to hum and walk

for voice and computer

commissioned by carmina escobar

brief description (that can be used as a program note):

the title *to hum and walk* refers to its process: a walk in harmonic space sonified by a vocalist activating digital resonators. to start, a digital resonator is instantiated with a resonant frequency that is an octave equivalent of unity (1/1 or the fundamental) in a 17-limit, multidimensional harmonic space. throughout, the vocalist sings long tones always favoring unity. once the amplitude of the first resonator exceeds a threshold, a furthering of the walk is triggered: several more resonators are instantiated with resonant frequencies closely related in harmonic space to the triggered resonator. each new resonator has a lifespan. if its amplitude exceeds the threshold within that span, the walk continues. The entropic tendency of the random walk is constantly counteracted as the vocalist provides a gravity towards unity and because immediate movement in the walk is constrained to a limited distance in harmonic space. as a result, the electronics continually reharmonize the tones sung by the vocalist resulting in a slow, sometimes almost chorale-like, harmonic movement.

performance instructions:

the vocalist sings long tones throughout repeating the phrase “why am i so tired”. each tone is the length of a long breath such that only one word per breath is articulated without any melismas. the vocalist should generally choose from the first 5 pitches of a minor scale built on a (a, b, c, d, e) favoring the fifth, the third, and especially the tonic. however, the vocalist can occasionally deviate from the set and choose pitches that are sounding from the digital resonators. regardless, the vocalist must always return to tonic with relative frequency while avoiding repeated sequences of tones. a performance of the piece should last at least 10 minutes and preferably more. the texture should remain rather dense throughout and the vocalist should attempt to minimize (particularly rapid) fluctuations in density.

computer program:

included with this score is a computer program written in supercollider that performs the digital process described in the description above. the program takes as input a few variables controllable by the performer.

*target number of resonators: the performer can set this between 5 and 15. a higher number will result in a denser texture. if the current number of resonators is greater than the target number of resonators, only one new resonator will be instantiated when the amplitude of a currently active resonator exceeds the threshold. otherwise, anywhere from 1 to 4 resonators will be instantiated.

*threshold: this is the amplitude level that a resonator must exceed in order for the walk to continue. it may be set manually and/or automatically. if set automatically (by toggling the auto-adjust button of the user interface), the threshold periodically decreases when the current number of resonators is less than the target number of resonators and increases otherwise.

*resonator amplitude and on/off toggle: this sets the level of the resonators and is preferably not adjusted during performance.

*flourish amplitude and on/off toggle: when this function is turned on, a series of sine tones are generated when the amplitude of a resonators exceeds the threshold. this is an optional addition to any newly instantiated resonators and thus not mentioned above. the frequencies of the sine tones are harmonics of a fundamental derived (by octave transpositions) from the resonant frequency of the triggering resonator. if the vocalist chooses to enable this function, they should turn it on towards the middle of the performance and keep it on for an amount of time that is relatively short and ephemeral with respect to the duration of the performance as a whole.

