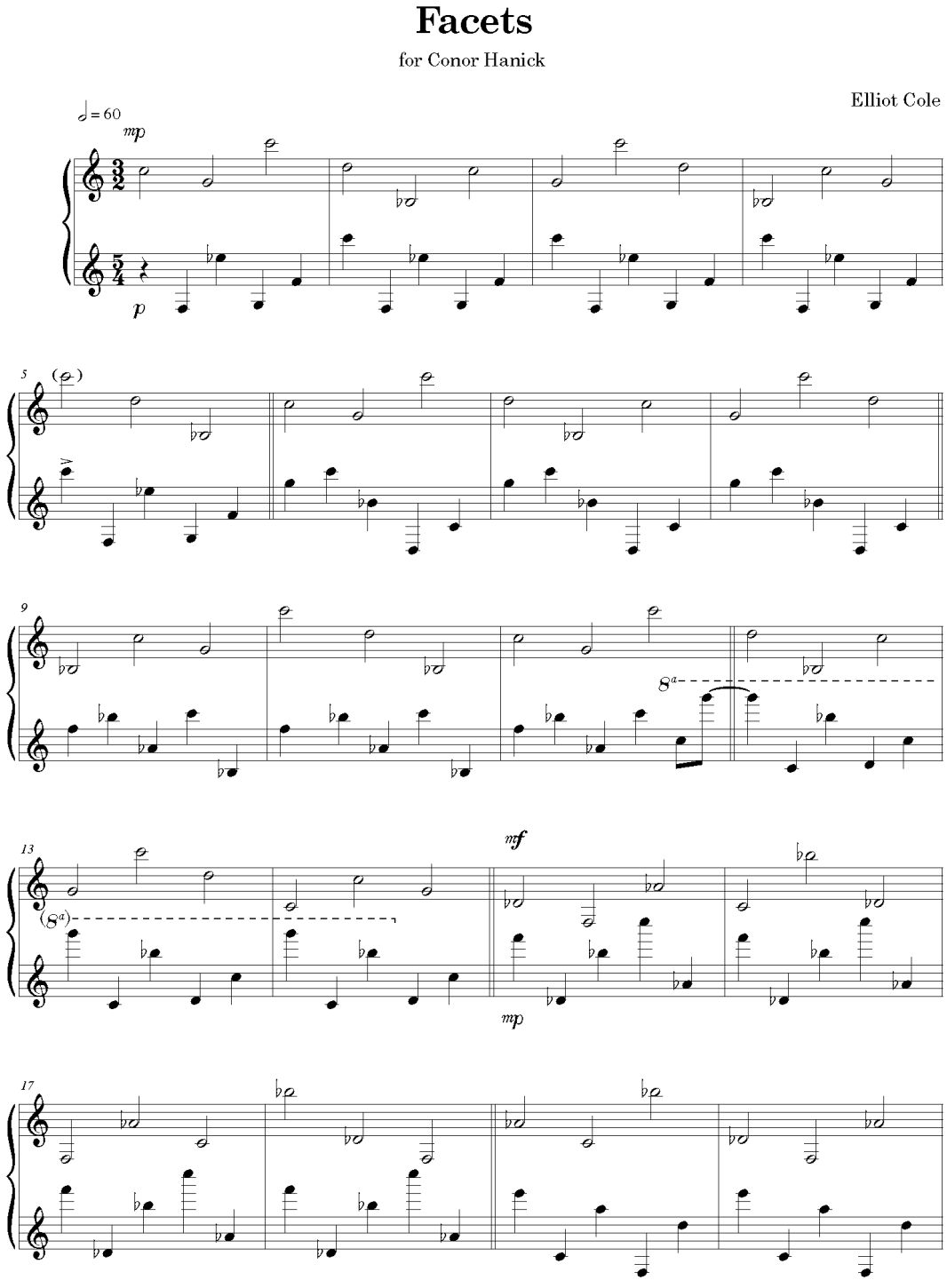
(continue to use E harmonic until 340)

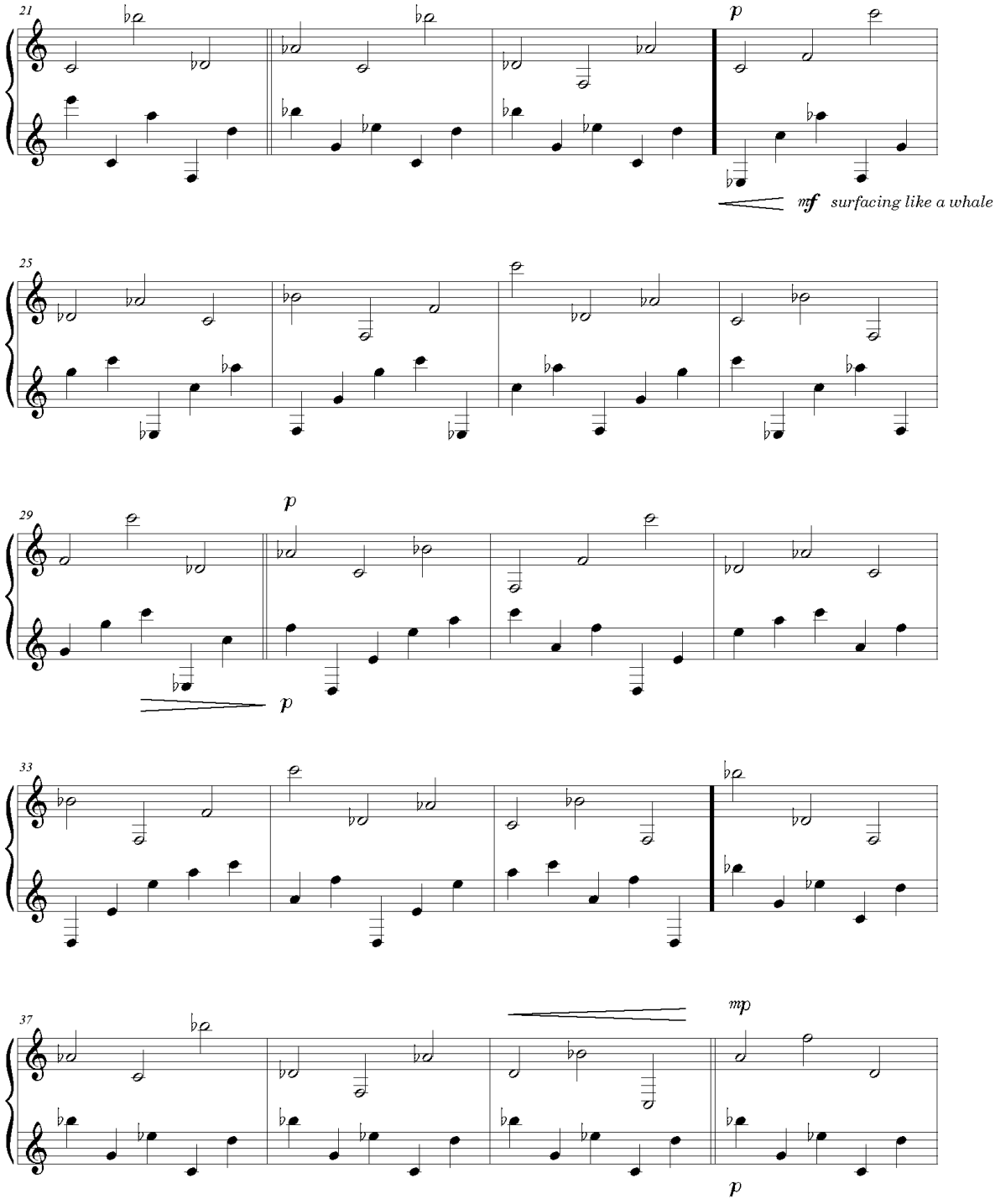

328
3/6 C5
331
335
339
U
♩. = 160
f bang!
350
1.
V
mf murmuring

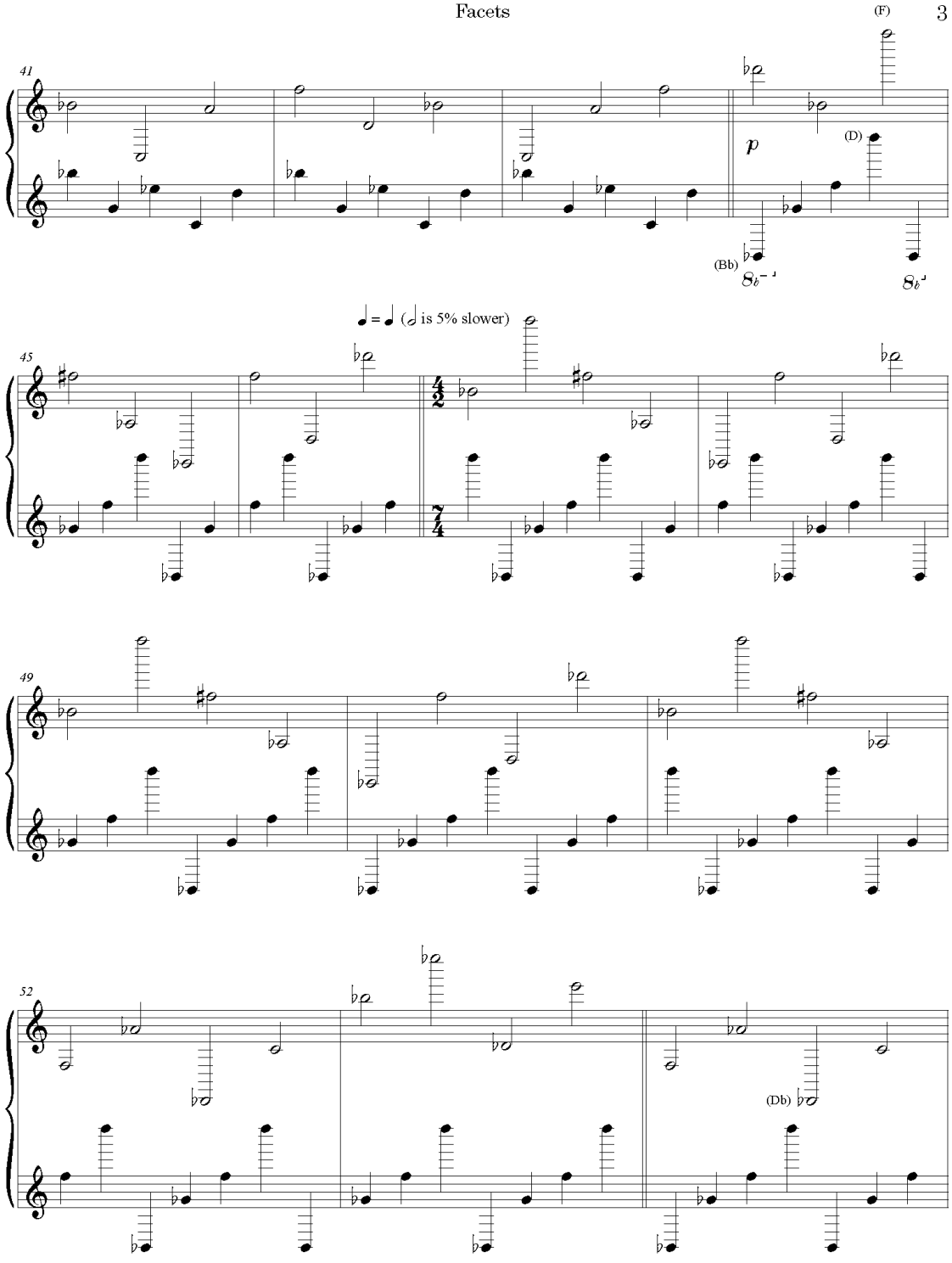

363
mp – p surfacing
368
1.
2.
mp – p ebbing
374
W
1.
f a march, or a wail
381
2.
p a nervous, urgent knock
387
X
mf downbeat...
393
1.
2.
mf
399
p
404
1.
2.
p
p

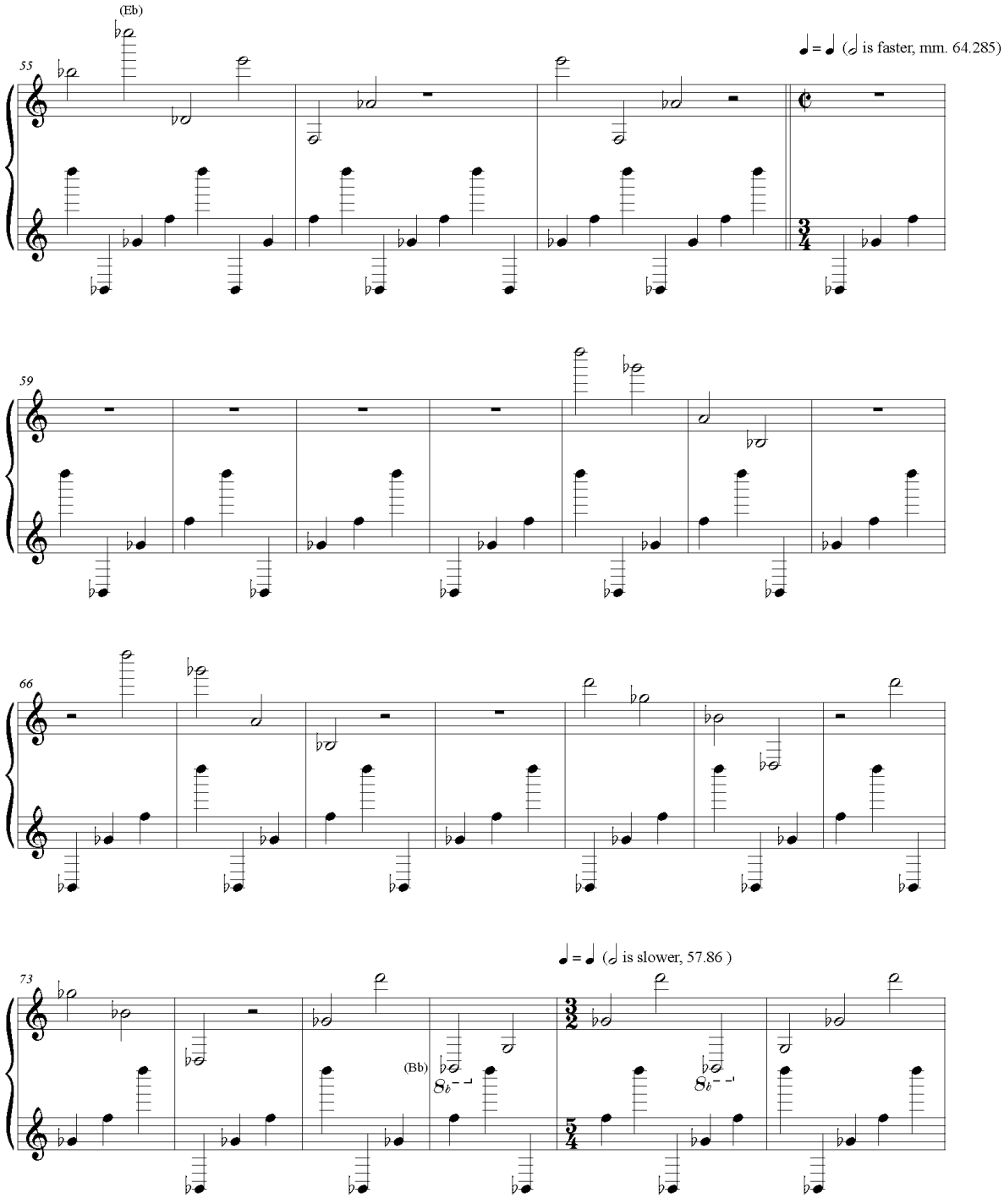

409
p
414
418
Y
f
423
428
432
436
2/6 C4
♩. = 33
f
Z
♩. = 160
ff
443
3x

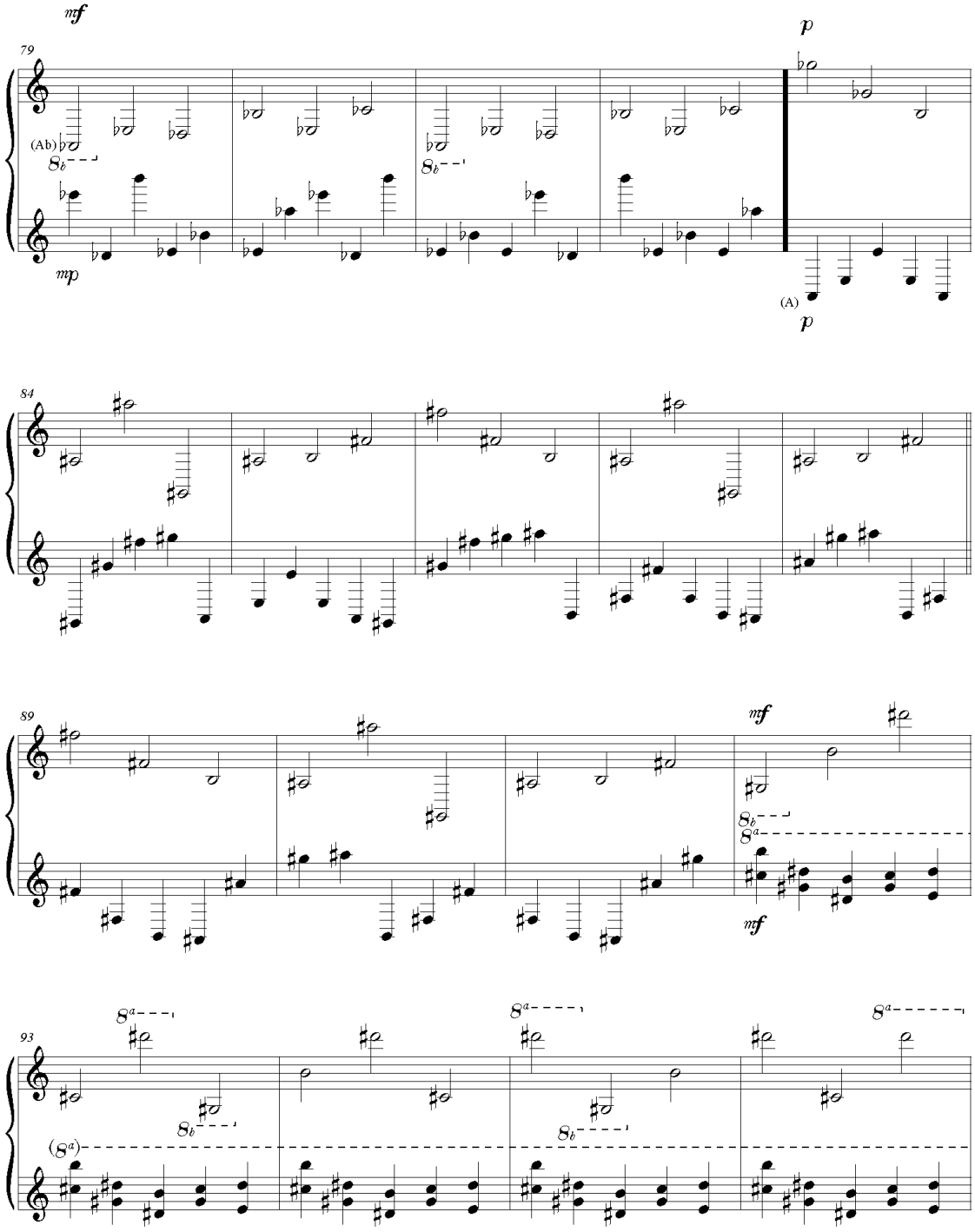
Facets (2013)

