Guido Kramann
*1968

ECADE
//1//

für Klavier

ECADE //1//

Notenbild

Die farbliche Kodierung der Anschlagstärke der Töne am Klavier eröffnet die Möglichkeit sehr leicht einzelne Töne in einem Zusammenklang hervorzuheben. Das wiederum macht es leicht, sehr verschachtelte polyphone Strukturen unmittelbar sichtbar und damit auch pianistisch interpretierbar zu machen.

Kompositionstechnik

Komponiert wurde das Stück am Computer unter Verwendung eines Disklaviers durch stetiges Verändern von Programmzeilen, die in einer selbst entwickelten generativen Sprache für musikalische Strukturen verfasst sind, bei gleichzeitigem Hineinhören in das nach und nach entstehende klangliche Resultat.
Das fertig gestellte generative Programm stellt am Ende eine höhere Ordnung dar, der alles was erklingt unterliegt. Zum einen der Bezug auf diese höhere Ordnung bei der fertigen Komposition und zum anderen die sorgsame Ausgestaltung dieser endgültigen Form in einem iterativen, kreativen Prozess, machen gemeinsam das hier verfolgte ästhetische Konzept aus.

Details

Nachfolgend ist sind die Programmzeilen aufgelistet, die die vorliegende Komposition generieren, diese also damit auch auf sehr kompakte Art und Weise repräsentieren:

```
\'>144*60y \'>336*60z \'-Y-Zh
\720000x
\H:1%6a \H:1/6*12b \A+B+Xo
\H:2%12a \H:2/12*24b \A+B+Xp
\H:3%24a \H:3/24*48b \A+B+Xq
\H:8%48a \H:8/48*96b \A+B+Xr
\H:7%96a \H:7/96*192b \A+B+Xs
\'>336*10s \H/48>0*50+S+130[B]
\H/96>0*100+300[P]
\'>336*5040u \420*24+Ub
\'>336*7w \'<240*7+W+43m
\'>336*12+97n
\'>336*2+1i
\P|Q!B:2*I)M(Na[M0]
\P|Q!B:3*I)M(Nb[M1]
\P|Q!B:5*I)M(Nc[M2]
\P|Q!B:7*I)M(Nd[M3]
\H/48%2q \H.2520*1*Q*I)M(N[M4]
\H/96%2q \H.2520*2*Q*I)M(N[M5]
\H/192%2q \H.2520*3*Q*I)M(N[M6]
\H/384%2q \H.2520*4*Q*I)M(N[M7]
\15g
\0"*2+G[V0]
\1"*2+G[V1]
\2"*2+G[V2]
\3"*2+G[V3]
\4"*3+G[V4]
\5"*3+G[V5]
\6"*3+G[V6]
\7"*3+G[V7]
\0"a*3 \1"b*3 \2"c*3 \3"d*3 \90-A-B-C-D[S]
\'>390[E]
```

