quieting rooms
for tashi wada

short description (use this for an accompanying note to the installation):

in this installation, a very crude genetic algorithm (i.e., a model of darwinian evolution) attempts to put two signals out of phase and quiet the room.

this particular algorithm does not model sex. each generation has a population of one which results from mutating the fittest/quietest generation up to that point in the evolution/life-cycle. the minor variations heard are the mutations from generation to generation.

in a successful life-cycle, the signals learn to be out of phase and the room gets quieter. when the room does not get quieter for an extended period of time, a new life-cycle begins.

slightly longer description (describing setup):

in this installation, a very crude genetic algorithm (i.e., a model of darwinian evolution) attempts to put two signals out of phase and quiet the room.

two speakers face each other. at the beginning of an evolution/life-cycle, one speaker is initialized to output a set of sine tones with random frequencies, amplitudes, and phases. the other speaker is initialized to output the same number of sine tones with the same frequencies and amplitudes as the first speaker, but with different phases. the output of the first speaker remains constant throughout a life-cycle while the output of the second speaker is mutated by varying some subset of the phases.

this particular algorithm does not model sex. each generation has a population of one which results from mutating the fittest/quietest generation up to that point in the evolution/life-cycle. each generation is tested by a set of microphones used to measure the volume of the room. the minor variations heard are the mutations from generation to generation.

in a successful life-cycle, the signals learn to be out of phase and the room gets quieter. when the room does not get quieter for an extended period of time, a new life-cycle begins.

considerations (after the first realization):

the first realization was presented at ArtShare in los angeles on december 13, 2012. the room was approximately 15 by 15 feet. initial considerations are provided below as a starting point for further exploration.

setting:

a dimly lit, rather dark room seems conducive to focused listening of the actually process. for the first realization, a window was left open, this minimized the reverberation of the space. theoretically, the algorithm should work better in dry/less-echoic spaces so that the measurement of the level of the room is of the current generation only and not does not include any reverberation or resonance from any prior generation. perhaps there is a method to account for the reverberation of the space.

spectral density:

the first realization used 10 sine tones between 30 and 1000 hertz out of each speaker. after much testing it seemed to be reasonably effective perceptually.

evolutionary rate:

for each generation/mutation, the algorithm implemented for the first realization uses four factors that determine the evolutionary rate and length of a life-cycle. 1) the number of phases in the second signal to shift. 2) the size of the shift for each phase. 3) the number of generations of unsuccessful mutations that must occur before stepping out of a local optimum. 4) the number of times a local optima has been determined to be reached (according to 3) before a new life-cycle begins.

theoretically, in an ideal setting, if the second of the factors above is 2pi and factors 3 and 4 are ignored (i.e., not implemented), then a global optima should be reached. however, minimizing the size of the phase shift in a mutation and biasing smaller mutations seems to allow for a more linear and gradual quieting of the room.

these factors of the installation as they relate to our perception were and should be the primary consideration in realizing the piece. during the conception of this work, gregory chaitin’s book Proving Darwin was hugely influential.

-michael winter (los angeles, 2012)

```
//quieting rooms by michael winter. first realization. Supercollider 3.5.6. 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;

		Out.ar(0, bank1);
		Out.ar(1, bank2);

	 }).send(s);

//this is the synthdef for monitoring the volume of the room.
SynthDef.new("monitor",
	{
		SendReply.ar(Impulse.ar(generation_rate), '/value', RunningSum.rms(SoundIn.ar(0) , 41000/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, 1000.0); 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;
					//number of phases mutated can go from 1 to size but with a bias towards smaller mutations.
					mutation_size = ceil(pow(1.0.rand, 5) * size);

					("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;
	)
```