-michael winter (los angeles, 2014)

```
//to hum and walk by michael winter. Supercollider 3.6.6. gpl.
(
//variables
var walk, count, releasedCount, thresh, target, resonate, flourish, resAmp, flourishAmp, resMute, flourishMute, threshUpdateRoutine,
guiUpdateRoutine, win, resLabel, resText, threshLabel, threshText, threshSlider, autoAdjustButton, autoAdjustLabel, targetLabel, targetText,
targetSlider, resAmpLabel, resAmpText, resAmpSlider, resMuteButton, flourishAmpLabel, flourishAmpText, flourishAmpSlider, flourishMuteButton;

//resonator definition
SynthDef(\resonator, {|freq = 220 thresh = 1 amp = 0.05| var tim, env, in, res, threshTrig, relTrig;
    SendTrig.kr(Impulse.kr(0), 0); //triggers count+1 and releasedCount+1
    relTrig = (LocalIn.kr(1) - 1).abs * Trig.kr(Impulse.kr(0), TRand.kr(10, 15, 1));
    env = EnvGen.kr(Env.asr(TRand.kr(1, 3, 1), 1, TRand.kr(10, 30, 1), 'sine'), relTrig, doneAction: 2);
    in = SoundIn.ar([0,1]);
    res = LPF.ar(Limiter.ar(CombC.ar(in, 0.1, freq.reciprocal, 1000, 1), 1, 0.01), 880);
    threshTrig = thresh < Amplitude.kr(res, 0.1, 0.5);
    threshTrig = Gate.kr(threshTrig, threshTrig);
    SendTrig.kr(TDelay.kr(threshTrig, 0.1), 1, freq); //triggers new resonators
    Out.ar([0,1], res * amp * env);
    LocalOut.kr(TDelay.kr(threshTrig, TRand.kr(10, 15)));
    SendTrig.kr(relTrig <= 0, 2, freq); //triggers releasedCount-1
    SendTrig.kr(Done.kr(env), 3); //triggers count-1
}).add;

//enveloped sine tone definition
SynthDef(\sine, {|fund = 220 harm = 1 amp = 0.1 minDur = 1.0 maxDur = 3.0|
    Out.ar([0, 1],
        amp * SinOsc.ar(fund * harm, 0, EnvGen.ar(Env.sine(1, 1/harm), Impulse.kr(0), timeScale:TRand.kr(minDur, maxDur), doneAction: 2)));
}).add;

//psuedo-random walk on harmonic lattice
walk = {|fund size| var mults = [], freqs = [], minFreq = 50, maxFreq = 880,
    dims = [3, 5, 7, 11, 13, 17], dimProbs = (pow(dims, 3)).reciprocal.normalizeSum,
    steps = {|i| i + 1} ! 3, stepProbs = (pow(steps, 3)).reciprocal.normalizeSum;
    while({mults.size < size}, {
        var mult = 1, curDims = all {: [d, d.reciprocal].choose, d <- dims };
        for(0, steps.wchoose(stepProbs), {mult = mult * curDims.wchoose(dimProbs)});
        if(mults.includes(mult).not, {mults = mults.add(mult)})});
    mults.do({|mult| var freq = fund * mult, equivs = [];
        while({freq > minFreq}, {freq = freq/2.0});
        while({freq*2.0 < maxFreq}, {freq = freq*2.0; if(freq >= minFreq, {equivs = equivs.add(freq)})});
        freqs = freqs.add(equivs.choose)});
    freqs};

//more helper definitions and functions
count = 0; releasedCount = 0; thresh = 0.75; target = 8; resAmp = 0.05; flourishAmp = 0.1; resMute = 0; flourishMute = 0;
resonate = {|freq| SystemClock.sched(0.5.rand, { Synth(\resonator, [\freq, freq, \thresh, thresh, \amp, resAmp * resMute])})};

flourish = {|freq|
    var fund, harms, probs, size, minDur, maxDur;
    if(flourishMute == 1, {
        fund = freq; while({fund > 100}, {fund = fund/2}); fund = fund * 2;
        harms = {|i| i + 1} ! 16; probs = (pow(harms, 2)).reciprocal.normalizeSum; size = harms.wchoose(probs);
        harms = harms.scramble[0..size-1];
        minDur = (size / 16.0 - 1.0).abs * 2.9 + 0.1; maxDur = (size / 16.0 - 1.0).abs * 7.0 + 2.0;
        harms.do({|harm|
            SystemClock.sched(5.0.rand, {Synth(\sine, [\fund, fund, \harm, harm, \amp, flourishAmp, \minDur, minDur, \maxDur, maxDur])})})})};
```

```
//routines
