necklaces

for krystina

for 4 plucked strings

on the notation

in the provided score (read left to right and top to bottom), each cell indicates a picking pattern given by a tablature where each line represents one of 4 strings. the ring around a cell is an alternative representation where the numbers, starting from the number centered above the tablature and moving clockwise, correspond to the strings of the picking pattern.

on realizing the piece

each string should be tuned such that the sounding tones are the same pitch or approximately the same pitch with very slight detunings. the strings may be played open or stopped. for example, if played with open strings, the strings themselves are tuned to (approximately) the same pitch. the piece is constant throughout: rhythmically and dynamically. the tempo should be comfortably fast such that the picking is as uniform as possible. each cell should be repeated at least 5 times with shorter cells repeated more. the piece as a whole can be played any number of times in performance without pause.*

the piece has three primary variations: 1) as a solo plucked strings piece; 2) accompanied with my work *quieting rooms*;** or 3) accompanied with a piano tone of some lower octave-equivalent as the strings repeated at the same tempo with the pedal always down. For the latter two, the accompaniment starts well before the entrance of the strings. with *quieting rooms*, the electronics continue after the plucked strings; whereas with piano, the two players end together allowing the tones to naturally decay.

* on the musical and mathematical construct

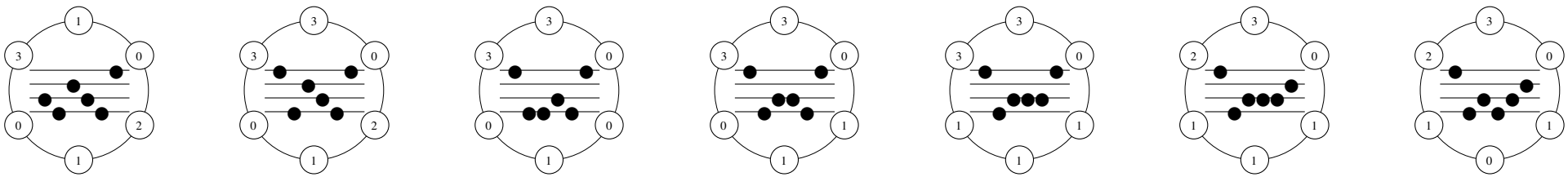
while the piece as a whole may be repeated verbatim, if played multiple times successively, it is preferred that new versions are generated observing the following construction.

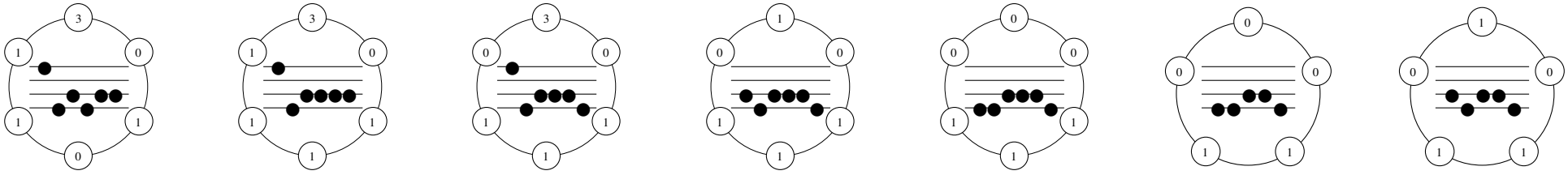
musically speaking, the cells in the provided score represent all possible unique picking patterns of 4 strings sounding the same pitch. such limited focus accentuates minor variations in tuning, string tension, and string gauge. the patterns are derived from unique aperiodic necklaces of size 6 or less with 4 symbols. the symbols are given abstractly in the score by the numbers in the ring around each cell which represent the strings of the instrument for the purposes of the piece. two patterns are considered equivalent if one can be rotated and/or have the symbols permuted to form the other. for example, under rotation, {1, 2, 3, 4} is equivalent to {3, 4, 1, 2}; and under permutations of the symbols, {1, 2, 2, 1} is equivalent to {3, 4, 4, 3}. a set of equivalent patterns form what is called an equivalency class. the morphology (order) of the patterns is generated by finding a hamiltonian cycle on a graph (a path where each vertex is passed once and only once and returns to the start vertex) such that the vertices are unique patterns of size 2 to 6 and edges are induced between two patterns that either add, remove, or swap an element. then, the trivial case of a single repeated tone is prepended and appended to the above morphology. note that the graph will vary based on which patterns are chosen from each equivalency classes and not all the graphs may contain a hamiltonian cycle.