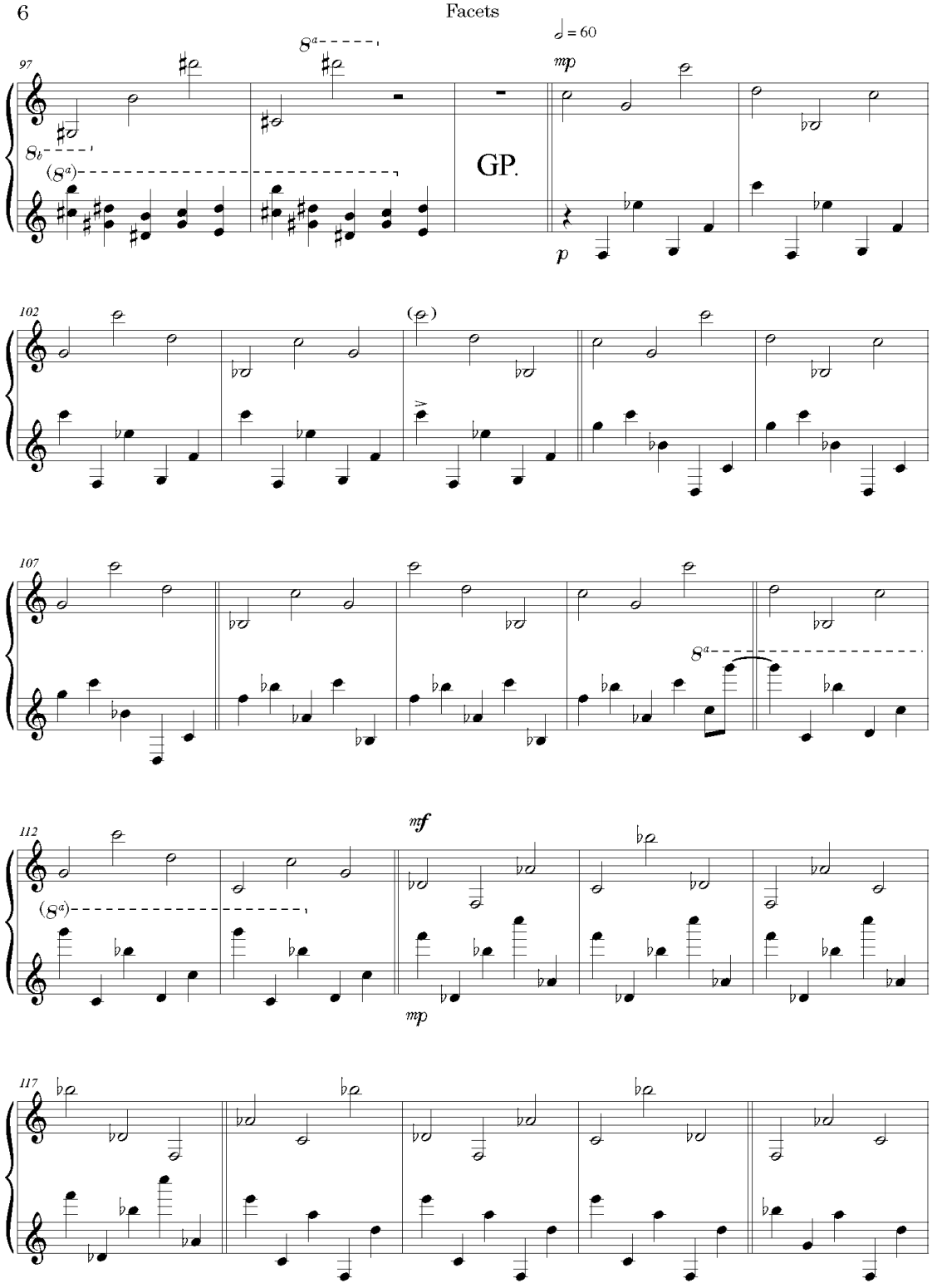

Facets
for Conor Hanick
Elliot Cole
mp
p
mf
mp
©2013 Elliot Cole / Long Echo Music (BMI)

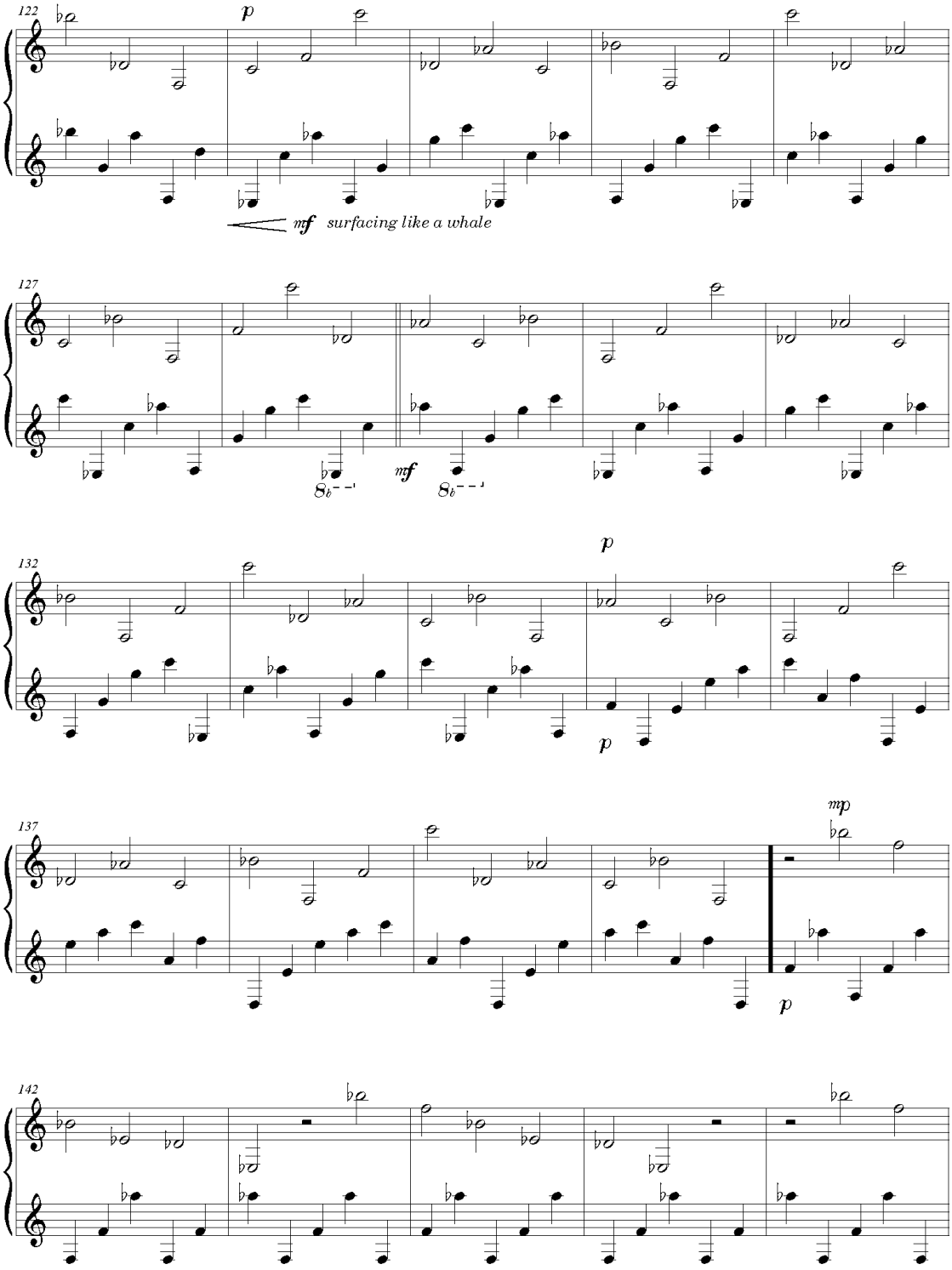

21
p
mf surfacing like a whale
25
29
p
p
33
37
mp
p

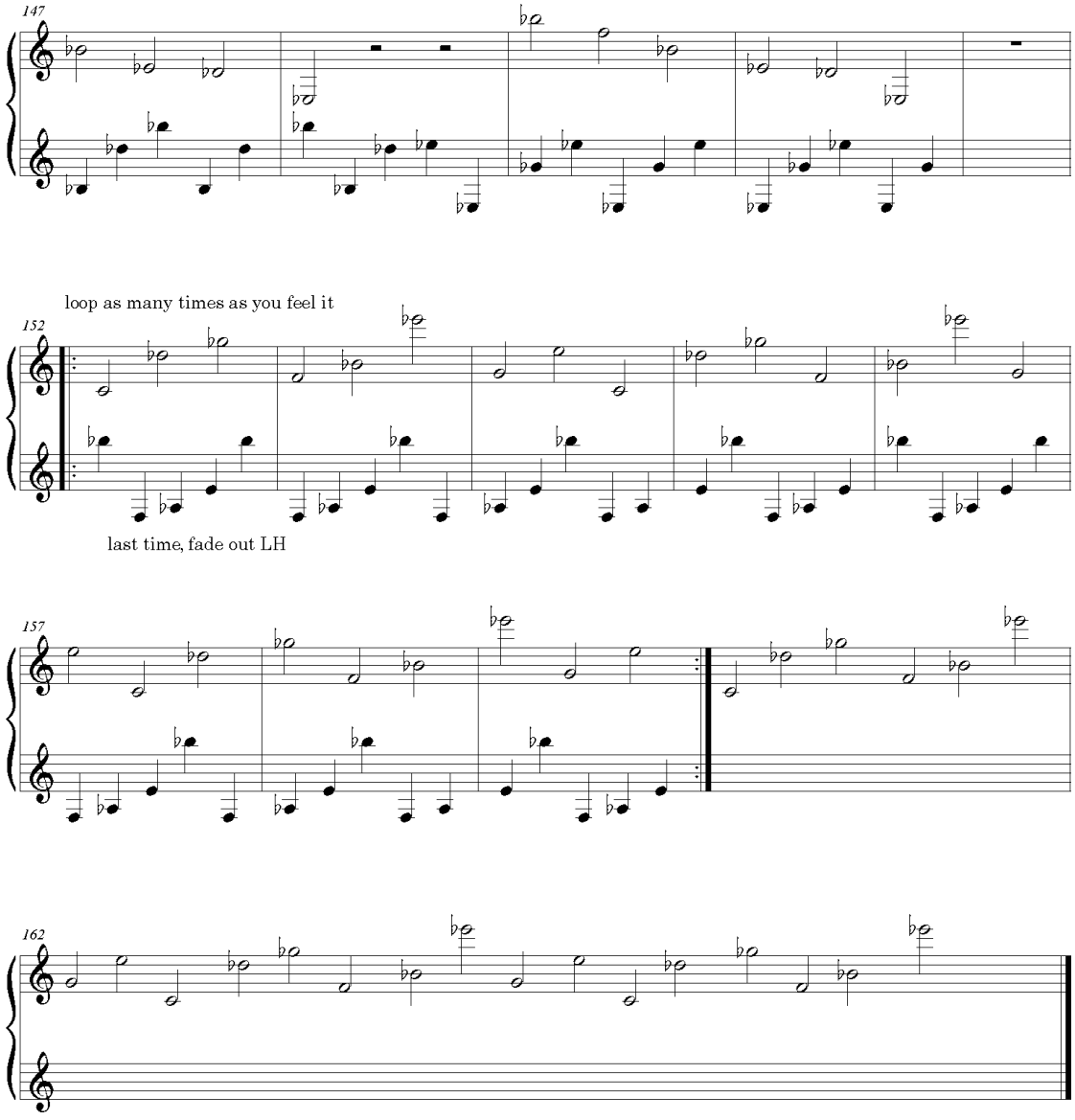

41
(F)
p
(D)
(Bb)
8b
8b
♩ = ♩ (𝅗𝅥 is 5% slower)
45
49
52
(Db)


(Eb)
♩ = ♩ (𝅗𝅥 is faster, mm. 64.285)
55
59
66
73
♩ = ♩ (𝅗𝅥 is slower, 57.86)
(Bb)
8b
8b


mf
79
(Ab)
8b
mp
p
(A)
p
84
89
mf
8b
8a
mf
93
8a
8b
(8a)


