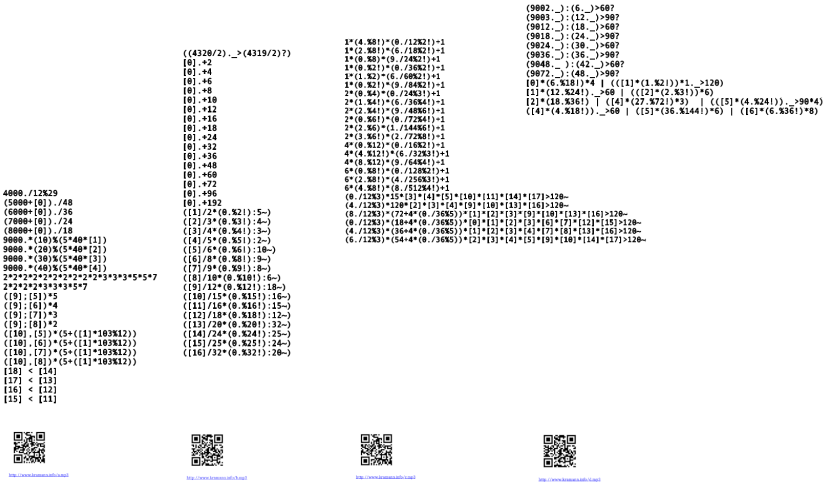
xCoAx 2023 11th Conference on Computation, Communication, Aesthetics & X
2023.xCoAx.org
Weimar, Germany

AOGscript, Generative Code as Conceptual Artwork: Giving Space to Astonishment



Guido Kramann
kramann@gmx.li
Leipzig, Germany

DOI 10.34626/xcoax.2023.11th.401

What is the aesthetic essence of lines of code in a scripting language designed to generate musical structures? Four simple black-and-white prints with code invite to trace this question. The code sequence on one sheet each served to generate a composition for clarinet and piano. In addition to the reception of the code sequences, audio files of these four compositions can be accessed in the exhibition. The links to the audio files are located at the bottom of the printouts and can be listened to via headphones on one's own smartphone.

Keywords: Conceptual Art, Computational Music, Generative Art, Arithmetic Operation Grammar (AOG).

Code that generates music can be seen as a concretization of a certain conception of what can be considered music. This conceptual notion is grounded in "Arithmetic Operation Grammar" (AOG) in the fact that the distribution of small prime factors (2,3,5,7) in the sequence of natural numbers is already considered to be organized in a musically meaningful way, and original musical compositions can be obtained by transformations of this sequence (Kramann 2021). With the line-by-line top-down interpreted script language *AOGscript*, such transformations can be described very easily. Everything that follows the # character in *AOGscript* is a line comment. The following lines represent an example of code in the *AOGscript* language and are also intended as an introduction to this script language:

```
22 #… is simply the integer 22

22. #The trailing dot makes it a time sequence
{22,23,24,…}

[1] # Content of line 1, so also {22,23,24,…} (Counting
starts at 0)

[1]_ # _ extracts the occurring powers p,q,r,s
of 2,3,5,7 in each number and combines them to
2^p*3^q*5^r*7^s, in the following called reduction,
here: {2,1,24,...}

2520,(9000+[1]) #The binary operator comma divides the
reduction of the constant 2520 by the reduction of the
expression in the round bracket
```

Other important features of the *AOGscript* scripting language:

- **N_0-Paradigma:** If the evaluation result of a row is less than zero, it is set to zero.
- **Fault tolerance:** Lines are evaluated as far as possible. Lines that cannot be evaluated are set to zero.
- **Abandonment of variables:** Instead of using variables, the evaluation result of preceding lines can be retrieved by using the square bracket (see example above).
- **Abandonment of operator priorities:** Operations are processed in their natural order. Exception: Expressions in round brackets are evaluated first.

AOGscript concentrates on the generation of the musical form. Usually a certain number of the lower lines of a script are interpreted in such a way that the integers occurring there in sequence by the evaluation are interpreted as frequencies. Each of these lines is assigned a musical instrument. If a frequency lies within the range of the musical instrument, this frequency is mapped to the nearest pitch of the tempered scale and finally played in real time on the

(virtual) musical instrument. A not inconsiderable part of the software that can interpret an *AOGscript* is also to take care of the interpretive part of the sequences of notes that result from the script, for example, deciding at what tempo and in what sequential repetitions the *ticks* — smallest temporal unit in the progression of the piece and at the same time the tempo at which the natural numbers are counted through in the script — run and with what dynamics and playing technique the instrument is currently playing.

Without a further exhaustive description of *AOGscript* and its implementation as software, an exemplary implementation is provided here. The software available under the following link was implemented with Java/Processing.[1] In order for the corresponding processing sketch to run, the "Contributed Library" "ComposingForEveryone" must still be included. While the four pieces for clarinet and piano use virtual midi instruments (physical modeling), the software provided does not do so, at the cost of a less interesting sonic experience, to ensure that the processing sketch runs as a stand-alone program.[2]

The provided software opens it also to amateurs to compose. This is done here by repeatedly modifying and saving the *AOGscript* in the file *code.txt* while the software is running. This objectifying approach to composing for amateurs with direct feedback (comprovization) can also be seen as a contribution to the movement "Ubiquitous Music", because an alternative way to composing is opened up, which can be followed without the prerequisite of a music degree. Ubiquitous Music is the name of a group of musicians and programmers, especially in Brazil and Portugal, who see computers and the Internet as an opportunity to establish new ways of composing and making music, including for amateurs in an everyday creative practice known there as "little c" (Keller et al. 2014, 29-30).