Zum Verständnis: Ausgangspunkt für die Idee diese generative Skriptsprache zu entwickeln, liegt in der Überzeugung, dass sich aus der Transformation der zeitlichen Abfolge der natürlichen Zahlen unmittelbar musikalische Strukturen generieren lassen. Konkret nutzt das Programm als dynamisches Element deshalb ausschließlich einen Zählvorgang 0,1,2,3,4, usw., der in einem Tempo vollzogen wird, das das Programm in der Ausgangs-Variable `[B]` speichert. Der aktuelle Zählstand kann überall im Skript durch `'` abgerufen werden. So endet das vorliegende Stück, wenn der Zähler größer 390 ist (siehe letzte Zeile). Auf den aktuellen Zählerstand `'` , auf eine konstante Zahl, oder auf eine der eine Zahl repräsentierenden Variablen A..Z, also auf drei Typen von Quellen, können der Reihe nach unäre oder binäre Operationen angewendet werden, wobei die binären Operationen wieder eine Quelle hinter dem Operator benötigen, die unären aber nicht. Die einfachsten binären Operationen entsprechen den gebräuchlichen arithmetischen Operationen `+-*/` .Besonders ist hier, dass Nachkommastellen abgeschnitten werden und alle nicht gültigen Operationen Null ergeben. Wegen dieser grundlegenden Operationen hat die Programmiersprache den Namen **Arithmetic Operation Grammar** (**AOG**) erhalten, siehe beispielsweise *Computer Music Journal CMJ 44(1):S.17-34*. Jede Abfolge mehrerer Operationen wird mit dem Symbol `\` eingeleitet und endet damit, dass das Resultat nach Durchlaufen aller, oder einiger Operationen entweder an eine externe Variable zur Steuerung des musikalischen Geschehens geschickt wird `[...]` oder aber an eine interne Variable `A..Z`, indem der entsprechende Buchstabe klein geschrieben wird `a..z`. Was hier innerhalb des Programms durch eine Vielzahl an Operationen geformt wird, sind im Wesentlichen Frequenzen, die von einem Instrument gespielt werden sollen. Die externen Variablen `[M0]` , `[M1]` , `[M2]` ... besagen, dass der hier vorliegende berechnete Wert als Frequenz interpretiert werden soll, nachdem er um eine Promillezahl `[P]` vergrößert, bzw. verkleinert wurde und der nächst liegende Ton der temperierten Stimmung gespielt werden soll. Es wird mit *MIDI* gearbeitet und bei `[M0]` , `[M1]` , `[M2]` ... wird immer ein *sendNoteOn*-Befehl auf den als Ziffer hinter `M` stehenden Kanal gesendet. Entsprechend legen `[V0]` , `[V1]` , `[V2]` … die jeweilige Tonlautstärke fest und mit `[S]` wird das Sustain-Pedal bei Verwendung eines Klaviers gesteuert. Die Operationen werden stets in der Reihenfolge ihres Auftretens abgearbeitet. Anders als in der Arithmetik gibt es hier also keine besondere Priorisierung. Dies zum Grundverständnis der Skriptsprache **AOG**. Der Vollständigkeit halber seien noch alle 5 unären `$@_~''` und 20 binären `+-*/:%^<>={}()|.,!?§` Operationen kurz in ihrer Bedeutung erklärt:

unär:

`$` Signum, liefert 1 wenn Quelle x größer 0 ist, sonst 0.

`@` inverses Signum: liefert 0 wenn Quelle größer 0 ist, sonst 1.

`_` Reduktion: liefert das Produkt aller Primfaktoren {2,3,5,7} aus q, beispielsweise 12=2*2*3 für x=132.

`~` liefert aus x den nächst höheren Wert, der sich nur aus den Primfaktoren 2,3,5,7 zusammensetzt.

`''` liefert den Durchschnitt des paarweise berechneten auf {2,3,5,7} beschränkten Gradus Suivitatis zwischen der Frequenz, die zuletzt im mit x bezeichneten Kanal aufgetaucht ist und jeder anderen Frequenz in den anderen Kanälen. Liefert damit so etwas, wie den Anteil, den der gewählte Ton auf die Gespanntheit des Gesamtzusammenklangs hat.

Gradus Suivitatis, musikalisch interpretierter Dissonanzgrad zweier ganzer Zahlen nach Leonhard Euler, siehe auch `§` weiter unten.

binär:

`+-*/` gebräuchliche arithmetische Operationen. Achtung: 1/0 liefert hier 0, 3/2 liefert hier 1, vergl. Text weiter oben.

`:` wie `/` , liefert aber 0, wenn die Division nicht ohne Rest erfolgen kann, nur für positive Operanden definiert, sonst 0.

`^` Potenzieren. Nur für positive Operanden definiert, sonst 0.

`%` Modulo- (Rest-) division, wie sie beispielsweise in den Programmiersprachen C oder Java definiert sind.

`<>=` Vergleichsoperatoren, liefern 1, wenn erfüllt, sonst 0.

`{}` Wie `<>`, liefern aber für linken Operand x wenn erfüllt x und nicht 1, sonst 0.

() Wie **{}**, das zweite Argument y wird aber als Midi-Tonhöhe interpretiert und vor dem Vergleich unter Berücksichtigung von **[P]** (vergl. Text weiter oben) erst in eine Frequenz umgewandelt.

| Verkettung. Ist das zweite Argument y größer 0, wird dieses als Ergebnis geliefert, sonst das erste x.

§ paarweise auf {2,3,5,7} beschränkter Gradus Suivitatis. Man nimmt die Primfaktoren, in denen x und y nicht miteinander übereinstimmen, aber hier eben nur {2,3,5,7}. Aus deren Auftrittshäufigkeit p für 2, q für 3, r für 5, s für 7 berechnet man: gradus = (2-1)*p+(3-1)*q+(5-1)*r+(7-1)*s.