97
8a
8b
(8a)
GP.
mp
p
102
(𝅗𝅥)
107
8a
112
mf
(8a)
mp
117


122
p
mf surfacing like a whale
127
8b
mf
8b
132
p
p
137
mp
p
142

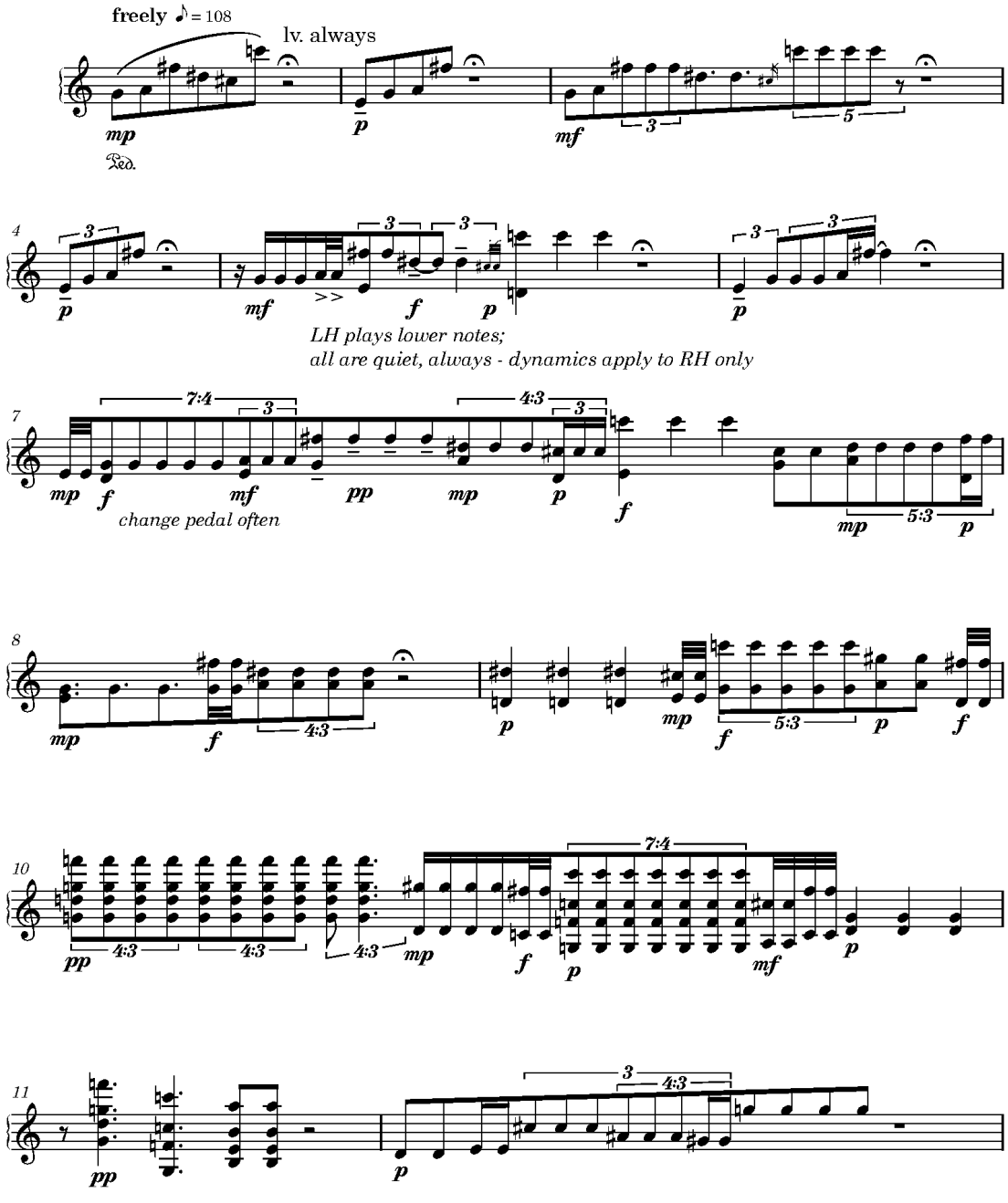

147
loop as many times as you feel it
152
last time, fade out LH
157
162

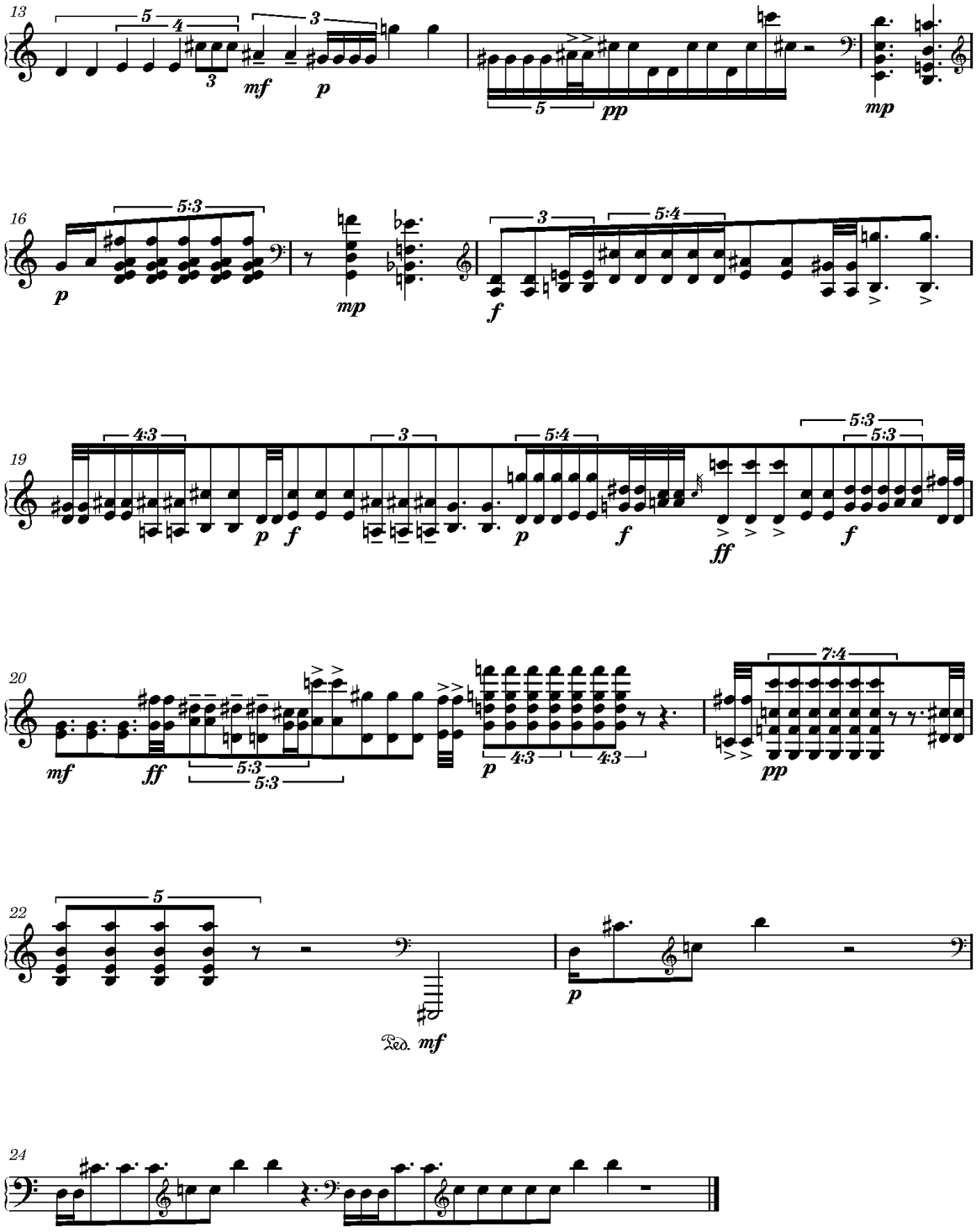
Two Nocturnes (2017)

corners of the night

for Andrea Lodge

Elliot Cole

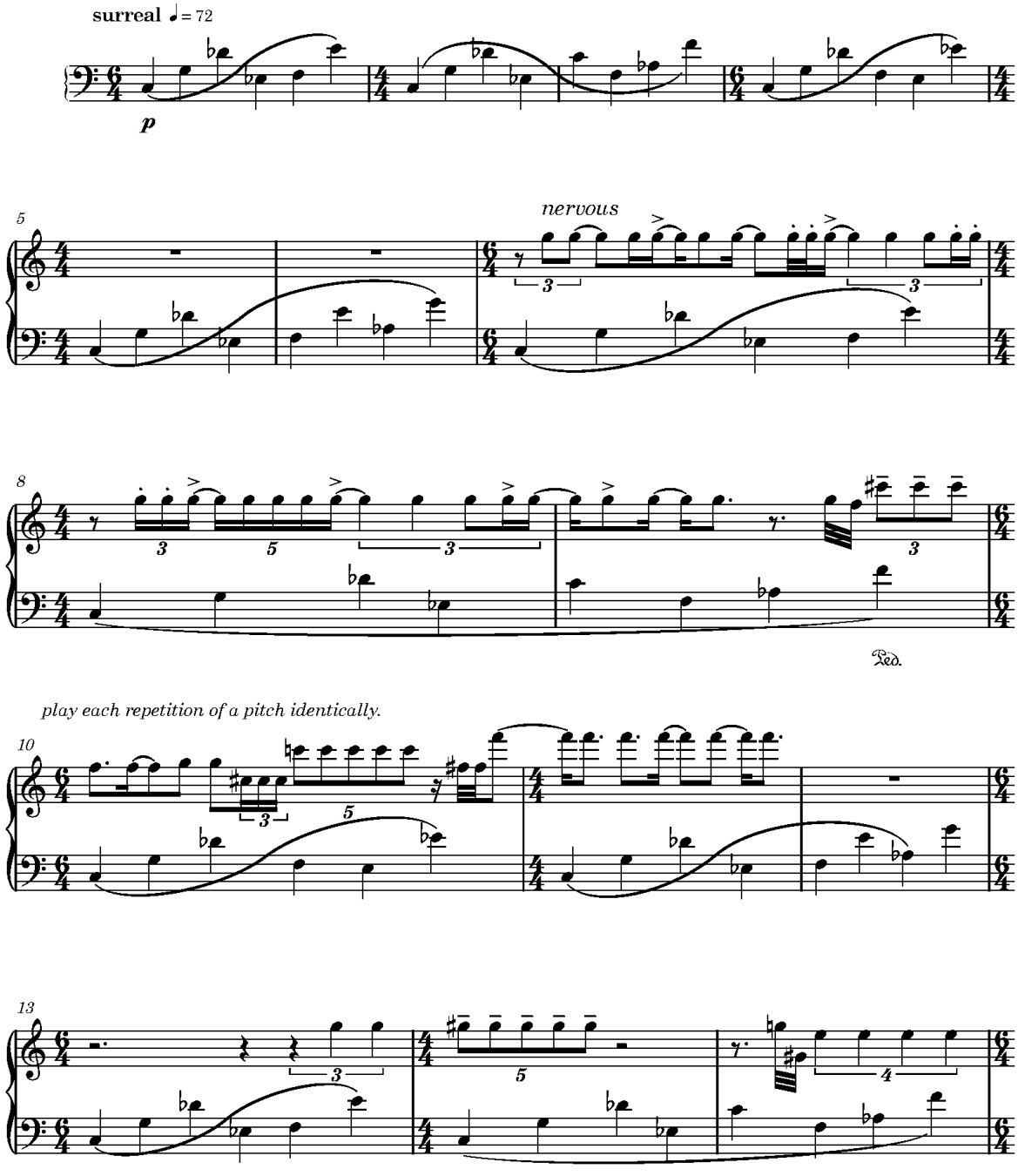


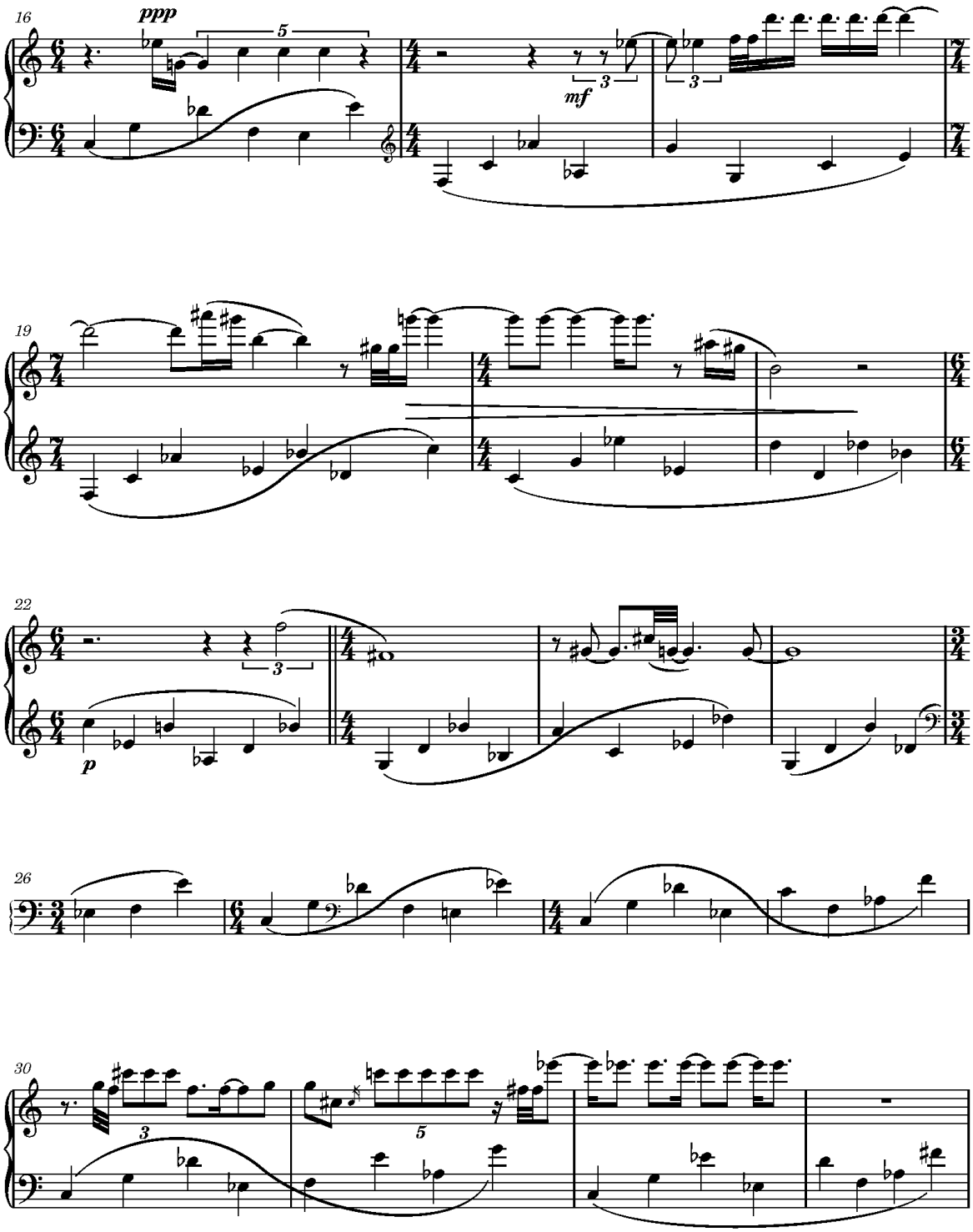

13
5
4
3
3
mf
p
5
pp
mp
16
5:3
p
mp
3
5:4
f
19
4:3
3
5:4
5:3
5:3
p
f
p
f
ff
f
20
mf
ff
5:3
5:3
p
4:3
4:3
7:4
pp
22
5
Ped. mf
p
24