Opening up the possibility of new kinds of creative practices, as is done here in the form of the provision of *AOGscript*, is one thing. Getting people interested in engaging in such a practice, or even establishing it as a cultural technique for musical laypeople, as is the case in choral societies and playing circles, is quite another. An initial first step to achieve the latter is certainly to awaken a certain interest in it in the first place. The four pages that now follow are to be understood as four conceptual works of art, intended to be printed out and hung up, side by side in a very simple way in an exhibition for reception. At best, they may arouse some amazement that these few lines can each represent an entire musical composition, or stimulate thought about what music is and whether anyone is willing to grant that a generative instruction imagined in its entirety

1. https://processing.org
2. http://www.kramann.info/AOGscript1.zip

is an equivalent substitute for a musical thought. And finally, when received without knowledge of the present text, the four sheets may simply be perceived, on the one hand, as a stimulating puzzle about what connection exists between music and code, and, on the other hand, the hanging may be perceived as a provocation to understand the code itself as an original work of art.

References

Keller, Damián, Victor Lazzarini, and Marcelo S. Pimenta (eds.). 2014. *Ubiquitous music*. Heidelberg: Springer International Publishing.

Kramann, Guido. 2021. "Composing by Laypeople: A Broader Perspective Provided by Arithmetic Operation Grammar." *Computer Music Journal* 44 (1): 17-34.

```
4000./12%29
(5000+[0])./48
(6000+[0])./36
(7000+[0])./24
(8000+[0])./18
9000.*(10)%(5*40*[1])
9000.*(20)%(5*40*[2])
9000.*(30)%(5*40*[3])
9000.*(40)%(5*40*[4])
2*2*2*2*2*2*2*2*2*3*3*3*5*5*7
2*2*2*2*3*3*3*5*7
([9];[5])*5
([9];[6])*4
([9];[7])*3
([9];[8])*2
([10],[5])*(5+([1]*103%12))
([10],[6])*(5+([1]*103%12))
([10],[7])*(5+([1]*103%12))
([10],[8])*(5+([1]*103%12))
[18] < [14]
[17] < [13]
[16] < [12]
[15] < [11]
```

http://www.kramann.info/a.mp3

```
((4320/2)._>(4319/2)?)
[0].+2
[0].+4
[0].+6
[0].+8
[0].+10
[0].+12
[0].+16
[0].+18
[0].+24
[0].+32
[0].+36
[0].+48
[0].+60
[0].+72
[0].+96
[0].+192
([1]/2*(0.%2!):5~)
([2]/3*(0.%3!):4~)
([3]/4*(0.%4!):3~)
([4]/5*(0.%5!):2~)
([5]/6*(0.%6!):10~)
([6]/8*(0.%8!):9~)
([7]/9*(0.%9!):8~)
([8]/10*(0.%10!):6~)
([9]/12*(0.%12!):18~)
([10]/15*(0.%15!):16~)
([11]/16*(0.%16!):15~)
([12]/18*(0.%18!):12~)
([13]/20*(0.%20!):32~)
([14]/24*(0.%24!):25~)
([15]/25*(0.%25!):24~)
([16]/32*(0.%32!):20~)
```

http://www.kramann.info/b.mp3

```
1*(4.%8!)*(0./12%2!)+1
1*(2.%8!)*(6./18%2!)+1
1*(0.%8)*(9./24%2!)+1
1*(0.%2!)*(0./36%2!)+1
1*(1.%2)*(6./60%2!)+1
1*(0.%2!)*(9./84%2!)+1
2*(0.%4)*(0./24%3!)+1
2*(1.%4!)*(6./36%4!)+1
2*(2.%4!)*(9./48%6!)+1
2*(0.%6!)*(0./72%4!)+1
2*(2.%6)*(1./144%6!)+1
2*(3.%6!)*(2./72%8!)+1
4*(0.%12)*(0./16%2!)+1
4*(4.%12!)*(6./32%3!)+1
4*(8.%12)*(9./64%4!)+1
6*(0.%8!)*(0./128%2!)+1
6*(2.%8!)*(4./256%3!)+1
6*(4.%8!)*(8./512%4!)+1
(0./12%3)*15*[3]*[4]*[5]*[10]*[11]*[14]*[17]>120~
(4./12%3)*120*[2]*[3]*[4]*[9]*[10]*[13]*[16]>120~
(8./12%3)*(72+4*(0./36%5))*[1]*[2]*[3]*[9]*[10]*[13]*[16]>120~
(0./12%3)*(18+4*(0./36%5))*[0]*[1]*[2]*[3]*[6]*[7]*[12]*[15]>120~
(4./12%3)*(36+4*(0./36%5))*[1]*[2]*[3]*[4]*[7]*[8]*[13]*[16]>120~
(6./12%3)*(54+4*(0./36%5))*[2]*[3]*[4]*[5]*[9]*[10]*[14]*[17]>120~
```

http://www.kramann.info/c.mp3

```
(9002._):(6._)>60?
(9003._):(12._)>90?
(9012._):(18._)>60?
(9018._):(24._)>90?
(9024._):(30._)>60?
(9036._):(36._)>90?
(9048._ ):(42._)>60?
(9072._):(48._)>90?
[0]*(6.%18!)*4 | (([1]*(1.%2!))*1._>120)
[1]*(12.%24!)._>60 | (([2]*(2.%3!))*6)
[2]*(18.%36!) | ([4]*(27.%72!)*3)  |
(([5]*(4.%24!))._>90*4)
([4]*(4.%18!))._>60 | ([5]*(36.%144!)*6) | ([6]*(6.%36!)*8)
```

http://www.kramann.info/d.mp3