, liefert das Produkt aller Primfaktoren in x, die auch in y vorkommen, 0, wenn keine gefunden werden.

. liefert das, was übrig bleibt, wenn man aus y alle Primfaktoren aus {2,3,5,7} entfernt, die auch in x vorkommen.

? wie **,** ...aber Modulodivision der berücksichtigten Anzahlen der Primfaktoren in x durch die Anzahlen in y plus eins.

! wie **.** ...aber Behandlung analog zu **?**.

Grundideen, nach denen das Verfertigen einer Komposition mit Hilfe von **AOG** erfolgt (Beispiele grob angesprochen):

◔ Etwas dividieren **/** , um es langsamer und gleichzeitig tiefer zu machen.

◑ Den Fluss der Zeit durch Wiederholungen (**%** Modulo), oder Sprünge (**+** Addition) verändern.

◕ Änderung des prozentualen Faktors **[P]** mit dem alle Frequenzen multipliziert werden entspricht annähernd einem Skalen- oder Tonartwechsel.

● Die Operationen **, . ! ?** können dazu genutzt werden, die verwendeten Frequenzen auf solche zu beschränken, die in einem einfachen Teilungsverhältnis zueinander stehen.

Die Kompositionen der ECADE-Reihe entstehen über den Umweg eines generativen Programms und unter Verwendung einer farbkodierten Notenschrift. Beides eröffnet die Umsetzung neuartiger musikalischer Strukturen und ermöglicht insbesondere auch deren Darstellung. Die ansprechende Kompaktheit der Notendarstellung soll im besten Fall Anklang bei fortgeschrittenen Pianistinnen und Pianisten finden und dazu motivieren, sich die Kompositionen einmal vorzunehmen. Gleichzeitig soll die Reihe auch die generative Sprache **AOG** populärer machen, welche auch in Verbindung mit der Bewegung Ubiquitous Music (siehe beispielsweise *D. Keller et. al.: Ubiquitous Music, Springer*) als möglicher Zugang für Laien zum Komponieren gedacht ist.