threshUpdateRoutine = Routine({ arg inval;
    loop {if(autoAdjustButton.value == 0, {
        if(count < target, {thresh = (thresh - 0.001).clip(0.1, 1)}, {thresh = (thresh + 0.001).clip(0.1, 1)});
        threshSlider.value = (thresh - 0.1)/0.9;
    }); 0.5.yield}
});
guiUpdateRoutine = Routine({ loop {resText.string = count; threshText.string = thresh; targetText.string = target; resAmpText.string = resAmp;
flourishAmpText.string = flourishAmp; 0.1.yield}});

//response from resonator to trigger walk
OSCFunc({|msg| var trig;
    switch(msg[2],
        0, {count = count + 1; releasedCount = releasedCount + 1},
        1, {if(count < 30, {walk.value(msg[3], if(count < target, {4.rand.trunc}, {1}).rand + 1).do({|freq| resonate.value(freq)});
flourish.value(msg[3])})},
        2, {releasedCount = releasedCount - 1; if(releasedCount == 0, {resonate.value(msg[3])})},
        3, {count = count - 1});
}, '/tr', s.addr);

//start
resonate.value(220); AppClock.play(threshUpdateRoutine); AppClock.play(guiUpdateRoutine);

//gui
win = Window.new("to hum and walk", Rect(128, 64, 1200, 800));
resLabel = StaticText(win, Rect(10, -250, 700, 600)).string_("current # of resonators: ").align_(\left).font_(Font("Courier-Bold", 50));
resText = StaticText(win, Rect(10, -150, 600, 600)).string_(count).align_(\left).font_(Font("Courier-Bold", 150));
threshLabel = StaticText(win, Rect(10, -25, 600, 600)).string_("threshold: ").align_(\left).font_(Font("Courier-Bold", 50));
threshText = StaticText(win, Rect(10, 150, 425, 600)).string_(thresh).align_(\left).font_(Font("Courier-Bold", 150));
targetLabel = StaticText(win, Rect(10, 260, 600, 600)).string_("target # of resonators: ").align_(\left).font_(Font("Courier-Bold", 50));
targetText = StaticText(win, Rect(10, 400, 560, 600)).string_(target).align_(\left).font_(Font("Courier-Bold", 150));
autoAdjustButton = Button(win, Rect(10, 300, 30, 30)).states_([["", Color.black, Color.green],["", Color.black, Color.red]]);
autoAdjustLabel = StaticText(win, Rect(45, 300, 500, 30)).string_("auto adjust threshhold on/off").align_(\left).font_(Font("Courier-Bold", 20));
threshSlider = Slider(win, Rect(10, 340, 600, 30)).value_((0.75 - 0.1)/0.9).action_({|v| thresh = v.value * 0.9 + 0.1});
targetSlider = Slider(win, Rect(10, 590, 600, 30)).value_((8 - 5)/10).action_({|v| target = (v.value * 10 + 5).trunc});
resAmpLabel = StaticText(win, Rect(800, 0, 140, 100)).string_("resonators").align_(\left).font_(Font("Courier-Bold", 25));
resAmpText = StaticText(win, Rect(800, 25, 55, 100)).string_(0.05).align_(\left).font_(Font("Courier-Bold", 25));
resAmpSlider = Slider(win, Rect(800, 100, 30, 600)).value_(0.05).action_({|v| resAmp = v.value});
resMuteButton = Button(win, Rect(800, 710, 30, 30)).states_(
    [["", Color.black, Color.red],["", Color.black, Color.green]]).action_({|v| resMute = v.value});
resMuteButton.valueAction = 1;
flourishAmpLabel = StaticText(win, Rect(1000, 0, 140, 100)).string_("flourish").align_(\left).font_(Font("Courier-Bold", 25));
flourishAmpText = StaticText(win, Rect(1000, 25, 55, 100)).string_(0.05).align_(\left).font_(Font("Courier-Bold", 25));
flourishAmpSlider = Slider(win, Rect(1000, 100, 30, 600)).value_(0.05).action_({|v| flourishAmp = v.value});
flourishMuteButton = Button(win, Rect(1000, 710, 30, 30)).states_([["", Color.black, Color.red],["", Color.black, Color.green]]).action_({|v|
flourishMute = v.value});

win.front;
s.meter();
)
```