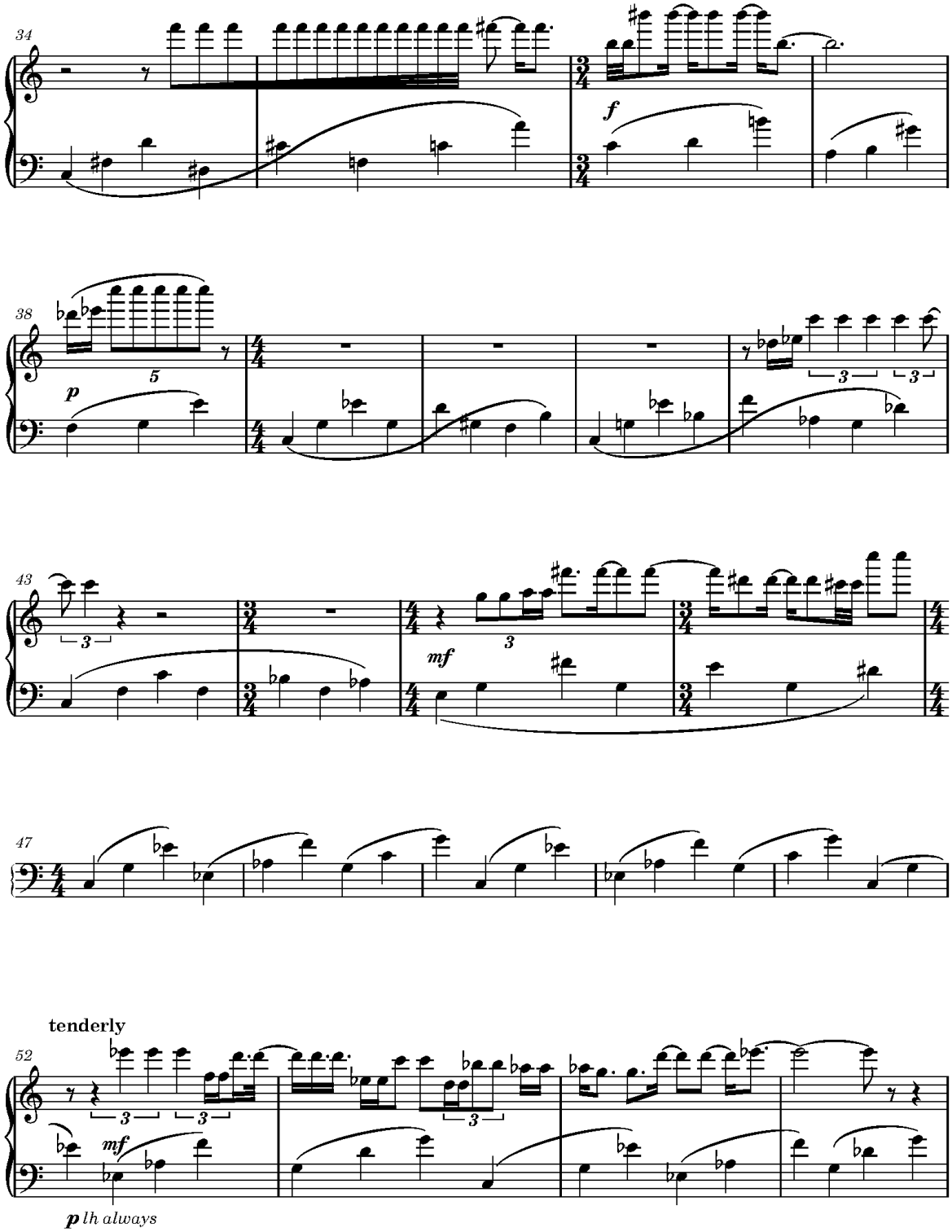
even the night has corners

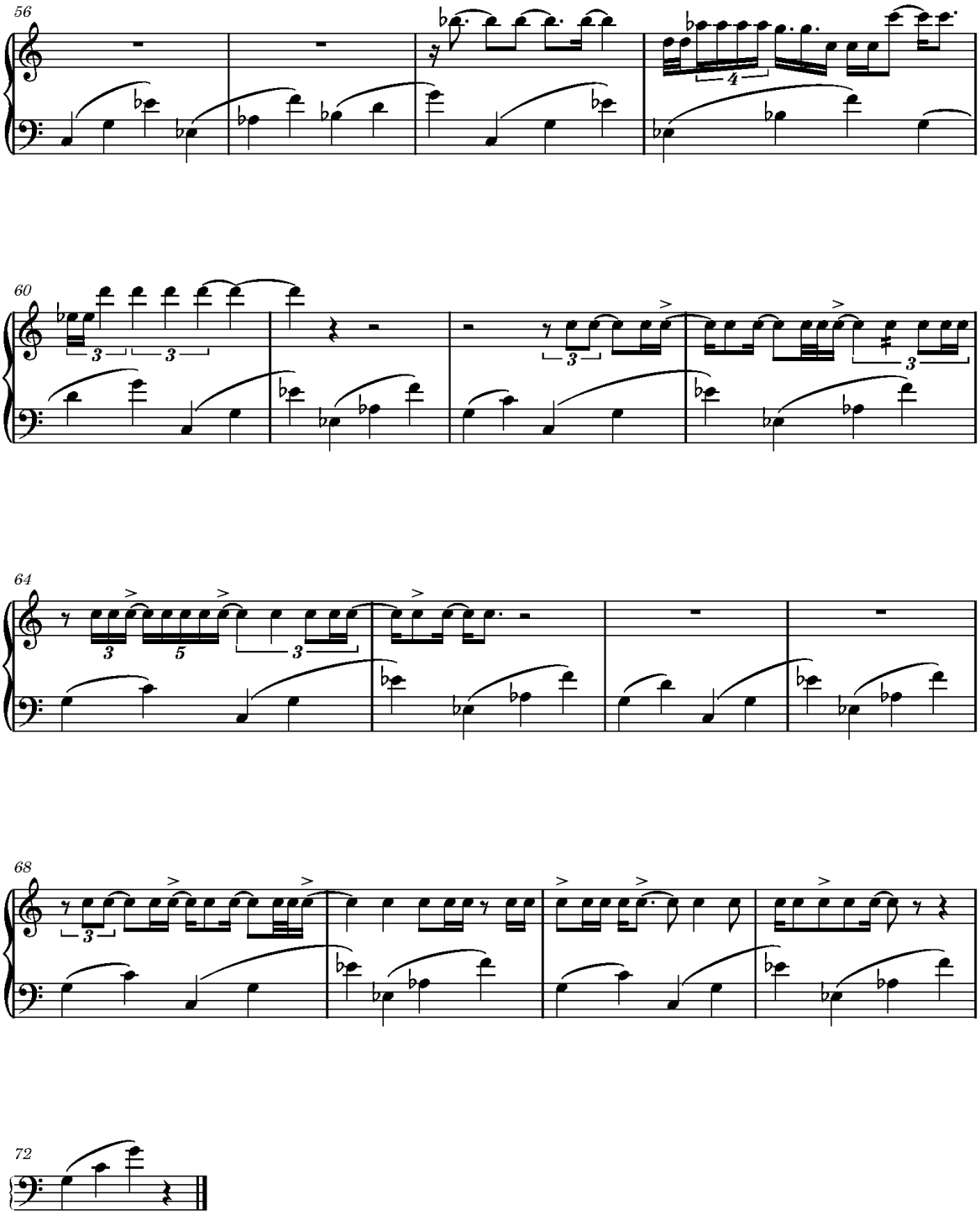
for Andrea Lodge

Elliot Cole

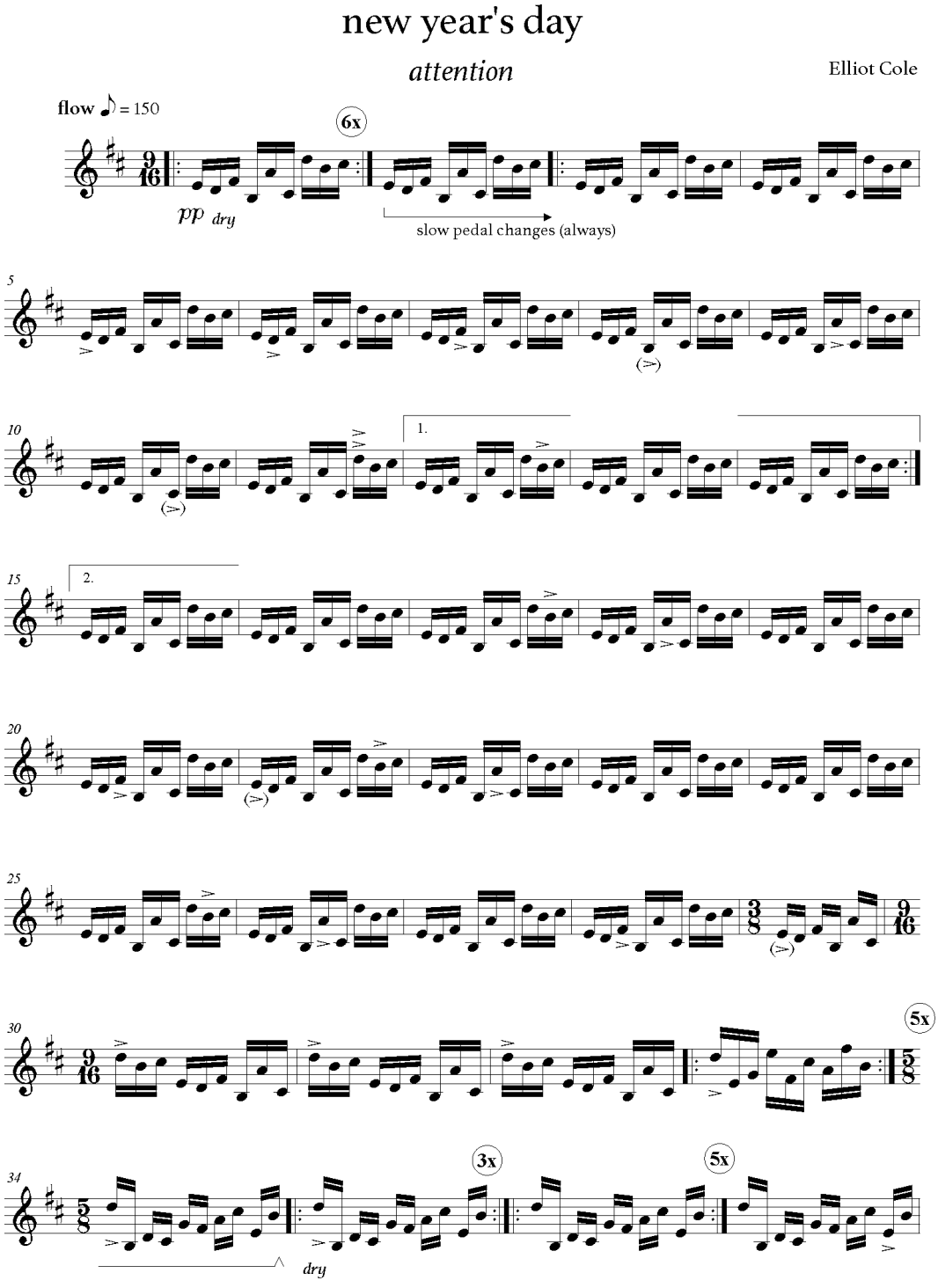


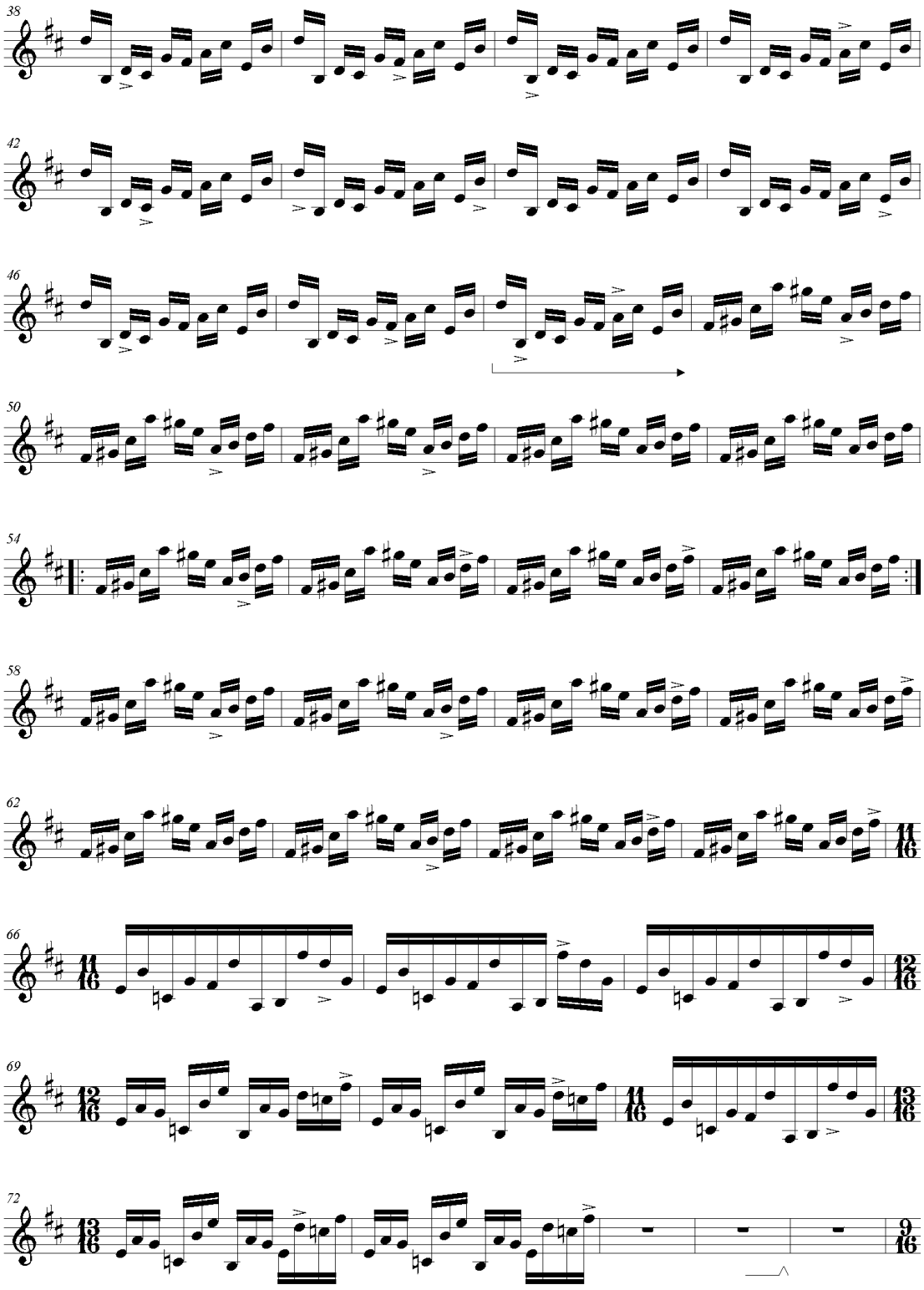

16
ppp
5
mf
3
3
19
22
3
p
26
30
3
5

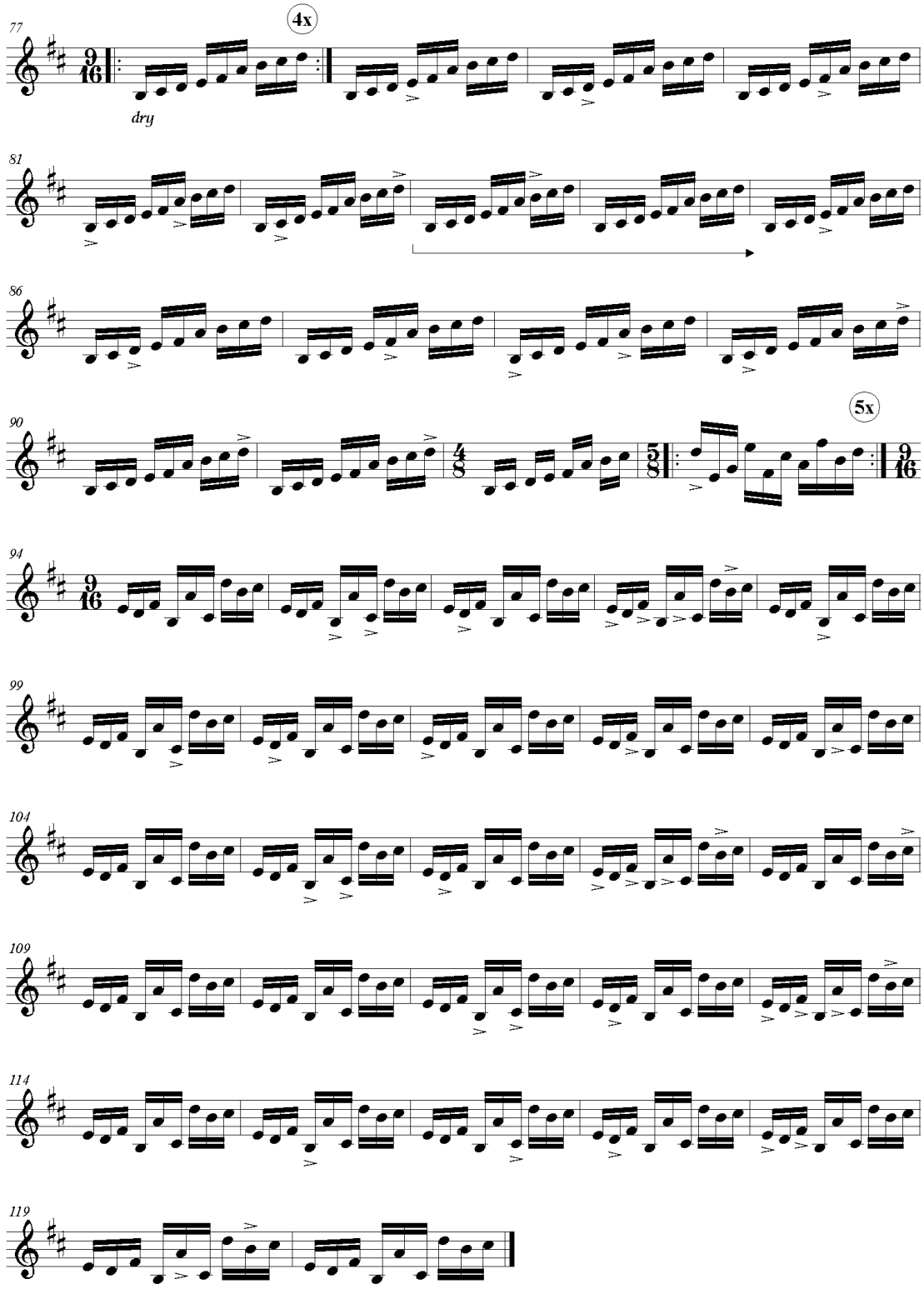

34
f
38
p
5
3
3
43
3
3
mf
47
52
tenderly
3
3
3
mf
p lh always


56
60
64
68
72

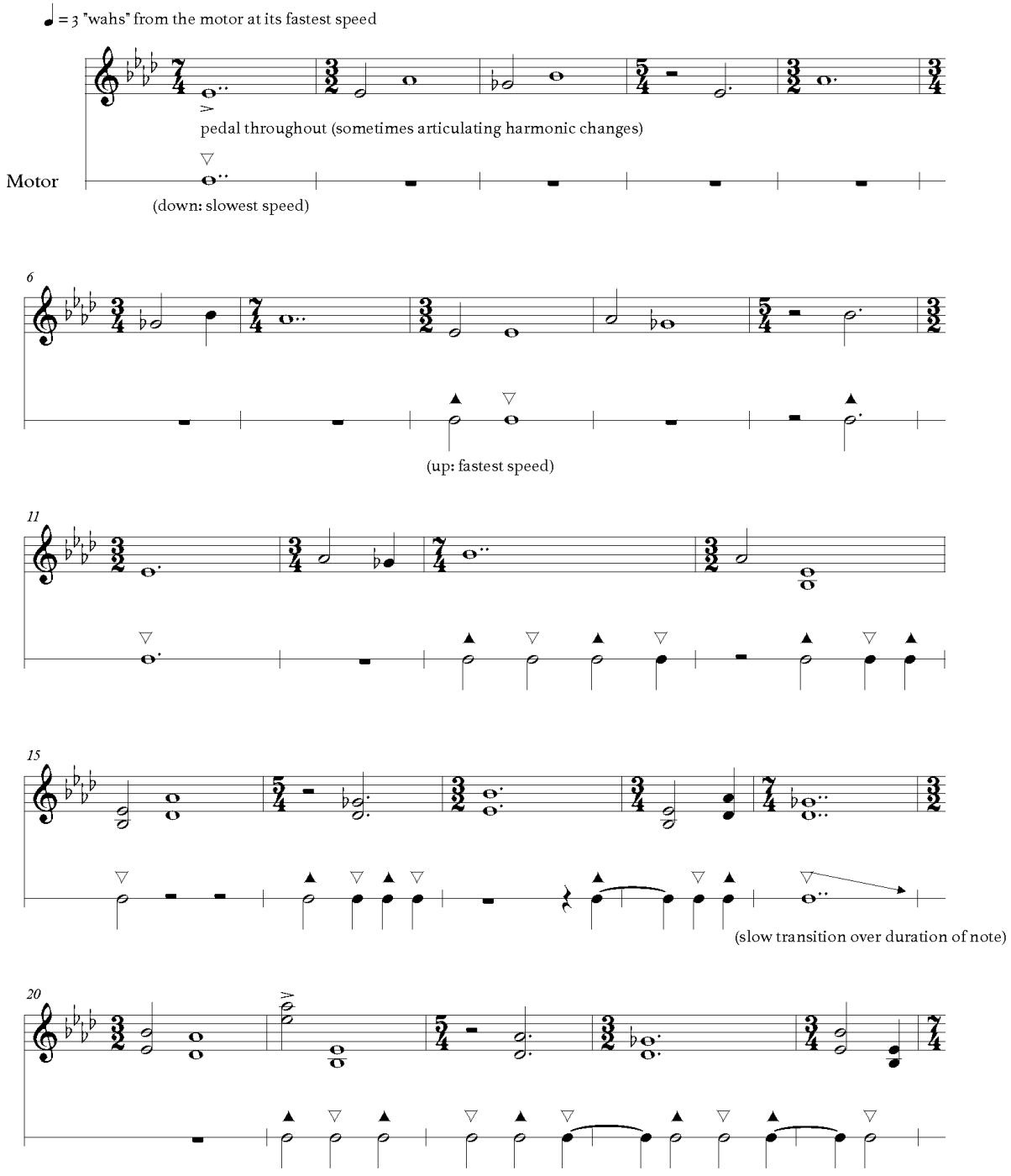
new year's day (2015)