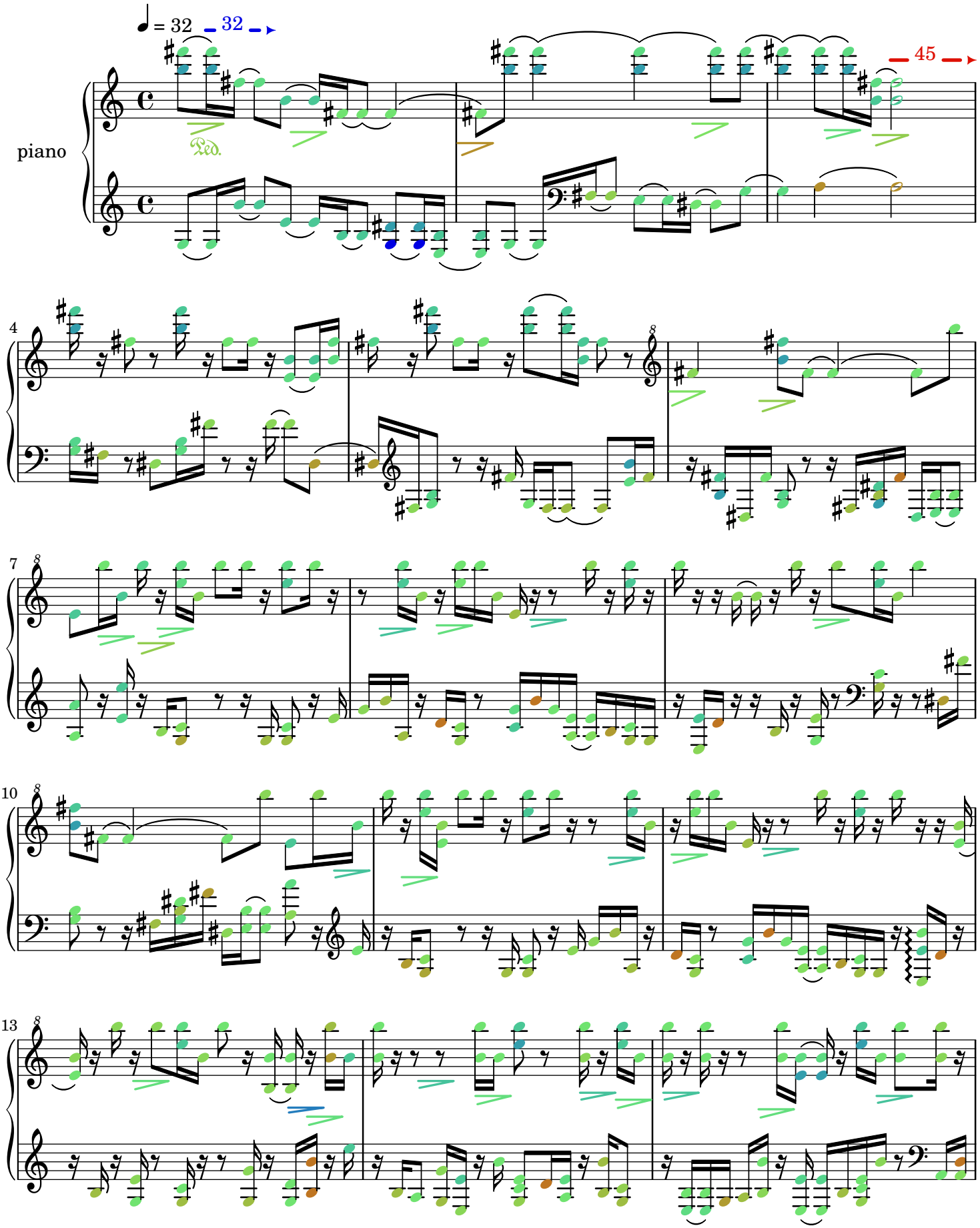
Guido Kramann
Leipzig, 15.08.2024

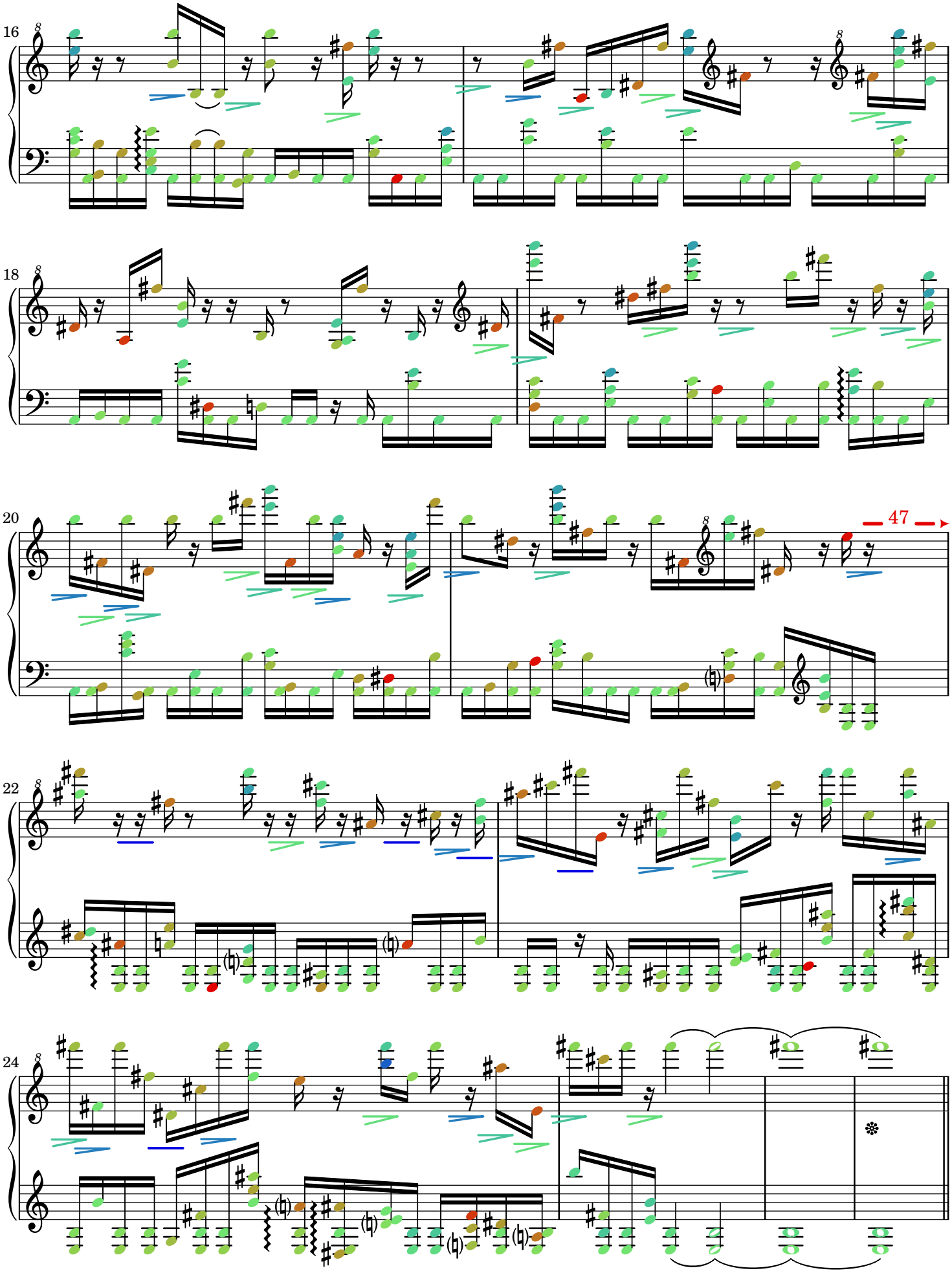
ECADE //1//

für Klavier, Stärke bei Tonanschlag und Haltepedal, sowie Tempo sind farbkodiert

min max

Guido Kramann





ECADE //1//

Musical notation

The colour coding of the velocity of the notes on the piano makes it very easy to emphasise individual notes in a harmony. This in turn makes it easy to immediately visualise very complex polyphonic structures and thus to interpret them pianistically.

Composition technique

The piece was composed on the computer with the use of a Disklavier by constantly changing lines of code written in a self-developed generative language for musical structures, while at the same time listening to the gradually emerging sonic result.
In the end, the completed generative programme represents a higher order to which everything that sounds is subject. On the one hand, the reference to this higher order in the finished composition and, on the other, the careful shaping of this final form in an iterative, creative process, together make up the aesthetic concept pursued here.

Details

Below is a list of the lines of the code that generate the present composition and thus represent it in a very compact way:

```
\'>144*60y \'>336*60z \'-Y-Zh
\720000x
\H:1%6a \H:1/6*12b \A+B+Xo
\H:2%12a \H:2/12*24b \A+B+Xp
\H:3%24a \H:3/24*48b \A+B+Xq
\H:8%48a \H:8/48*96b \A+B+Xr
\H:7%96a \H:7/96*192b \A+B+Xs
\'>336*10s \H/48>0*50+S+130[B]
\H/96>0*100+300[P]
\'>336*5040u \420*24+Ub
\'>336*7w \'<240*7+W+43m
\'>336*12+97n
\'>336*2+1i
\P|Q!B:2*I)M(Na[M0]
\P|Q!B:3*I)M(Nb[M1]
\P|Q!B:5*I)M(Nc[M2]
\P|Q!B:7*I)M(Nd[M3]
\H/48%2q \H.2520*1*Q*I)M(N[M4]
\H/96%2q \H.2520*2*Q*I)M(N[M5]
\H/192%2q \H.2520*3*Q*I)M(N[M6]
\H/384%2q \H.2520*4*Q*I)M(N[M7]
\15g
\0"*2+G[V0]
\1"*2+G[V1]
\2"*2+G[V2]
\3"*2+G[V3]
