threads

2014-*

Vincent Giles

Introductory Notes

This is a master copy of all existing threads, consolidating earlier works and acting as an introductory/explanatory note.

Individual pieces will have their own typeset files, but this will act as the larger ”score” for the collection and grow with the project.

These pieces began in 2014 when they were called *Logarithm n*, with the intention of building structured or seeded improvisations using pseudo-code. I am interested in the idea that algorithmic processes described using pseudo-code can be interpreted and performed by humans. It is algorithmic music for humans. In 2020 I was given a residency from Lebowskis Music to further develop these works, but then a global pandemic happened and the performances were accordingly adjusted. Since 2018 I had been further developing my actual programming technique in a range of languages, and consequently, the works rendered in 2020 and onwards lean into a somewhat more ”accurate” programming syntax, whilst retaining the elements of pseudo-code explored in earlier works. However, things are pretty loose, syntactically. The pieces were renamed *threads* in 2020 because ”logarithm” is pretty daft, and ”algorithm” is pretty obvious.

In 2020 I became increasingly interested in algorithms beyond computers, largely due to working with the livecoding environment developed by Alex McLean called *TidalCycles.* Alex’s research work explores the notion of algorithms throughout human history, looking at patterns, weaving, and other ”crafts” as following algorithmic processes. Indeed, this was something in my mind in the earlier works but I was unable to articulate that aspect of the intention until 2020.

This collection is a way of re-consolidating and re-typesetting these works into a single, growing collection, and to do away with what was a rambling and fairly incoherent set of instructions for the development of the pseudo-code syntax. Instead, I would encourage any performers or performances to either involve me, or, to lean in to the unknown and take what is written as a prompt, and consider themselves a collaborator in spontaneous, seeded composition.

It is hoped that performers do not feel offput by the use of pseudo-code, but rather lean into the suggestiveness and view the code as art in its own right.

Vincent Giles, July 15, 2020

thread 1

2014
For any number of improvising performers

```
// This is the original 2014 typesetting

music.function { x = (x-x2)+(y+y3) };
music.function.DoUntil;
music.function { x = x+1 };

// 2020 Re-imagining
// This can act as a guide for other interpretation of other early works.

musicFunction() {
    x = (x-x2)+(y+y3);
}
musicFunction().doUntil {
    musicFunction() {
        x = x+1;
    }
}
```

thread 2

2014

For any number of improvising performers

```
// Begin here
var x = sound;
var y = silence;
{
   sound.function(1);
      {
      sound.function.do(a = a+x-y);
      sound.function.doWhile(y/x2+a)=(y3)3;
      sound.function.doUntil(y=x2);
      }
   sound.function(0);
   }
```

thread 3b

2016

For (a small quantity (20-30) of) massed instruments

```
(
// Define variables -zeroed by default

instQuant = 0; // quantity of performers
instFreqn = 0; // frequency
instSim(s) | s = 0 |; // similarity value (a value of 1 means that the
    similarity is high bordering on exact)
instAmpn = 0; // amplitude
instRhyn = 0; // rhythm OR gesture
instFXn = 0; // timbral technique

// The algorithm
set instQuant(n.performers);
process = {
   process.doLoop {
      instQuant(instFreqn, instAmpn, instRhyn, instFXn); // each
          performe independently sets their parameters in real time on
          each "loop"
      process.doLoopWhile(instFreqn(n) != instSim(1));
      process.doLoopUntil (instFreqn(n) == instSim(1), then
          process.stop); }
   }
process.start;
)
```

thread 4

2020

For variable sized ensemble

Developed for Lebowskis DevResIX Residency

```
// define initial conditions (players, each player's sound types, etc.)

var array.player[0,...,n];
var array.soundType[0,...,4];
var array.duration[];
var perfLen = random;

// each player runs their part
part() {
   var player = player[n];
   var sound = soundType.choose;
   var duration = duration.random;
   var play = player.sound.duration;
   pause(duration.random);
}

// conditions for piece iteration
var perfDur = n;
while perfDur < perfLen {
   part().do;
}
```

thread 5

2020

For variable sized ensemble

Developed for Lebowskis DevResIX Residency

```
/* This is a piece for multiple, asynchronous improvising players,
    either in-person, or in total isolation from each other.
In the case of total isolation, the duet must be imaginary for later
    synchronisation and emergence via digital means. */

// Set up initial conditions.

var perfDuration = "Circa 40 minutes";
var duoPartner = ensembleMember.choose;

// Particular player's individual rules
part() {
    var duet = thisPlayer + duoPartner.doNotTell;
}

// The main structure
performance() {
    while (perfDuration < "40 minutes") {
        duet.improvise; // Leave space!
        if (duoPartner != currentlyPlaying) {
            duoPartner = ensembleMember.choose;
        }
    }
}
```

thread 6

2020

For any two performers
Developed for Lebowskis DevResIX Residency

```
/* This piece is for any two performers. */

// Set up initial conditions.

var perfDuration = context.define;
var duoPartner = duoPartner.define;
var thisPlayer = thisPlayer.define;

// The main structure
performance() {
    var range;
    var dur;
    var time;
    while (time < perfDuration) {
        if (duoPartner.sound == TRUE) {
            range = [duoPartner.sound.min, ..., duoPartner.sound.max];
            dur = context.define;
            thisPlayer.improvise(dur) {
                range[].choose;
            }
        }
        else if (duoPartner.sound == FALSE) {
            range = [thisPlayer.sound.min, ..., thisPlayer.sound.max];
            dur = context.define;
            thisPlayer.improvise(dur) {
                range[].choose;
            }
        }
        else {
            thisPlayer.silence;
        }
    }
}
```