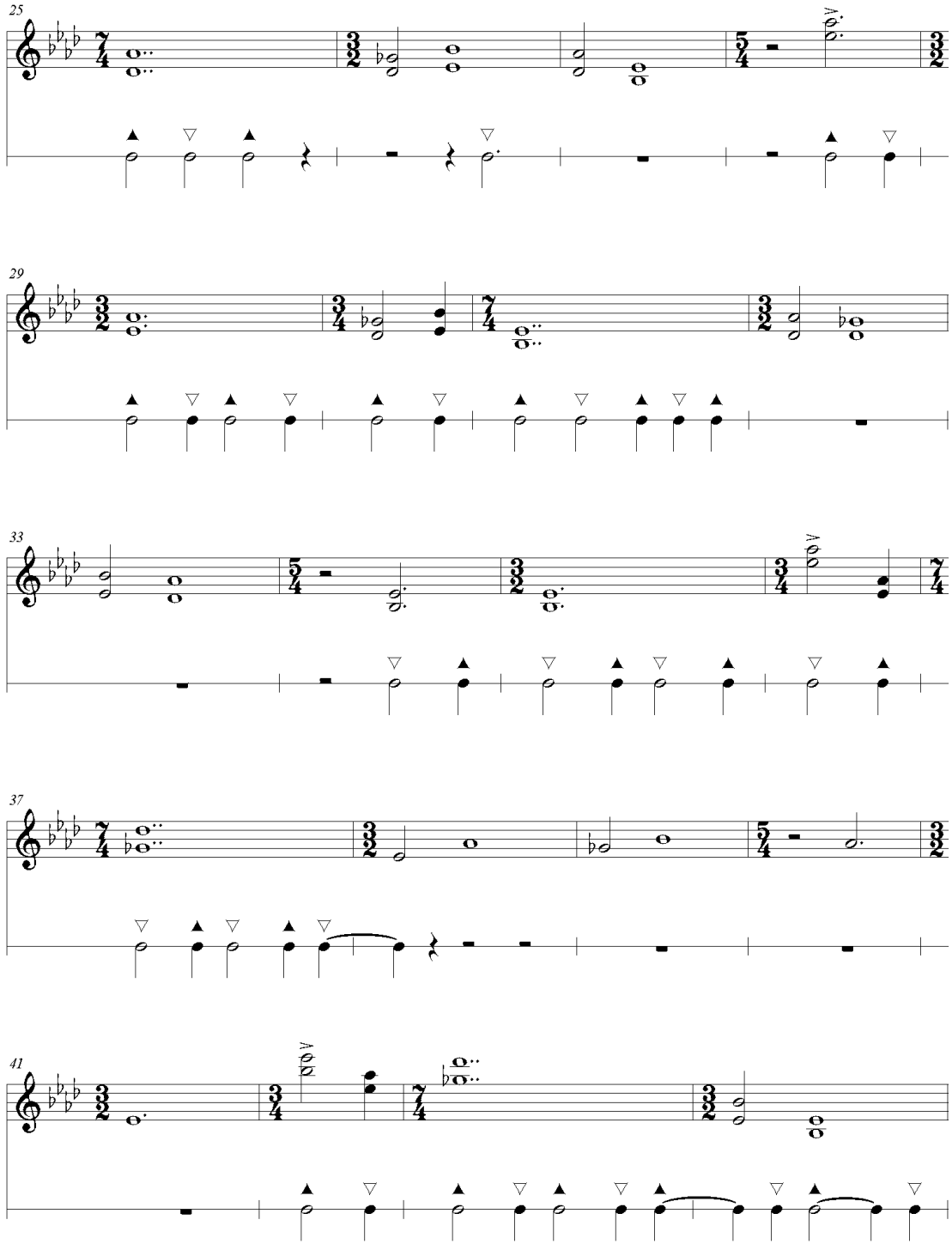


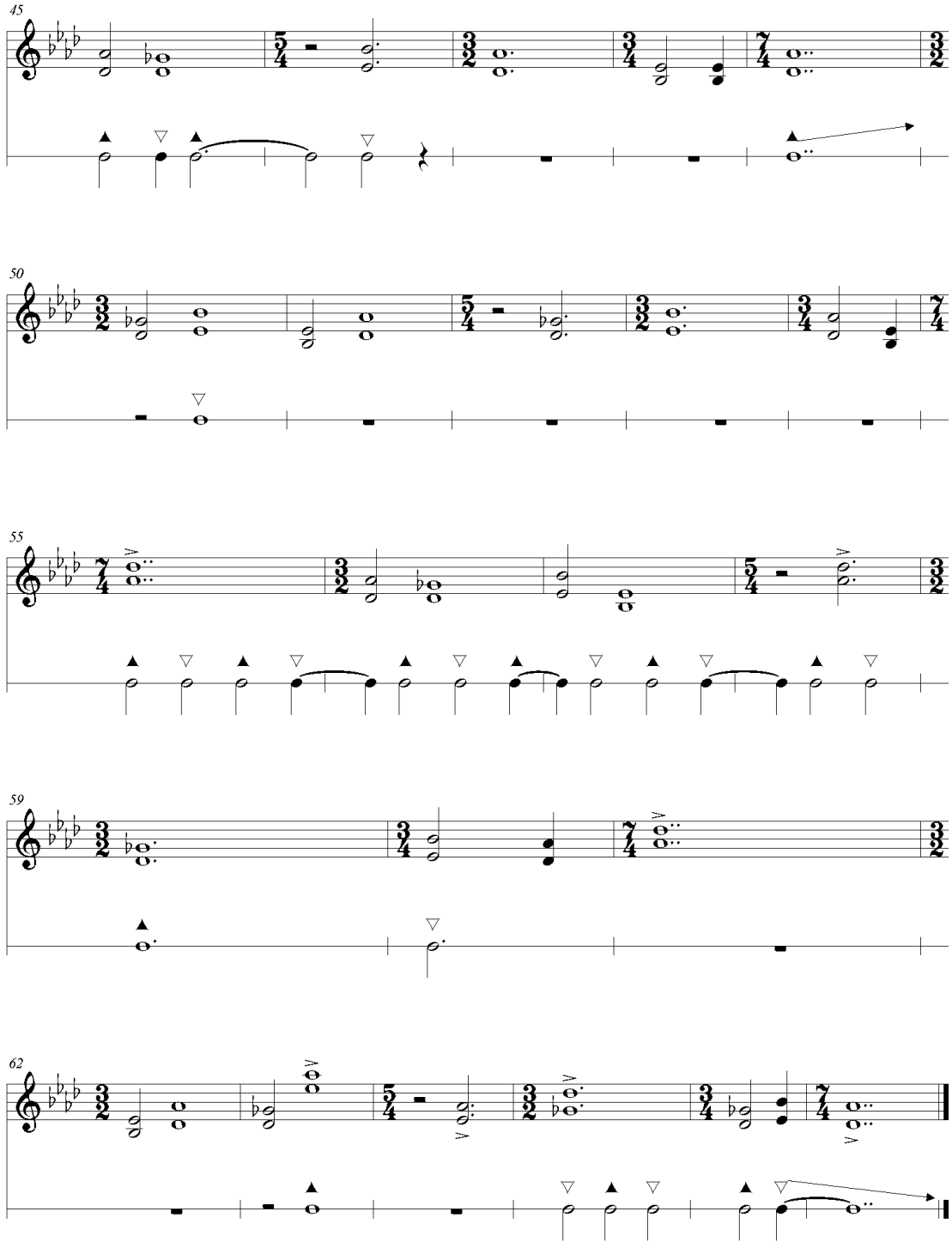

35
42
46
50
54
58
62
66
69
72


77
4x
dry
81
86
90
5x
94
99
104
109
114
119

vibration

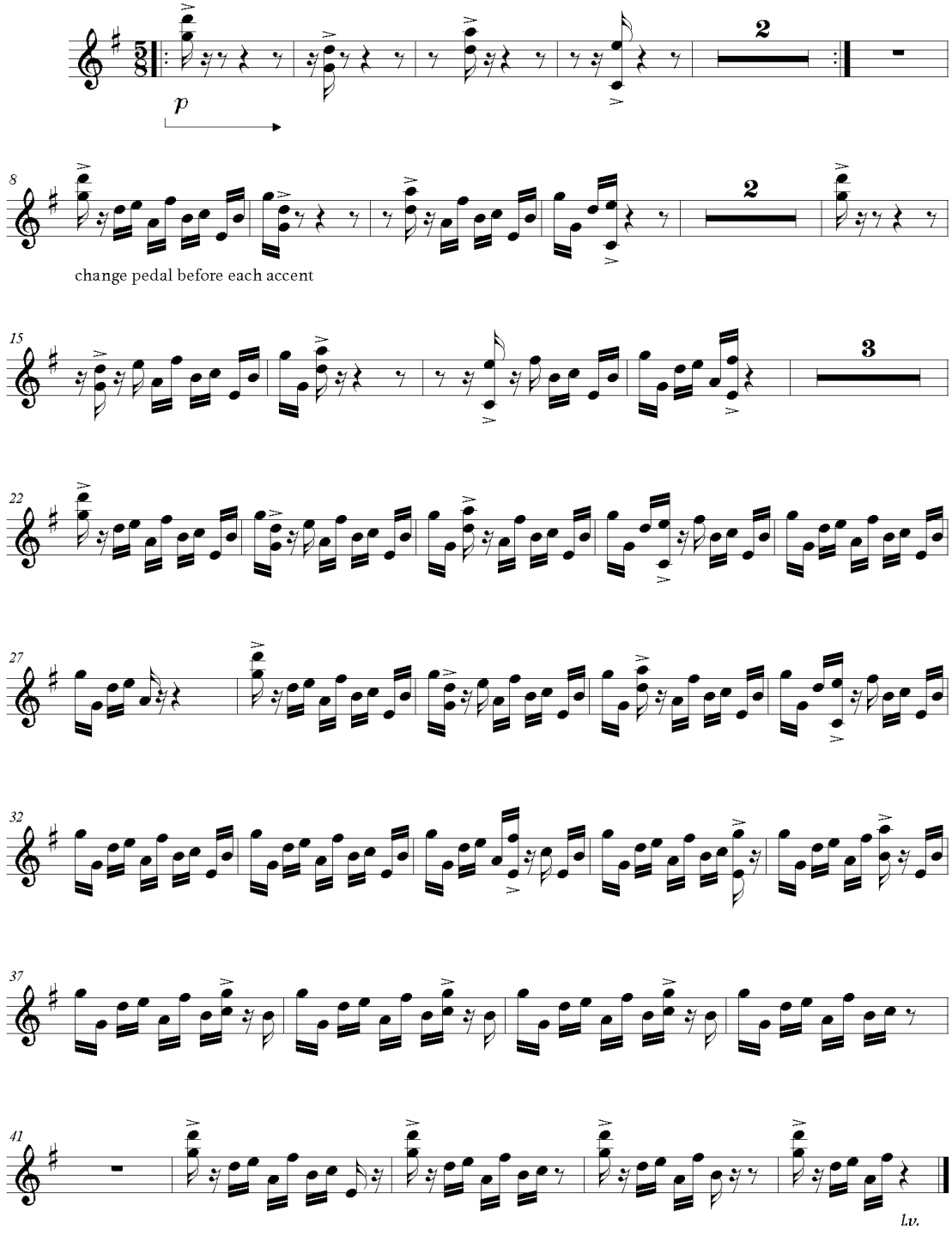

♩ = 3 "wahs" from the motor at its fastest speed
pedal throughout (sometimes articulating harmonic changes)
Motor
(down: slowest speed)
6
(up: fastest speed)
11
15
(slow transition over duration of note)
20


25
29
33
37
41


45
50
55
59
62

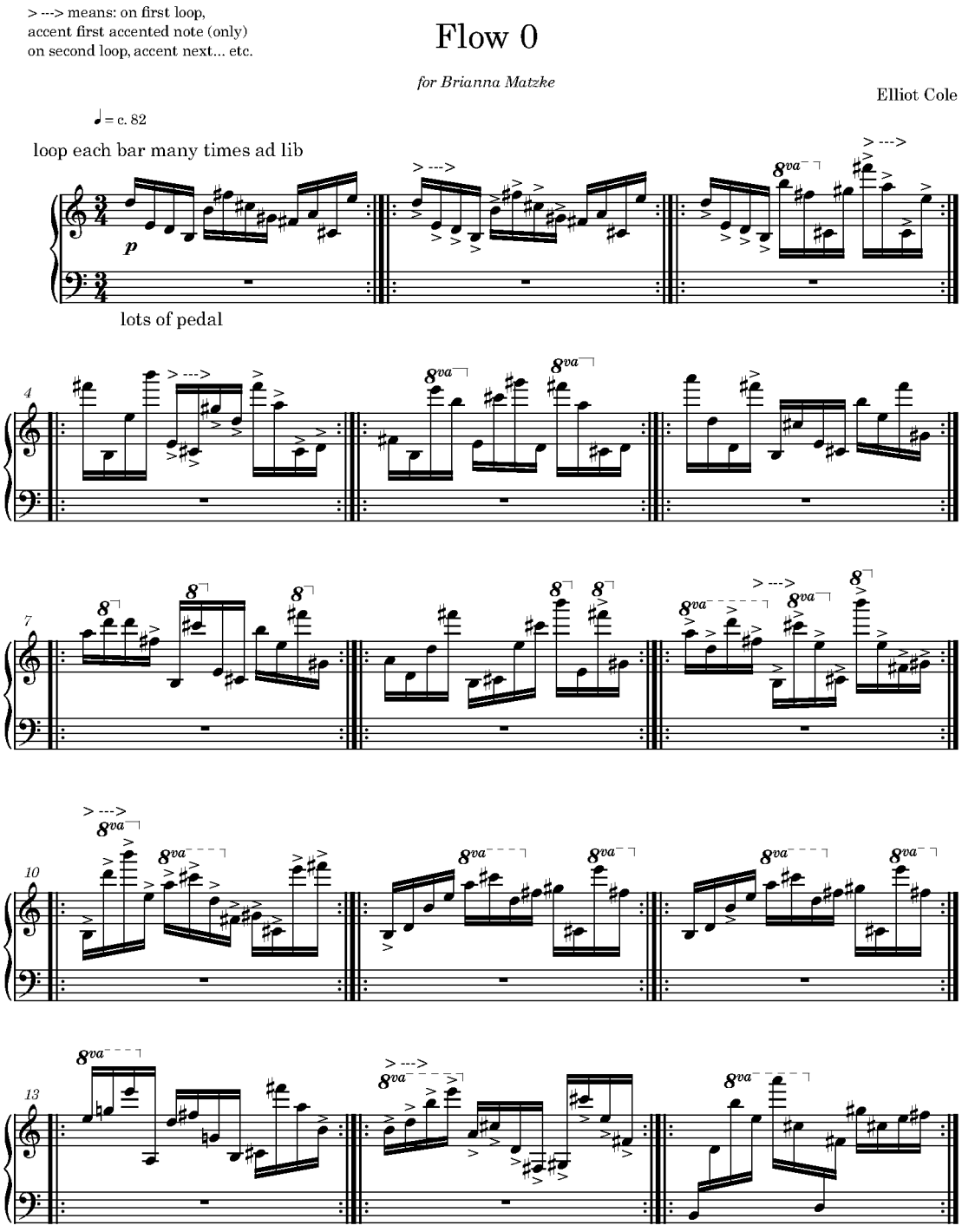
suspension

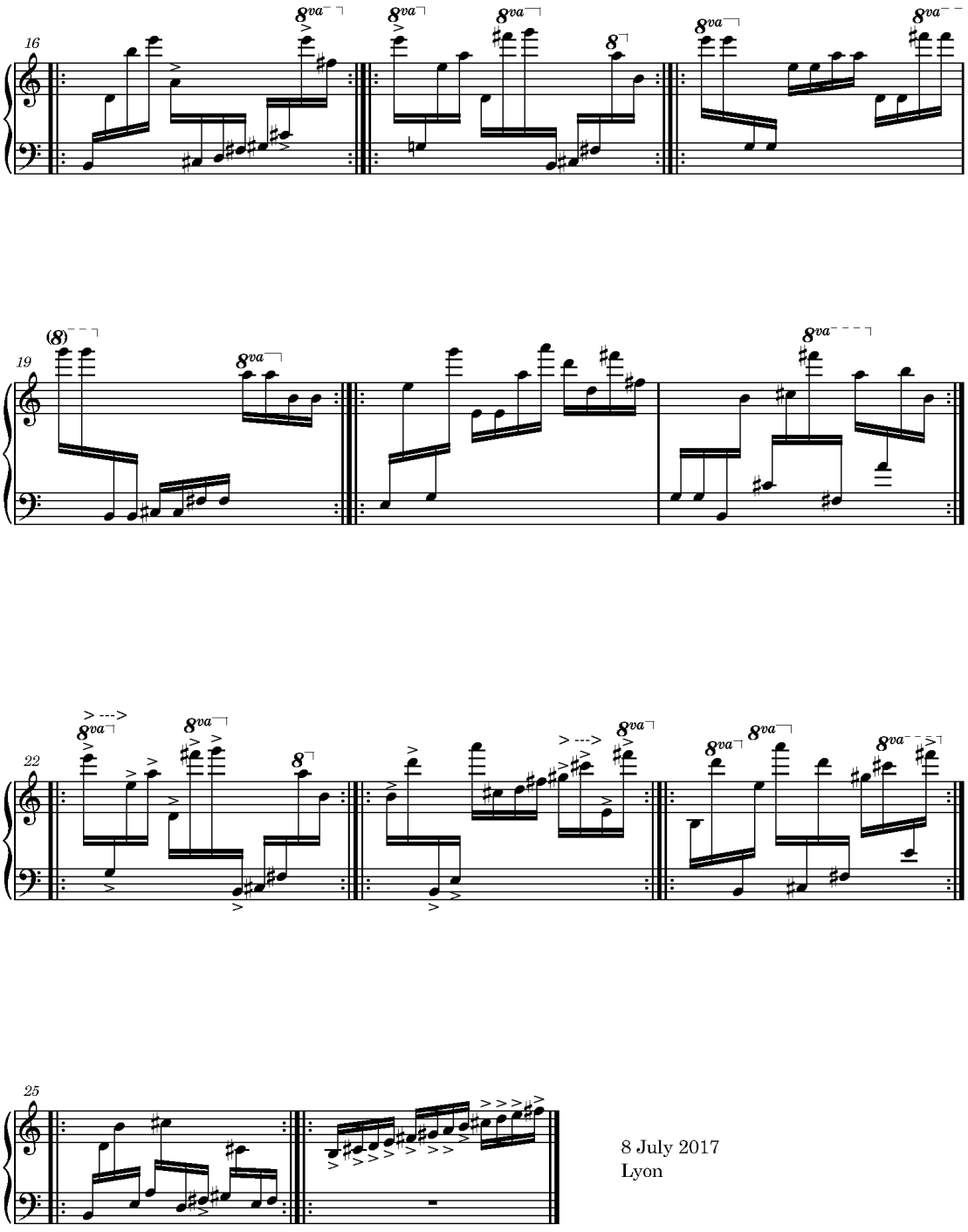
joyfully ♪ = 144



new year's eve 2015 / jersey city

Flows (2017)


> ---> means: on first loop,
accent first accented note (only)
on second loop, accent next... etc.
Flow 0
for Brianna Matzke
Elliot Cole
♩= c. 82
loop each bar many times ad lib
p
lots of pedal