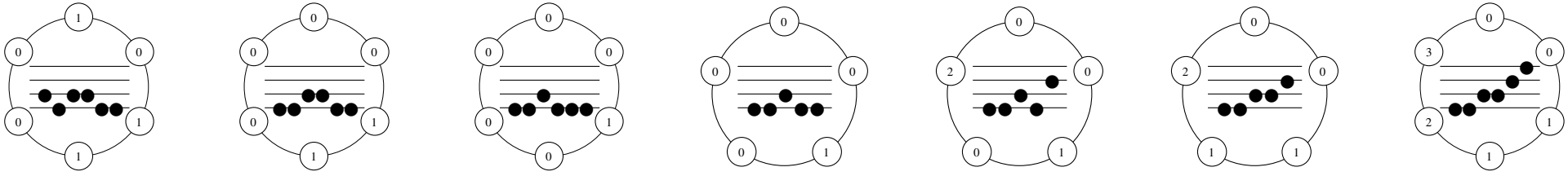
new morphologies can be made upon request and will be added to the score as they are generated.

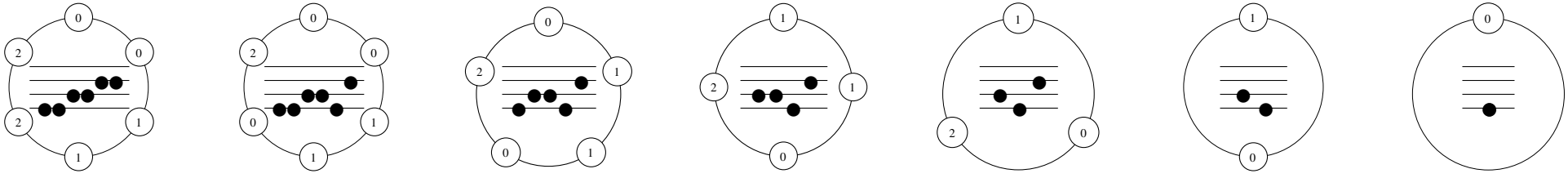
** on *quieting rooms*

quieting rooms was originally conceived as an installation. when the piece is played with *necklaces*, it works well to start the computer program before the event allowing it to run as people enter and program *necklaces* first. *quieting rooms* is a genetic algorithm that attempts to put to signals out of phase. the original version requires two speakers to face each other to ensure proper cancellation; however I realize that for the purposes of the simultaneous playing of *quieting rooms* with *necklaces*, that the process alone can work with any number of speakers where the cancellation and even the amplitude measurements are made digitally within the computer program. that is, the very process of the algorithm functions musically regardless of whether or not the cancellation is produced acoustically. while the original version is perfectly acceptable, modified versions of the computer program, one where the cancellation is digital and one where both the cancellation and amplitude measurement are digital, are included with this score.


1 3 0 0 2 1
3 3 0 0 2 1
3 3 0 0 0 1
3 3 0 0 1 1
3 3 0 1 1 1
3 2 0 1 1 1
3 2 0 1 1 0


3 1 0 1 1 0
3 1 0 1 1 1
3 0 0 1 1 1
1 0 0 1 1 1
0 0 0 1 1 1
0 0 0 1 1
1 0 0 0 1 1