\4"*3+G[V4]
\5"*3+G[V5]
\6"*3+G[V6]
\7"*3+G[V7]
\0"a*3 \1"b*3 \2"c*3 \3"d*3 \90-A-B-C-D[S]
\'>390[E]
```

For understanding: The starting point for the idea of developing this generative scripting language lies in the conviction that musical structures can be generated directly from the transformation of the temporal sequence of natural numbers. In concrete terms, the programme therefore exclusively uses a counting process 0,1,2,3,4, etc. as a dynamic element, which is carried out at a tempo that the programme stores in the output variable `[B]`. The current count can be called up anywhere in the script using `'`. For example, this piece ends when the counter is greater than 390 (see last line). Unary or binary operations can be applied in sequence to the current count `'` , to a constant number, or to one of the variables A..Z representing a number, i.e. to three types of sources, whereby the binary operations again require a source after the operator, but the unary operations do not. The simplest binary operations correspond to the common arithmetic operations `+-*/` . What is special here is that decimal places are cancelled and all invalid operations result in zero. Because of these basic operations, the programming language has been given the name Arithmetic Operation Grammar (**AOG**), see for example *Computer Music Journal CMJ 44(1):p.17-34*. Each sequence of several operations is introduced with the symbol `\` and ends with the result being sent either to an external variable to control the musical event after all or some of the operations have been performed `[...]` or to an internal variable `A..Z` by writing the corresponding letter `a..z` in lower case. What is formed here within the programme by a large number of operations are essentially frequencies that are to be played by an instrument.