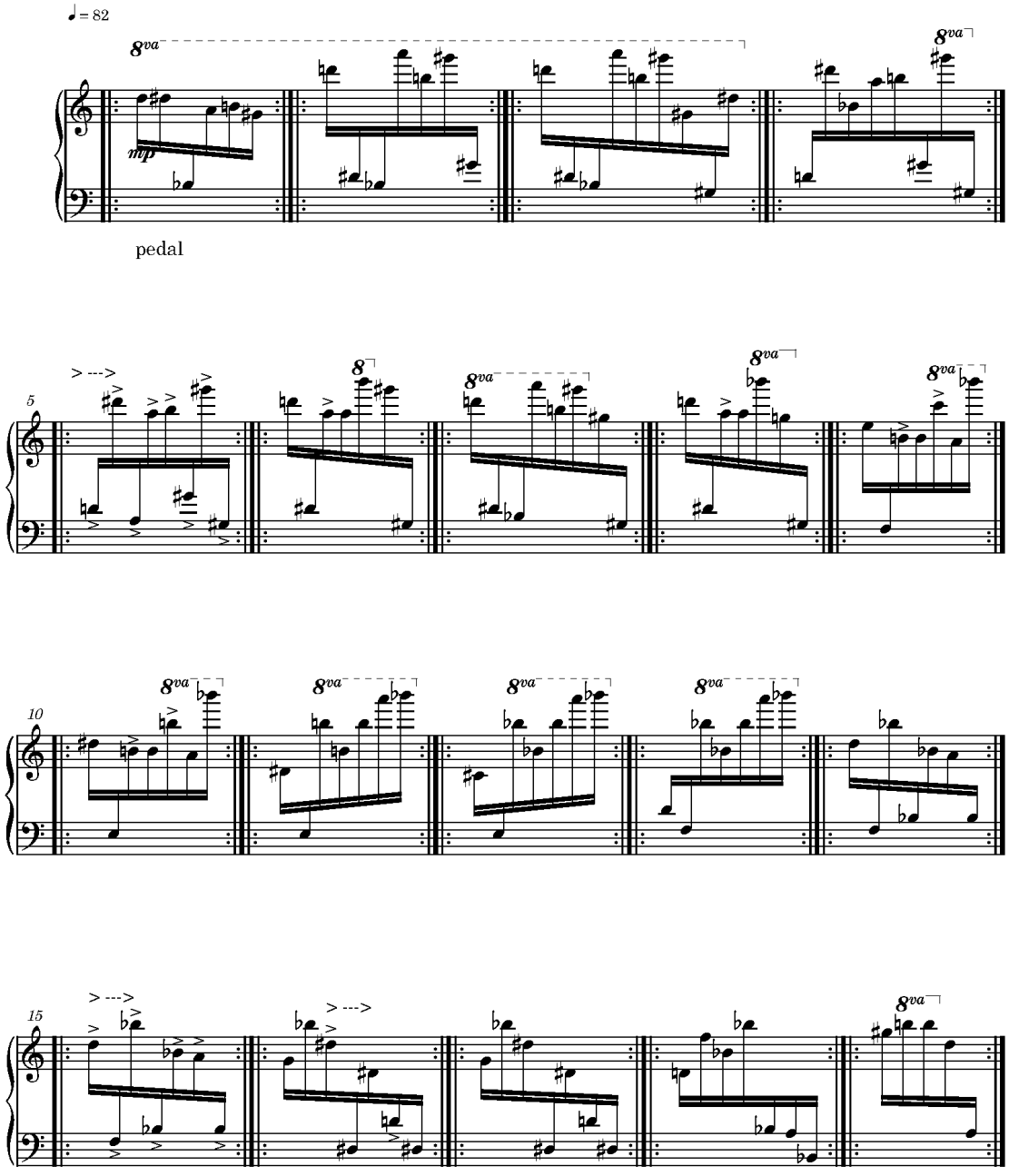


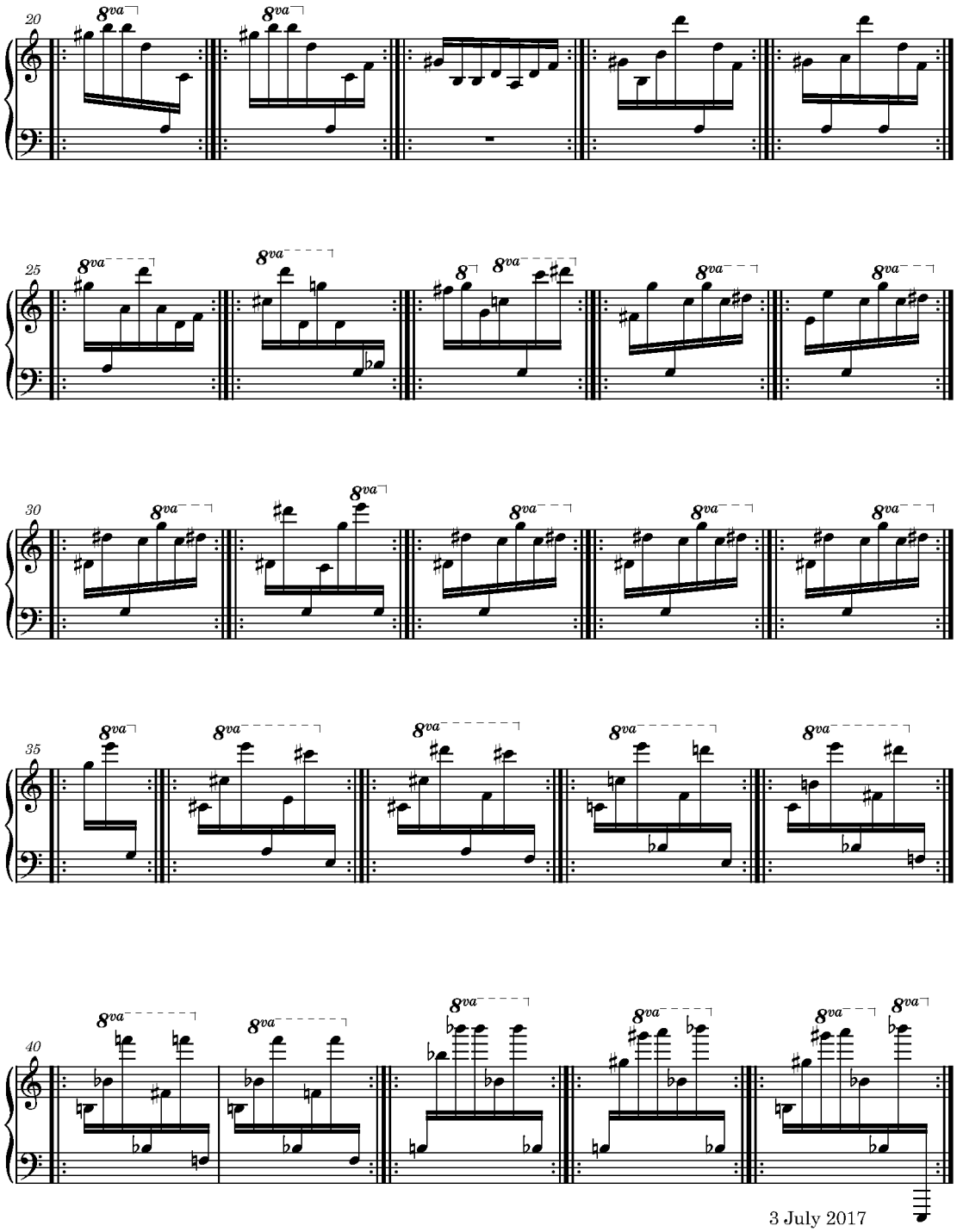
8 July 2017
Lyon

Flow 1

for Brianna Matzke

Elliot Cole



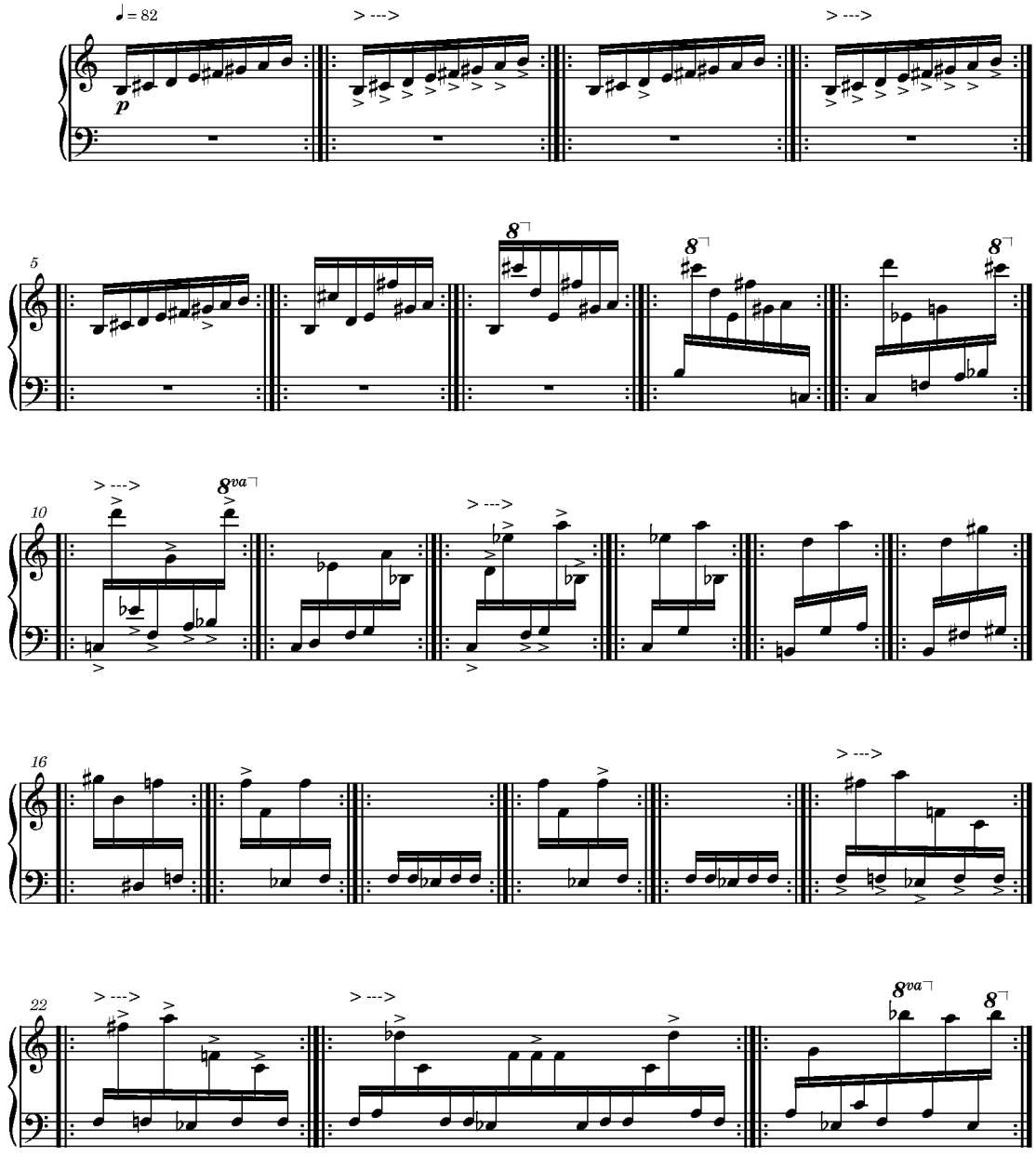


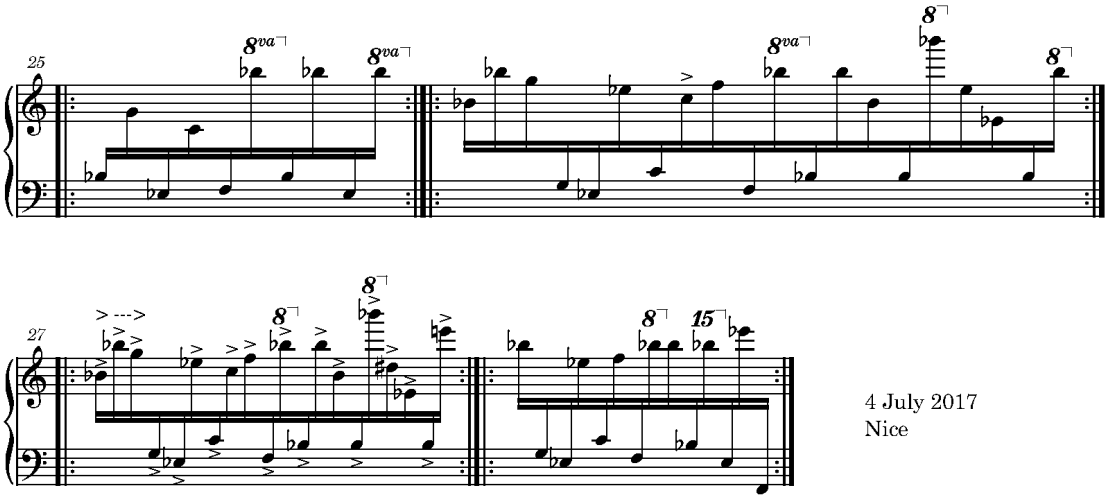
3 July 2017
Nice

Flow 2

for Brianna Matzke

Elliot Cole



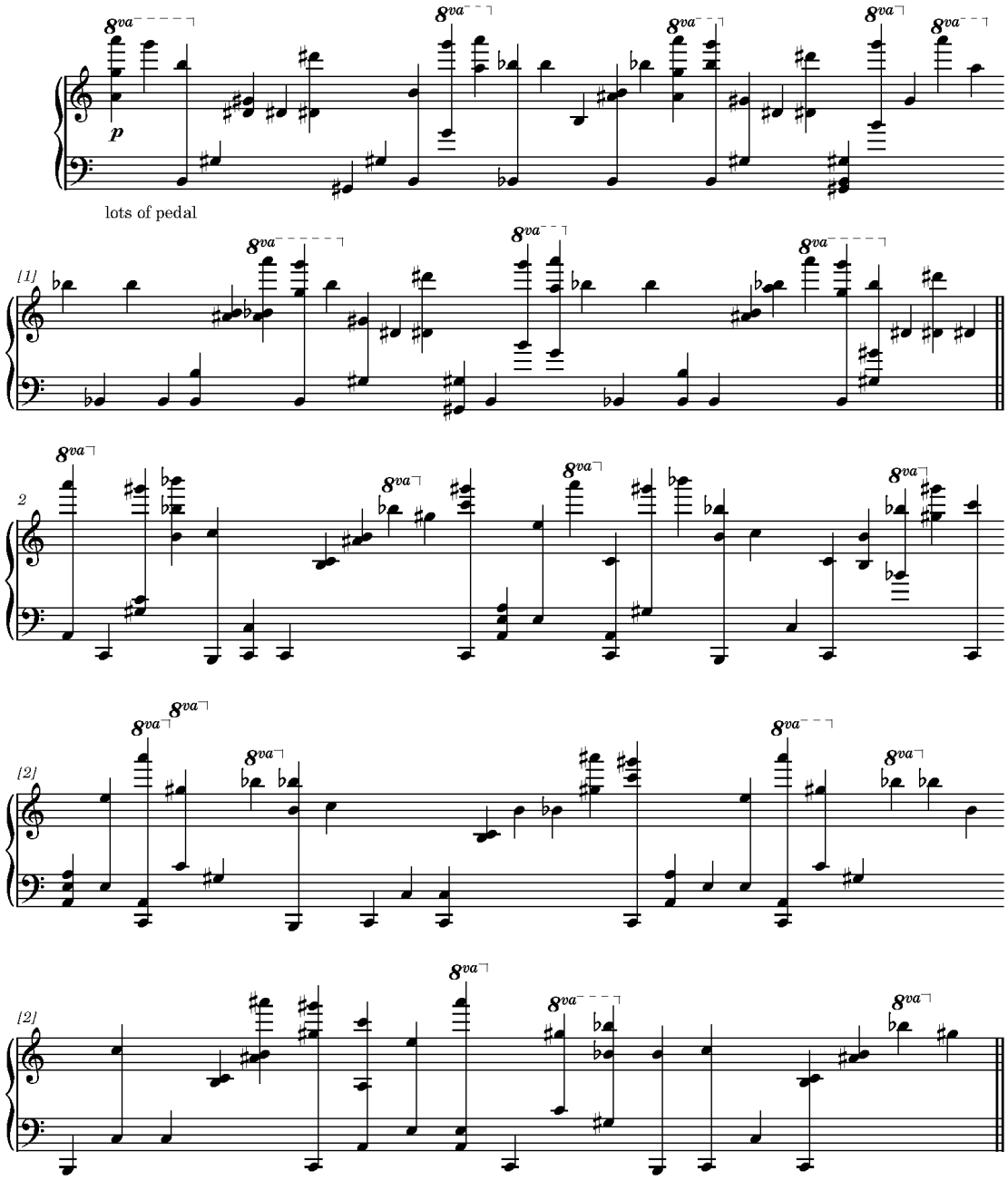


4 July 2017
Nice

