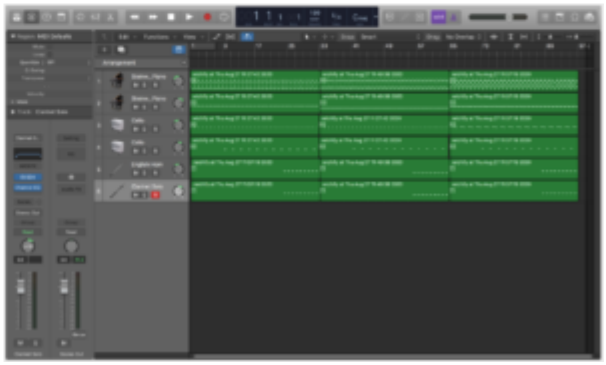
Imitating Steve Reich

2020-10-18 BY GENE



Recently I was enjoying Steve Reich's "Octet" (Eight Lines) and became curious if I could mimic the simple, staccato phrases with which he starts off his piece. tl;dr: reichify

Fortunately I have been immersed in algorithmic composition of late and had a fair idea what tools I would need. Number one is Perl and its vast ecosystem of modules – music ones in particular. Let's go through the code!

```
#!/usr/bin/env perl
use strict;
use warnings;

use Data::Dumper::Compact qw(ddc);
use List::Util qw(shuffle);
use MIDI::Praxis::Variation qw(transposition);
use MIDI::Util;
use Music::Interval::Barycentric qw(cyclic_permutation);
```

```perl
use Music::Scales qw(get_scale_MIDI);
use Music::VoiceGen;
```

Here the standard perl preamble starts things off, followed by a handful of modules with methods and functions to use below. Next up is to define a few crucial parameters (that can be provided on the command-line):

```perl
my $bars  = shift || 32;
my $bpm   = shift || 180;
my $note  = shift || 'B';
my $scale = shift || 'major';
```

Except for the number of bars, these are based on Reich's piece.

So this program uses global variables throughout. This is not a recommended practice for mission-critical, production software! Anyway, one of these globals is a voice generator created from the excellent Music::VoiceGen module:

```perl
my $octave = 2;
my @pitches = (
    get_scale_MIDI($note, $octave, $scale),
    get_scale_MIDI($note, $octave + 1, $scale),
    get_scale_MIDI($note, $octave + 2, $scale),
);
my $voice = Music::VoiceGen->new(
    pitches   => \@pitches,
    intervals => [qw(-4 -3 -2 -1 1 2 3 4)],
);
```

This allows a random pitch to be generated based on three octaves and the given legal interval jumps. Next, a shuffled clarinet motif is defined and a clarinet note array declared:

```perl
my $cmotif = [shuffle qw(dhn en en en en)];
my @cnotes;
```

These will be used below, in the clarinet phrase generators… Next the notes for the piano tracks are generated with this code:

```perl
my $pmotif = [('en') x 10];
my @pnotes;
my @transp;
for my $n (0 .. $#$pmotif) {
    my $note = note_or_rest($n, $pmotif, \@pnotes);
    push @pnotes, $note;
    if ($note eq 'r') {
        push @transp, 'r';
    }
    else {
        my @transposed = transposition(-12, $note);
        push @transp, $transposed[0];
    }
}
```

Here the piano motif is defined as ten eighth notes. Notes are added to the **`@pnotes`** and **`@transp`** arrays in a loop over the **`$pmotif`** and the function **`note_or_rest()`** is called. This function returns – you guessed it – either a note (as a MIDI pitch number) or a rest (as an 'r' character). This note is added to the **`@pnotes`** array, and a transposed version is added to the **`@transp`** array.

Ok. With all those things defined and populated, the next thing is to setup the MIDI stuff:

```
my $volume = 98;
my $pan = 10; # control change number
my $pan_left = 32;
my $pan_right = 86;
my $score = MIDI::Util::setup_score(
    lead_in   => 0,
    signature => '5/4',
    bpm       => $bpm,
    volume    => $volume,
);
```

And synchronize the piano, violin and clarinet parts to play:

```
$score->synch(
    \&piano1,
    \&piano2,
    \&violin1,
    \&violin2,
    \&clarinet1,
    \&clarinet2,
);
```

Finally, as far as the execution of the program goes, a MIDI file, named after the program, is written to disk:

```
$score->write_score("$0.mid");
```

Now for the subroutines! Let's consider the first one we encountered above, `note_or_rest()`:

```perl
sub note_or_rest {
    my ($n, $motif, $notes) = @_;
    if (
        # We're at the end of the motif and the first note is a rest
        ($n == $#$motif && $notes->[0] eq 'r')
        ||
        # The previous note is a rest
        (defined $notes->[$n - 1] && $notes->[$n - 1] eq 'r')
    ) {
        $note = $voice->rand;
    }
    else {
        $note = int(rand 10) <= 3 ? 'r' : $voice->rand;
    }
    return $note;
}
```

Here, a note is generated so that two rests are not in a row. That is what the if condition says basically. Otherwise either a rest or a note is generated based on a probability (i.e. return a rest approximately 40% of the time).

The companion to this function is the following, which actually adds either a rest or notes to the score:

```perl
sub play_note_or_rest {
    my ($motif, $notes) = @_;
    if ($notes->[0] eq 'r') {
        $score->r($motif);
    }
    else {
        $score->n($motif, @$notes);
    }
}
```

In the generation of piano notes, if the first is a rest, so is the second. That is why this code only considers the first element.

All that remains are the subroutines that play the pianos, violins and clarinets – which makes up the majority of the program actually. The first piano looks like this:

```perl
sub piano1 {
    MIDI::Util::set_chan_patch($score, 0, 0);
    $score->control_change(0, $pan, $pan_left);
```

```
        print 'Piano 1.1: ', ddc(\@pnotes);
        print 'Piano 1.2: ', ddc(\@transp);
        for my $i (1 .. $bars) {
            for my $n (0 .. $#$pmotif) {
                play_note_or_rest($pmotif->[$n], [$pnotes[$n],
$transp[$n]]);
            }
        }
    }
```

Here, the MIDI channel and patch are both set to zero. Then the pan is set to the defined left value. The actual notes are shown (with the `ddc()` function of the also excellent Data::Dumper::Compact module). Next the computed notes (or rests) are added to the score in a loop over the defined number of "$bars." For each bar, a note or rest is played for each MIDI duration element of the motif. *Voilà!*

The other subroutines defining the other instruments are *each* different. This is so that the composition is not stale and redundant. But they are left to the reader to explore in the program.

Here is a composition based on three runs of *very very many* that I finally sort-of liked: Reichified II

FILED UNDER: MUSIC, SOFTWARE
TAGGED WITH: MIDI, PERL

Epistemologist-at-large