The external variables `[M0] , [M1] , [M2] ...` indicate that the calculated value here is to be interpreted as a frequency after it has been increased or decreased by one per mille `[P]` and the nearest note of the tempered tuning is to be played. *MIDI* is used and with `[M0] , [M1] , [M2] ...` a sendNoteOn command is always sent to the channel indicated by the number behind M. Accordingly, `[V0] , [V1] , [V2] ...` define the respective note volume and `[S]` controls the sustain pedal when using a piano. The operations are always processed in the order in which they occur. Unlike in arithmetic, there is no special prioritisation here. This provides a basic understanding of the AOG scripting language. For the sake of completeness, all 5 unary `$@_~'` and 20 binary `+-*/:%^<>={}()|.,!?§` operations are briefly explained in their meaning:

unary:

`$` Signum, returns 1 if source x is greater than 0, otherwise 0.

`@` inverse signum: returns 0 if source is greater than 0, otherwise 1.

Reduction: returns the product of all prime factors {2,3,5,7} from q, for example 12=2*2*3 for x=132.

`~` returns the next higher value from x, which is only composed of the prime factors 2,3,5,7.

`'` returns the average of the pairwise calculated gradus suivitatis limited to {2,3,5,7} between the frequency that last appeared in the channel labelled x and every other frequency in the other channels. This provides something like the proportion that the selected tone has on the tension of the overall sound. *Gradus Suivitatis*, musically interpreted degree of dissonance of two whole numbers according to Leonhard Euler, see also `§` below.

binary:

`+-*/` common arithmetic operations. Note: 1/0 returns 0 here, 3/2 returns 1 here, see text above.

`:` like `/` , but returns 0 if the division cannot be performed without remainder, only defined for positive operands, otherwise 0.

`^` Exponentiation. Only defined for positive operands, otherwise 0.

Modulo (remainder) division, as defined in the programming languages **C** or **Java**, for example.

`<>=` comparison operators, return 1 if fulfilled, otherwise 0.

`{}` Like `<>`, but return x for the left operand if fulfilled and not 1, otherwise 0.

`()` Like `{}`, but the second argument y is interpreted as a midi pitch and first converted into a frequency before the comparison, taking into account `[P]` (see text above).

| concatenation. If the second argument y is greater than 0, this is returned as the result, otherwise the first x.
§ Pairwise gradus suivitatis restricted to {2,3,5,7}. Take the prime factors in which x and y do not match, but here only {2,3,5,7}. From their frequency of occurrence p for 2, q for 3, r for 5, s for 7, you calculate: gradus = (2-1)*p+(3-1)*q+(5-1)*r+(7-1)*s.
, returns the product of all prime factors in x that also occur in y, 0 if none are found.
. returns what remains if all prime factors from {2,3,5,7} that also occur in x are removed from y.
? like , ...but modulo division of the considered numbers of prime factors in x by the numbers in y plus one.
! like . … but treatment according to ?.

Basic ideas according to which a composition is created with the help of **AOG** (examples roughly mentioned):

◔ Divide something / to make it slower and at the same time lower in pitch.

◑ Change the flow of time through repetitions (% modulo) or jumps (+ addition).

◕ Changing the percentage factor [P] by which all frequencies are multiplied corresponds approximately to a change of scale or key.

● The operations ,.!? can be used to restrict the frequencies used to those that are in a simple division ratio to each other.

The compositions in the ECADE series are created using a generative program and colour-coded notation. Both enable the realisation of new musical structures and, in particular, their presentation. The appealing compactness of the notation should ideally appeal to advanced pianists and motivate them to try out the compositions. At the same time, the series is also intended to popularise the generative language **AOG**, which, in conjunction with the *Ubiquitous Music* movement (see, for example, *D. Keller et. al.: Ubiquitous Music, Springer*), is intended as a possible approach to composing for laypeople.

Guido Kramann
Leipzig, 15/08/2024