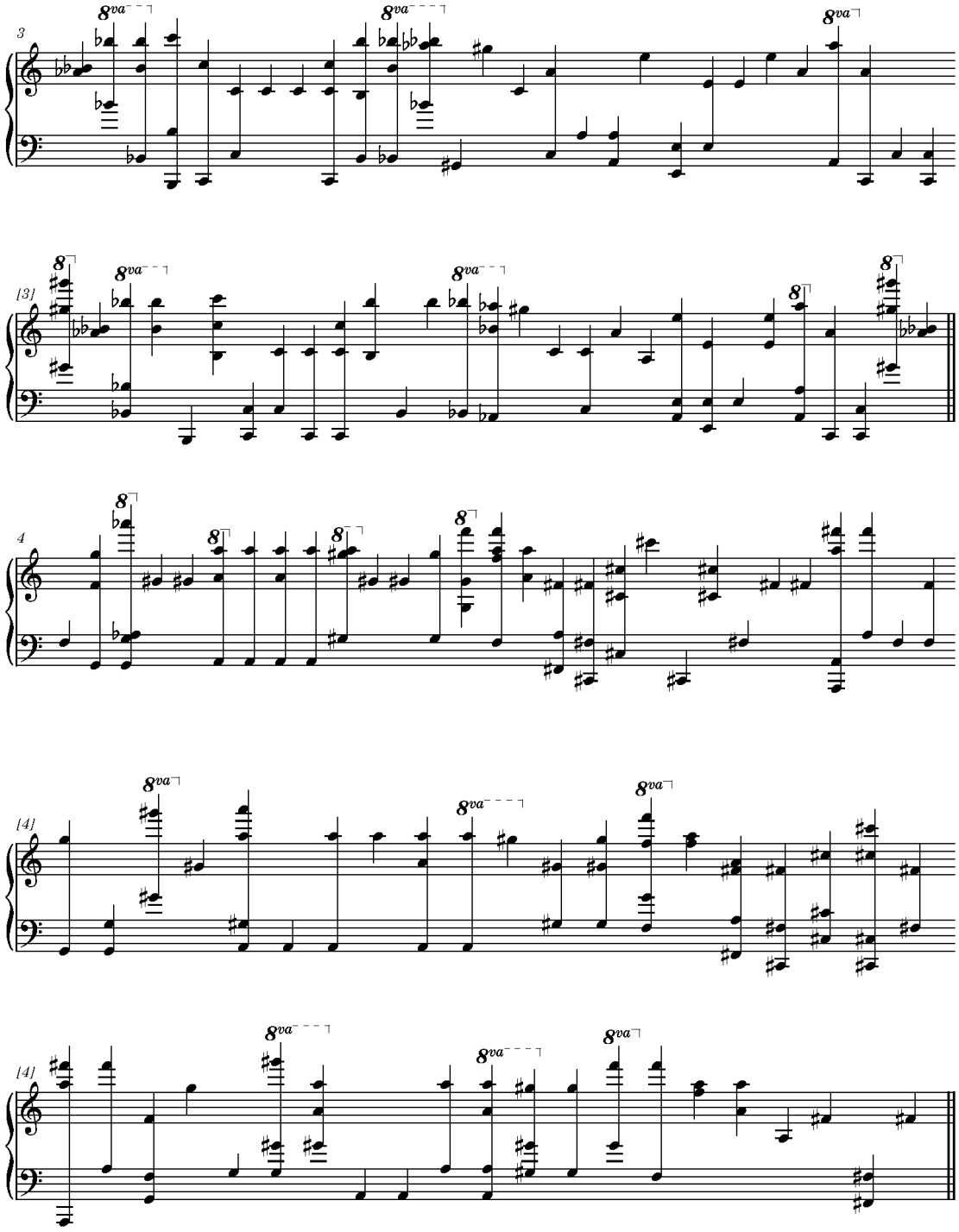
Flow 3

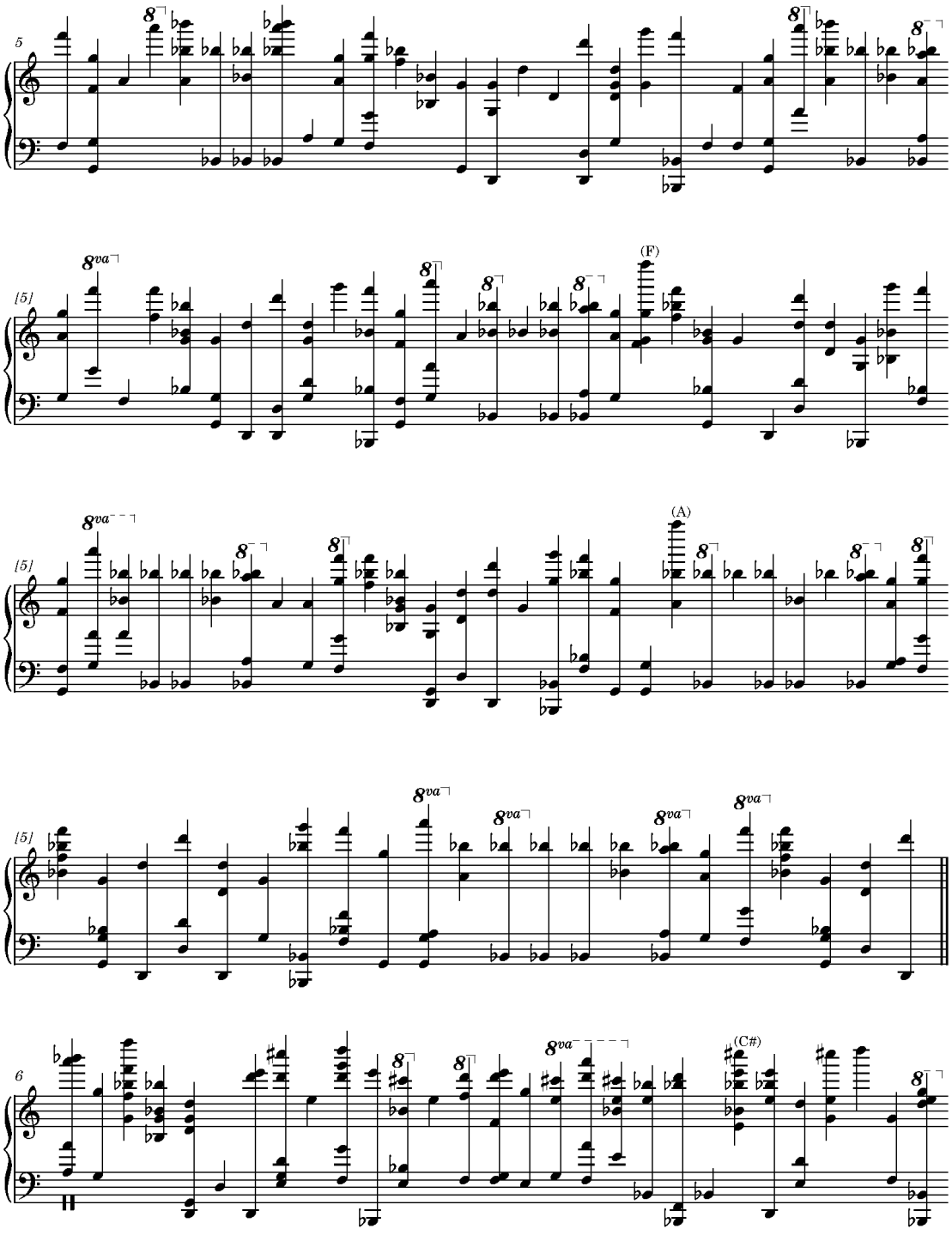
for Brianna Matzke

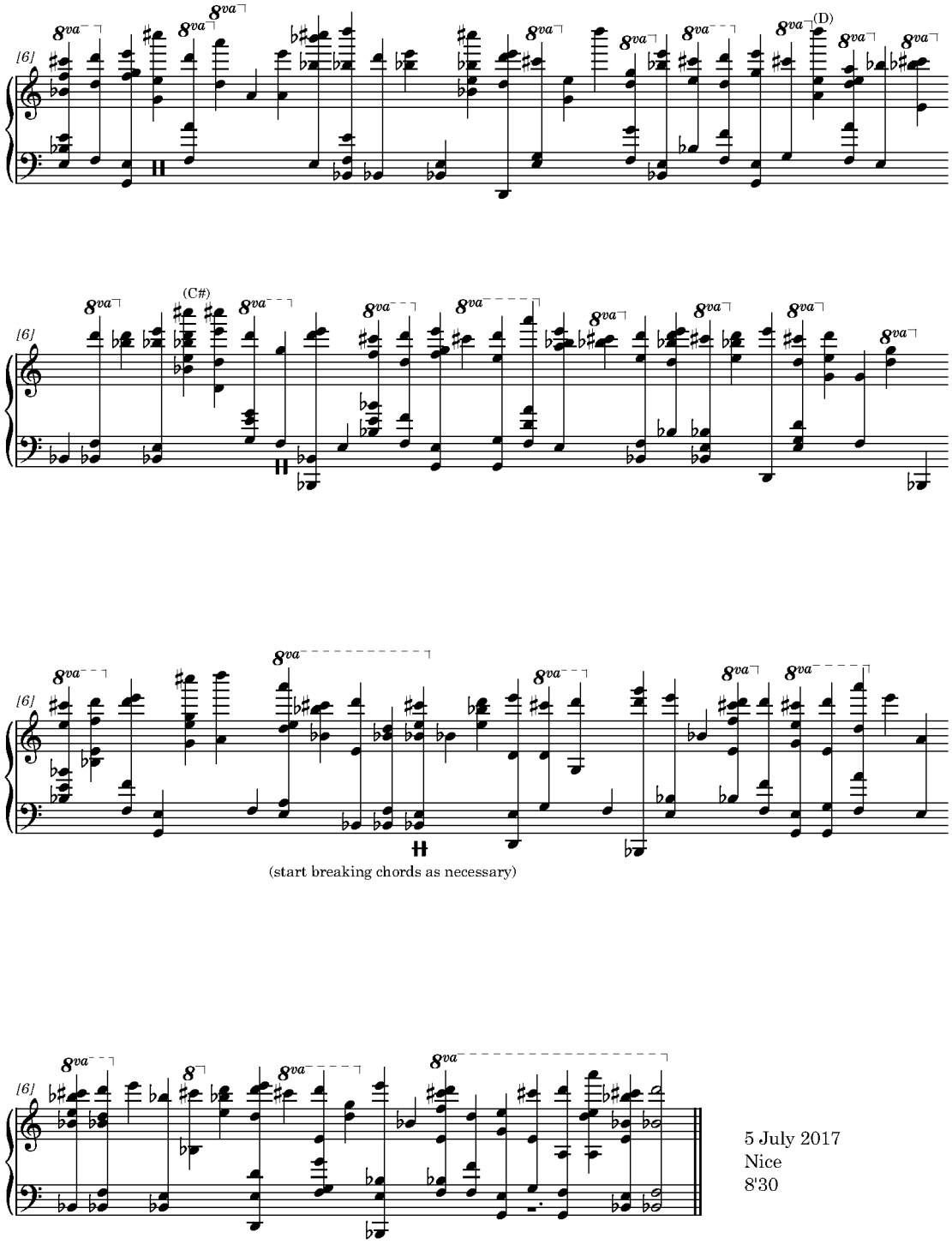
Elliot Cole



(accidentals apply to one note only. 8va does not apply to notes beamed to lower staff.)


3
8va
8va
8va
[3]
8va
8va
8
4
8
8
8
[4]
8va
8va
8va
[4]
8va
8va
8va


5
[5]
(F)
[5]
(A)
[5]
6
(C#)