1 0 0 0 1 1
0 0 0 0 1 1
0 0 0 0 1 0
0 0 0 1 0
2 0 0 1 0
2 0 0 1 1
3 0 0 2 1 1


0 2 0 2 1 1
0 2 0 0 1 1
0 2 1 0 1
1 2 1 0
1 2 0
1 0
0

```
//quieting rooms by michael winter. necklaces version with digital cancellation. Supercollider 3.6.6. gpl.
(
var initialize, mutate, listener, banks, monitor,
generation, last_mutation, last_reset,
init_freqs1, init_amps1, init_phases1,
init_freqs2, init_amps2, init_phases2,
current_phases, fittest_phases, fittest_fitness,
size = 10, //number of sine tones
generation_rate = 2; //number of generations per second

//this synthdef is two oscillator banks in one synth as to keep the phases locked.
SynthDef.new("double_osc_bank",
     {
          var freqs1, freqs2, amps1, amps2, phases1, phases2, bank1, bank2;

          freqs1 = Control.names([\freqs1]).kr(Array.rand(size, 30.0, 1000.0));
          amps1 = Control.names([\amps1]).kr(Array.rand(size, 0.0, 1.0));
          phases1 = Control.names([\phases1]).kr(Array.rand(size, 0.0, 2.0pi));

          freqs2 = Control.names([\freqs2]).kr(Array.rand(size, 30.0, 1000.0));
          amps2 = Control.names([\amps2]).kr(Array.rand(size, 0.0, 1.0));
          phases2 = Control.names([\phases2]).kr(Array.rand(size, 0.0, 2.0pi));

          bank1 = DynKlang.ar(`[ freqs1, amps1, phases1], 1, 0) * 1.0/size;
          bank2 = DynKlang.ar(`[ freqs2, amps2, phases2], 1, 0) * 1.0/size;

          //this is where the two banks are summed for digital cancellation; add more outputs for more speakers
          Out.ar(0, bank1 + bank2);
          Out.ar(1, bank1 + bank2);

      }).send(s);

//this is the synthdef for monitoring the volume of the room using two or less mics/inputs
SynthDef.new("monitor",
     {SendReply.ar(Impulse.ar(generation_rate), '/value', RunningSum.rms(SoundIn.ar(0) + SoundIn.ar(1), s.sampleRate/generation_rate/2 ));
     }).send(s);

initialize =
     {
          generation = 0;
          last_mutation = 0; //this is the number of generation since a successful/fitter mutation.
          last_reset = 0; //this is the number of times the algorithm has been allowed to climb out of a local optimum.
          init_freqs1 = Array.rand(size, 30.0, 700.0); init_freqs2 = init_freqs1; init_amps1 = Array.rand(size, 0.0, 1.0);
          init_amps2 = init_amps1; init_phases1 = Array.rand(size, 0.0, 2.0pi); init_phases2 = Array.rand(size, 0.0, 2.0pi);
          current_phases = init_phases2; fittest_phases = init_phases2;
          banks.setn(\freqs1, init_freqs1); banks.setn(\amps1, init_amps1); banks.setn(\phases1, init_phases1);
          banks.setn(\freqs2, init_freqs2); banks.setn(\amps2, init_amps2); banks.setn(\phases2, init_phases2);
          fittest_fitness = 100; //set very high at first
     };

mutate =
     {
          arg current_fitness;
          var mutation_size, range;

          generation = generation + 1;

          //print statements
          ("--------------").postln;
          ("generation\t fittest fitness\t current fitness").postln;
          (generation + "\t\t\t" + fittest_fitness + "\t" + current_fitness).postln;
```

```
          ("fittest phases\n" + fittest_phases).postln;
          ("current phases\n" + current_phases).postln;

          //if last set of phases made the room quiter mutate those
          fittest_phases = if (current_fitness < fittest_fitness, current_phases.copy, fittest_phases.copy);
          last_mutation = if (current_fitness < fittest_fitness, 0, last_mutation + 1);
          fittest_fitness = if (current_fitness < fittest_fitness, current_fitness, fittest_fitness);

          //more print statements
          ("last mutation =" + last_mutation).postln;
          ("last reset =" + last_reset).postln;

          //this allows an evolution to get out of a local optimum. level of the room is very much effected by people, doors and window.
          //this should account for that.
          if (last_mutation >= 50,
               {fittest_phases = current_phases.copy;
               fittest_fitness = current_fitness;
               last_reset = last_reset + 1;
               last_mutation = 0;
               },{});

          //this starts a new evolution if the above statement has occured a certain number of times.
          if (last_reset >= 10,
               {initialize.value},
               {
                    current_phases = fittest_phases.copy;
                    mutation_size = ceil(pow(1.0.rand, 5) * size);//number of phases mutated can go from 1 to size but with a bias towards smaller mutations.

                    ("mutation size =" + mutation_size).postln; //yet another print statement.

                    //pick some phases and mutated them by 0.1pi.rand2.
                    //tweaking this determines with mutation_size how drastic the 2nd signal changes from one generation to another.
                    range = Array.new[size];
                    size.do({arg item; range = range.add(item)});
                    mutation_size.do({var c = range.choose; range.remove(c); current_phases[c] = current_phases[c] + 0.1pi.rand2});
                    banks.setn(\phases2, current_phases);
               });

     };

//trigger.
listener =
     {
          OSCFunc(
          {
               arg msg;
               mutate.value(msg[3]);
          }, '/value');
     };

//run.
Routine {
     1.0.wait;
     banks = Synth.new("double_osc_bank");
     monitor = Synth.new("monitor");
     initialize.value;
     (1/generation_rate).wait;
     listener.value;
}.play;
)
```

```