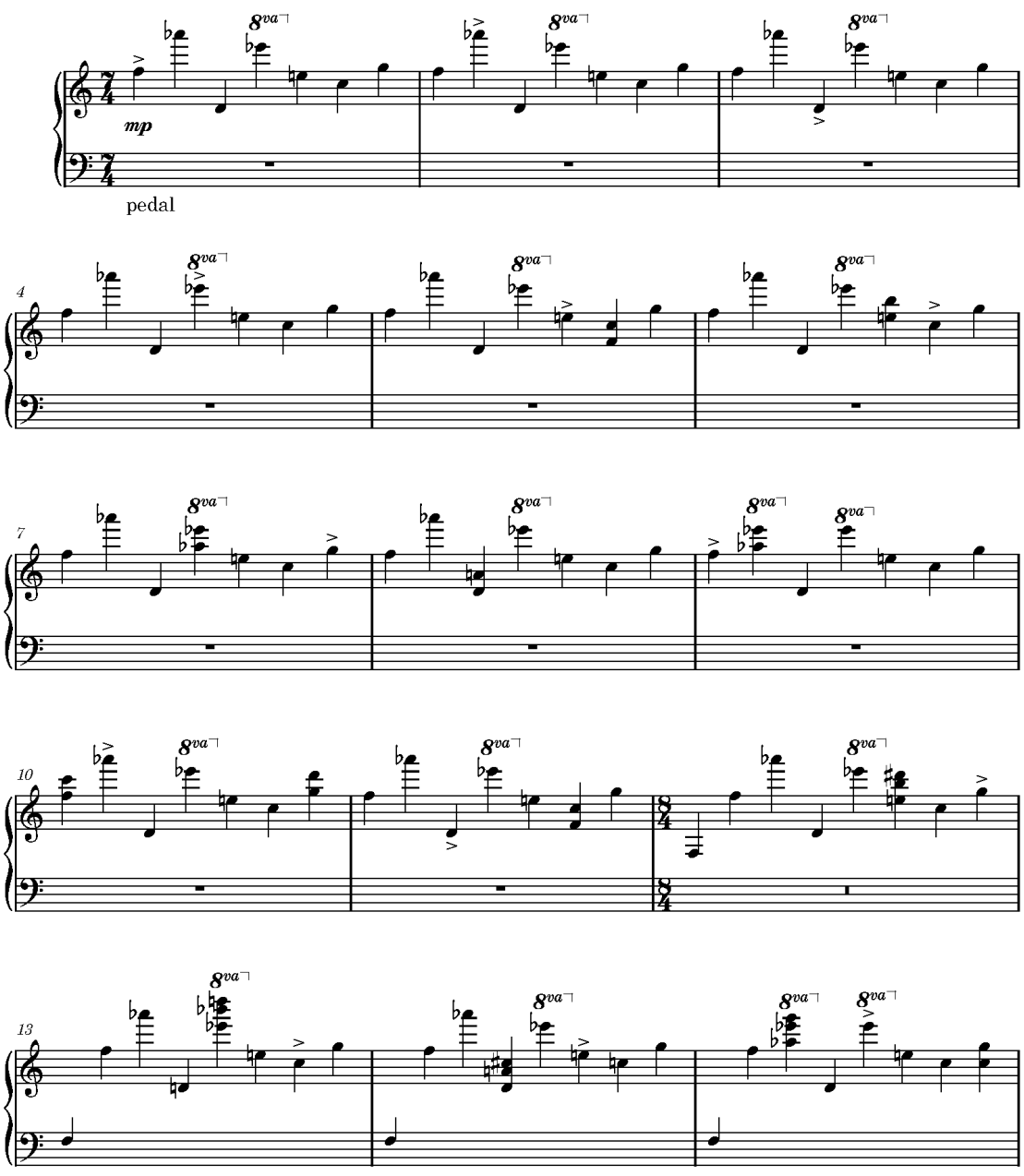


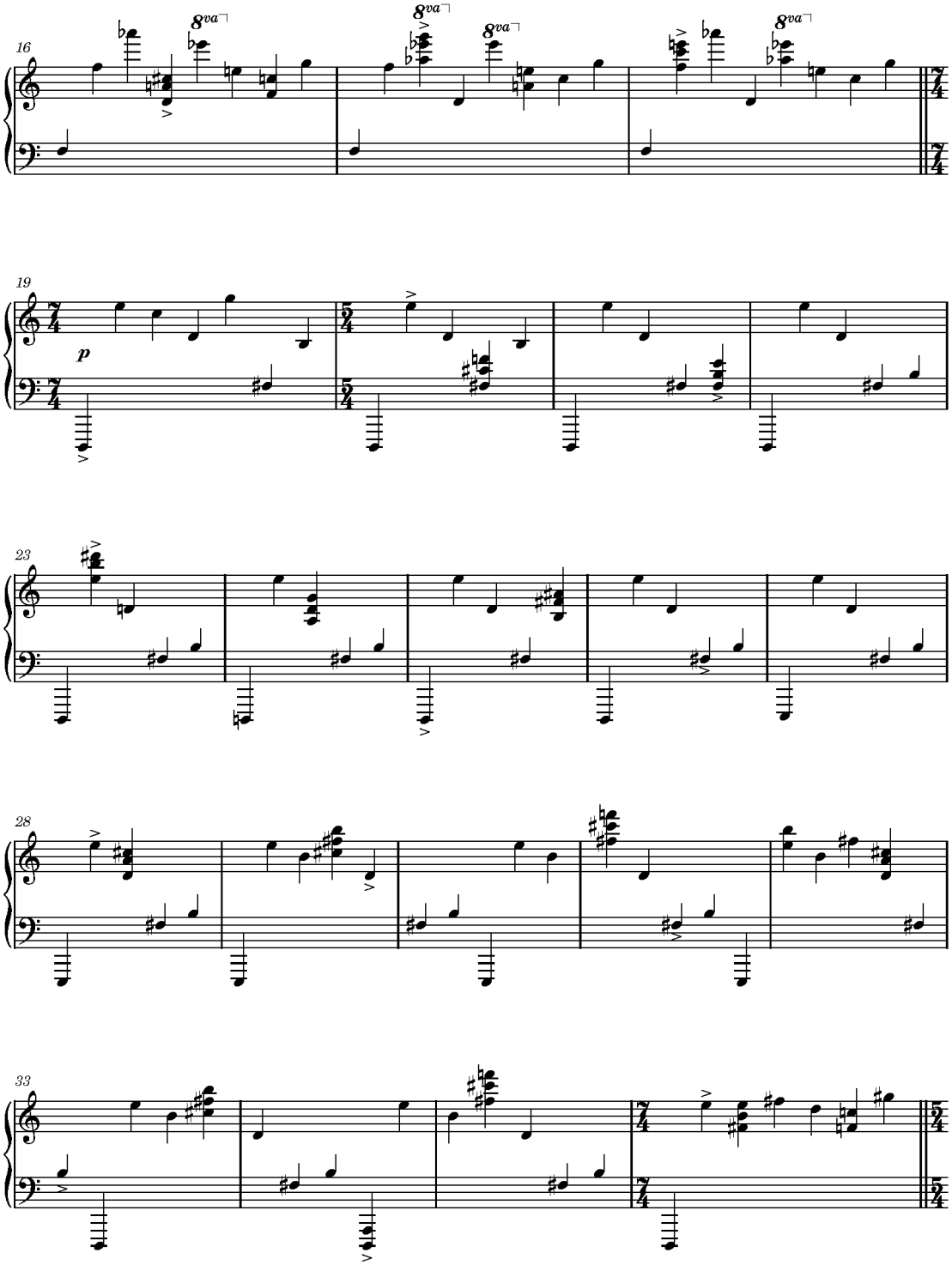
5 July 2017
Nice
8'30

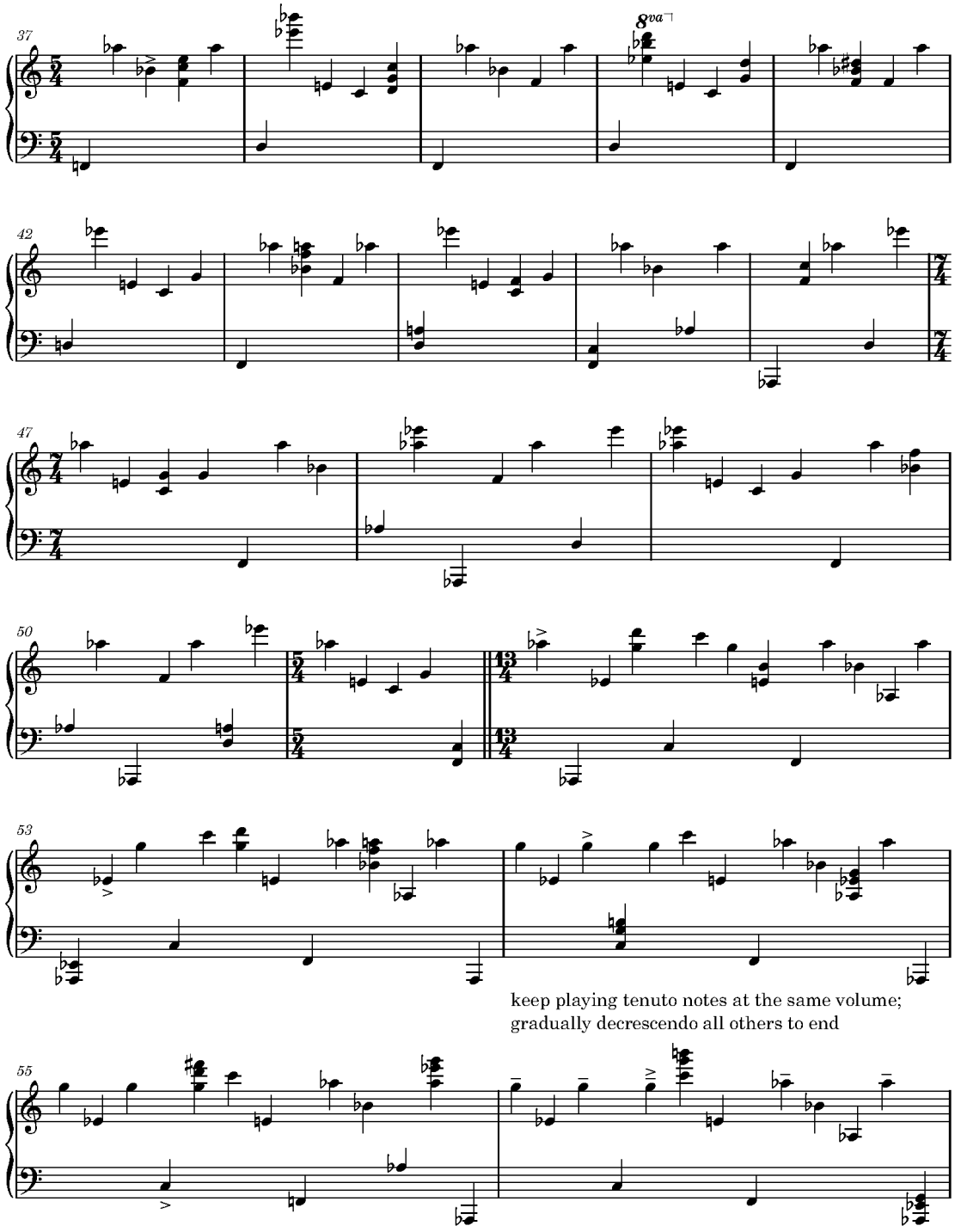
Flow 4

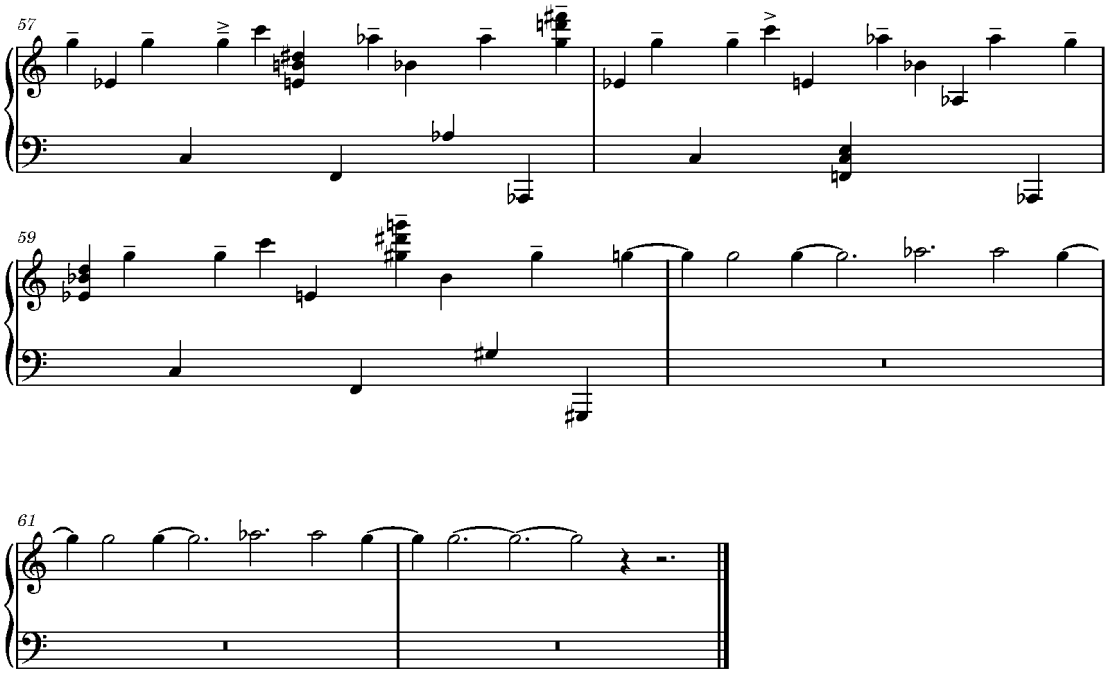
for Brianna Matzke

Elliot Cole




16
8va
8va
8va
19
p
23
28
33


37
8va
42
47
50
53
keep playing tenuto notes at the same volume;
gradually decrescendo all others to end
55



12 July 2017
Paris
5'10

Flowerpot Music (NYU Book) (2018)

THE PLAYERS and the INSTRUMENTS

For 20 or more players.

All players spread out as uniformly as possible through the space. Stand.

Each player has a number (Player 1, Player 2, etc…)

Each player should have 1 pot in one hand, 1 mallet in the other, and have 1 or 2 pots and a second mallet on the floor near them.

THE SYNTAX

[>A] remember this bloom ('store in A'). [A] reproduce it exactly.

Every statement separated by 1 or more periods is an event.

Every statement in parentheses prepares an event but is not an event.

. a pause,
.. a longer pause
… a pause long enough to end a phrase.

, leader continues (in time) after the last player

ll: repeat signs :ll

THE VOCABULARY

Actions

new bloom (aka spreading activation)

Someone strikes their pot once. They become the leader:

if they are near you,
you strike after them as soon as you can;

if they are not near you,
wait until someone near you strikes, and then strike –

Dynamics: leader makes a full sound, ***f*** or ***mf*** and each person plays either the same or quieter volume than the player they follow (their 'trigger').

Listen to the pattern that emerges as the notes ripple through the space.

thick bloom

Bloom but each player plays all of their pots in quick succession instead of just one

exactly

Attempt to exactly reproduce the last bloom.

double tap (triple tap, 10x tap…)

The leader strikes twice, triggering two blooms (that should be identical). Blooms must overlap.

follow someone else

The leader remains the same, but other players choose someone else near them to follow.

toll X

Play the pot X times, equally spaced, slow, like the tolling of the bell. If multiple people are tolling, they should play together, cued by the leader.

frisbee

Just like frisbee: each strikes their pot and 'passes' it to another player.

Think of the pacing of frisbee: the travel takes some time, and then the receiver can take any amount of time they like before sending it off again.

It can be nice to get into a rhythm, and then it can be fun to break it by switching up the order or timing.

frisbee: varied

Frisbee players vary the timing and volume of their passes.

frisbee: pulse

Frisbee players each wait the same duration as the person they follow. Always follow the same person… except sometimes, when you can switch up the pattern.

frisbee: groove

Frisbee players find a shared pulse but vary their rhythms within it.

frisbee: open

Frisbee players can include people not yet in the circle.

Methods for volume

softer

Repeat the last bloom at a softer volume.

louder

Repeat the last bloom at a louder volume.

different volume

Repeat with the same timing, but play with a different volume (no relation to neighbors).

smooth volume

Listen to the wave of notes as it comes at you.

If you think that it, when it gets to you, is in the process of getting louder, help it along by playing louder.

If it is getting softer, help it get softer.

decrescendo

Repeat the last bloom; the leader plays full volume and then each person plays slightly softer than the person they follow.

crescendo

Leader begins soft and everyone plays a little louder than the person before.

soft

loud

Methods for time

slow

fast

slower

Repeat the last bloom with a little more time between each note.

faster

Repeat the last bloom but respond more quickly to your trigger.

smooth time

Repeat the last bloom but try to match the wait time of the person you follow.

jagged time

Repeat the last bloom but each player should add a slight variation in time, volume, or order of notes (by following a different person near you).

rit.

The first follower plays quickly after the leader, then each person plays slightly slower than the person they follow.

accel.

The first follower plays slowly after the leader, then each person plays slightly faster than the person they follow.

Methods for space

shuffle the map

Your goal is to repeat the last bloom, as it sounded, over and over, with a patient breath in between each repetition, as accurately as possible, while you slowly walk to a new position in the room. When everyone is still, *shuffle the map* is over.

Notice the inevitable variations that emerge from imperfection.

When you arrive at your new position, stay there. The leader leads the pattern a few more times, while you close your eyes and consider the new sonic map of the room.

traveling unisons

Two or more people play —
||:
they strike their pots in unison once and let it ring
then move to different places (not together)
:||

track movement

The leader must travel in between each stroke.

Everyone else must listen for changes in the position of their leader and travel themselves to put it back in the same position (to maintain the same distance and direction).

SCORES

harmonic pivots with fades

(evens change pots)
(odds) new bloom. faster. exactly.

(odd last player) toll 2..

||:

(odds) exactly.

<< (evens) new soft bloom.. (odds) exactly.. >> *leaders play together*
<< (evens) louder.. (odds) softer.. >>
<< (evens) loud.. (odds) soft.. >>
(odds change pots)
(evens) exactly… …
<< (evens) exactly.. (odds) new soft bloom.. >>
<< (evens) softer.. (odds) louder.. >>
<< (evens) soft.. (odds) loud.. >>
(evens change pots)…

:||

wheels within wheels

(three players closest to the center of the space) play frisbee

ll: (a ring of people a little farther out) play frisbee :ll (until the whole room is playing)

(after a while) smooth time (so each circle has its own steady pulse)

slowly accel. (each circle independently)

(when very fast) decrescendo… crescendo… decrescendo… crescendo…

change direction from time to time

double acceleration

(new leader) accel. (first follower starts slow)
ll: accel. (both leader and first follower starts slightly sooner than before) :ll
(loops will start to overlap)

hypnosis

(new leader) new bloom… faster…

(leader starts each new one in time, whatever that comes to mean; it should be after or near the end of the series)

exactly, exactly, smooth time, smooth time, exactly,
ll: different volume, smooth volume, :ll
softer, different volume, (odds sticks) exactly, exactly, decrescendo,
(odds heads) jagged time, smooth time, smooth time, louder, faster, faster,
softer, softer, louder, louder, louder, louder, soft, exactly, exactly,
follow someone else, follow someone else, follow someone else,
change volumes
(everyone sticks) faster, softer, exactly, exactly, exactly
(everyone sticks except 1-3 heads, loud) exactly, exactly, exactly,
(everyone sticks except 1-8 heads, loud) exactly, exactly, exactly,
(everyone sticks except 15-20 heads, loud) exactly, exactly, exactly,
(everyone sticks except 1-5 heads, loud) exactly, exactly, exactly,
(everyone sticks) quiet… double tap…

falling voice leading

(everyone plays on the higher of their two pots)

(new leader) new bloom… exactly…
(sometime in the next few minutes change to a larger pot) ||: exactly… :||
(until everyone is on their low pot)

pulse and pass

One person starts: plays some number of pulses on one pot.
With eye contact and a cue, pass it to someone else.
They continue the same pulse.
They pass it on OR end it.

…

The same leader starts again.
If you've been passed to before, try to be more "on the ball" and pass it along more quickly than last time (but continue to pulse in time; just pulse fewer times).

Play the score *smooth volume* OR be present to the dynamic shape and make choices reflecting that presence.

Variants:
pulse and pass: changing tempo
pulse and pass: steady tempo

thick frisbee

4 corner frisbee begins…
… let it go for a while …

One by one, other players join by:

choosing one sound to be their trigger: when you hear yours, play. Wait the same amount of time between the trigger and your response every time.

If you hear yourself play at exactly the same time as someone else, change pots.

When everyone is playing, 4 corner frisbee players slow down their game…
and stop.

negative/positive

||:

new bloom

||: **5x**
(odds play with mallet sticks / evens heads) exactly.. (odds sticks / evens heads) exactly..
:||

…

:||

(all players with sticks) new bloom.. tighter & smoother.. tighter 10 taps…

alternate paths

||:
leader tolls some number of bells; shows which is last, which triggers a new bloom… exactly…
follow someone else… follow someone else, softer…
:||

knock out

new slow smooth bloom involving 4-8 people only

||: exactly… exactly… exactly… :||

If you are not involved in the bloom, you can get in by playing at the exact same time as someone else. if you succeed at a perfect unison, you have 'knocked them out' of the sequence and you must take over that spot.

If you are in the bloom and you hear someone play exactly at the same time as you, you are out.

If you try to knock someone out but don't play at exactly the same time, you must lay out for 5 blooms before you can try to knock back in

unfolding

(player 1) slow toll 3,
do the phrase: **[new fast bloom decrescendo… exactly… exactly… slower… softer… …]**

do for: (players 1-5, 1 leads)… (players 1-10)… (players 1-20)… (players 1-20 thick) … soft double tap … …

5x (overlapping) ll: (all sticks! leaders, in order: 1, 7, 11, 13, 19) new fast decrescendo bloom, :ll

4x ll: (players 1-5, all heads, different leader for each) new slow bloom decrescendo… :ll

(player, sticks) toll 3 … … …

a/b

(1-10) new bloom… (11-20) new bloom…

(1-10) faster.. exactly… (11-20) faster.. slower.. decrescendo… softer… exactly…

ll:
(1-10, some people change pots) faster & softer… (11-20) exactly…
(1-10, some people change pots) louder… (11-20) exactly…
(1-10, some people change pots) new bloom… slower… (11-20) exactly… …
:ll

spring day on the quad

(players 1-3, player 1 leads) new bloom (id each other)… play frisbee

(every once in a while) throw to a new person, who must then throw it to someone already playing. They are not now in rotation; this was a one time deal.

(after a while) get into a steady rhythm

(after a while) someone playing can throw twice, spinning off a second game… and a third… until everyone is playing.

Games should be local: play with people near you. Each game should settle over time into a consistent pulse. When the texture seems thick and static, pause your game for a bit.

neighbors

Softly roll on one of your pots.

If you think your neighbor is getting louder, get louder. If you think your neighbor is getting softer, get softer.

Change pots freely.

Avoid sustained loud dynamics. *Forte* should be a peak, not a plateau. Enjoy and explore many shades of *piano* and *mezzo*.

(After a bit) cresc all together, earthquake*, decrescendo to silence

* set a second pot against your pot to get a loud rattle

undo

(everyone) new bloom decrescendo… faster…
make more beautiful… make more beautiful… exactly… [>A]

…

10x ll: [A]… softer & some people don't play.. :ll

Works Cited

Alsop, Roger. “Exploring the Self Through Algorithmic Composition.” *Leonardo Music Journal* 9 (1999): 89–94. https://doi.org/10.1162/096112199750316866.

Baird, Davis. Essay. In *Thing Knowledge: A Philosophy of Scientific Instruments*. Berkeley, CA: University of California Press, 2004.

Cage, John. *A Year from Monday New Lectures and Writings*. Middletown, CT: Wesleyan University Press, 1985.

Cage, John. Experimental Music: Doctrine. In *Silence: Lectures and writings*. Middletown, CT: Wesleyan University Press, 1961.

Chomsky, N. “Three Models for the Description of Language.” *IEEE Transactions on Information Theory* 2, no. 3 (1956): 113–24. https://doi.org/10.1109/tit.1956.1056813.

Eno, Brian. “Composer as Gardener.” *The Serpentine Gallery*. Lecture presented at the The Serpentine Gallery Garden Marathon, October 16, 2011. Retrieved February 28, 2021, from https://www.edge.org/conversation/brian_eno-composers-as-gardeners

Feldman, Morton. *Three Voices*. Universal Editions, 1982.

Hankins, Thomas L., and Robert J. Silverman. *Instruments and the Imagination*. Princeton, NJ: Princeton University Press, 2014.

Harrison, Lou. *Music Primer; Various Items about Music to 1970*. C.F. Peters Corp., 1971.

Ihde, Don. *Technics and Praxis*. Dordrecht, Holland: D. Reidel, 1979.

Larson, Kay. *Where the Heart Beats: John Cage, Zen Buddhism, and the Inner Life of Artists*. New York: Penguin Books, 2013.

Lundell, A., and Arnold Schoenberg. Alfred Lundell: Interview with Arnold Schoenberg. Arnold Schönberg Center. Accessed February 27, 2021. https://www.schoenberg.at/index.php/en/alfred-lundell-interview-with-arnold-schoenberg.

Magnusson, Thor. “Designing the Threnoscope or, How I Wrote One of My Pieces.” *Sound Work*, 2021, 219–30. https://doi.org/10.2307/j.ctv1ccbg96.14.

Magnusson, Thor. *Sonic Writing: Technologies of Material, Symbolic, and Signal Inscriptions*. London: Bloomsbury Academic, 2019.

Meyer, Leonard B. “Meaning in Music and Information Theory.” *The Journal of Aesthetics and Art Criticism*, vol. 15, no. 4, 1957, pp. 412–424., doi:10.1111/1540_6245.jaac15.4.0412.

Messiaen, O. *Technique of My Musical Language*. Paris: Leduc, 1956.

Matossian, Nouritza. *Xenakis*. London: Kahn & Averill, 1986.

Prusinkiewicz, Przemyslaw, James S. Hanan, and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. New York etc.: Springer, 1996.

Sapolsky, Robert. "Dopamine Jackpot! Sapolsky on the Science of Pleasure." *FORA.tv*. Lecture. www.youtube.com/watch?v=axrywDP9Ii0.

Shannon, C. E. "A Mathematical Theory of Communication." *Bell System Technical Journal*, vol. 27, no. 4, 1948, pp. 623–656., doi:10.1002/j.1538-7305.1948.tb00917.x.

Shultis, Christopher. "Silencing the Sounded Self: John Cage and the Intentionality of Nonintention." *The Musical Quarterly* 79, no. 2 (1995): 312–50. https://doi.org/10.1093/mq/79.2.312.

Southam, Ann. *Simple Lines of Enquiry,* Eve Egoyan, piano. Centrediscs, 2015, CD.

Waddington, C. H. *The Strategy of Genes*. London: Allen & Unwin, 1957.

Xenakis, I., Brown, R. & Rahn, J. "Xenakis on Xenakis." *Perspectives of New Music*, Vol. 25, No. 1/2 (1987): 16-63.

Xenakis, Iannis, and Sharon Kanach. *Formalized Music: Thought and Mathematics in Composition*. Styvesant, NY: Pendragon Press, 1992.