//quieting rooms by michael winter. necklaces version with digital cancellation and monitoring. Supercollider 3.6.6. gpl.
(
var initialize, mutate, listener, banks, monitor,
generation, last_mutation, last_reset,
init_freqs1, init_amps1, init_phases1,
init_freqs2, init_amps2, init_phases2,
current_phases, fittest_phases, fittest_fitness,
size = 10, //number of sine tones
generation_rate = 2, //number of generations per second
listener_bus = Bus.audio(s, 1); //create a listener bus for digital monitoring

//this synthdef is two oscillator banks in one synth as to keep the phases locked.
SynthDef.new("double_osc_bank",
     {
          var freqs1, freqs2, amps1, amps2, phases1, phases2, bank1, bank2;

          freqs1 = Control.names([\freqs1]).kr(Array.rand(size, 30.0, 1000.0));
          amps1 = Control.names([\amps1]).kr(Array.rand(size, 0.0, 1.0));
          phases1 = Control.names([\phases1]).kr(Array.rand(size, 0.0, 2.0pi));

          freqs2 = Control.names([\freqs2]).kr(Array.rand(size, 30.0, 1000.0));
          amps2 = Control.names([\amps2]).kr(Array.rand(size, 0.0, 1.0));
          phases2 = Control.names([\phases2]).kr(Array.rand(size, 0.0, 2.0pi));

          bank1 = DynKlang.ar(`[ freqs1, amps1, phases1], 1, 0) * 1.0/size;
          bank2 = DynKlang.ar(`[ freqs2, amps2, phases2], 1, 0) * 1.0/size;

          //this is where the two banks are summed for digital cancellation; add more outputs for more speakers
          //the summed banks are also sent to the listener_bus for monitoring
          Out.ar(0, bank1 + bank2);
          Out.ar(1, bank1 + bank2);
          Out.ar(listener_bus, bank1 + bank2);

      }).send(s);

//this is the synthdef for monitoring the amplitude.
SynthDef.new("monitor",
     {SendReply.ar(Impulse.ar(generation_rate), '/value', RunningSum.rms(In.ar(listener_bus), s.sampleRate/generation_rate/2))}).send(s);

initialize =
     {
          generation = 0;
          last_mutation = 0; //this is the number of generation since a successful/fitter mutation.
          last_reset = 0; //this is the number of times the algorithm has been allowed to climb out of a local optimum.
          init_freqs1 = Array.rand(size, 30.0, 700.0); init_freqs2 = init_freqs1; init_amps1 = Array.rand(size, 0.0, 1.0);
          init_amps2 = init_amps1; init_phases1 = Array.rand(size, 0.0, 2.0pi); init_phases2 = Array.rand(size, 0.0, 2.0pi);
          current_phases = init_phases2; fittest_phases = init_phases2;
          banks.setn(\freqs1, init_freqs1); banks.setn(\amps1, init_amps1); banks.setn(\phases1, init_phases1);
          banks.setn(\freqs2, init_freqs2); banks.setn(\amps2, init_amps2); banks.setn(\phases2, init_phases2);
          fittest_fitness = 100; //set very high at first
     };

mutate =
     {
          arg current_fitness;
          var mutation_size, range;

          generation = generation + 1;

          //print statements
          ("--------------").postln;
```

```
          ("generation\t fittest fitness\t current fitness").postln;
          (generation + "\t\t\t" + fittest_fitness + "\t" + current_fitness).postln;
          ("fittest phases\n" + fittest_phases).postln;
          ("current phases\n" + current_phases).postln;

          //if last set of phases made the room quiter mutate those
          fittest_phases = if (current_fitness < fittest_fitness, current_phases.copy, fittest_phases.copy);
          last_mutation = if (current_fitness < fittest_fitness, 0, last_mutation + 1);
          fittest_fitness = if (current_fitness < fittest_fitness, current_fitness, fittest_fitness);

          //more print statements
          ("last mutation =" + last_mutation).postln;
          ("last reset =" + last_reset).postln;

          //this allows an evolution to get out of a local optimum. level of the room is very much effected by people, doors and window.
          //this should account for that.
          if (last_mutation >= 50,
               {fittest_phases = current_phases.copy;
               fittest_fitness = current_fitness;
               last_reset = last_reset + 1;
               last_mutation = 0;
               },{});

          //this starts a new evolution if the above statement has occured a certain number of times.
          if (last_reset >= 10,
               {initialize.value},
               {
                    current_phases = fittest_phases.copy;
                    mutation_size = ceil(pow(1.0.rand, 5) * size);//number of phases mutated can go from 1 to size but with a bias towards smaller mutations.

                    ("mutation size =" + mutation_size).postln; //yet another print statement.

                    //pick some phases and mutated them by 0.1pi.rand2.
                    //tweaking this determines with mutation_size how drastic the 2nd signal changes from one generation to another.
                    range = Array.new[size];
                    size.do({arg item; range = range.add(item)});
                    mutation_size.do({var c = range.choose; range.remove(c); current_phases[c] = current_phases[c] + 0.1pi.rand2});
                    banks.setn(\phases2, current_phases);
               });
     };

//trigger.
listener =
     {
          OSCFunc(
          {
               arg msg;
               mutate.value(msg[3]);
          }, '/value');
     };

//run.
Routine {
     1.0.wait;
     banks = Synth.new("double_osc_bank");
     monitor = Synth.tail(s, "monitor");
     initialize.value;
     (1/generation_rate).wait;
     listener.value;
}.play;
)
```