Idea:
As someone who loves listening to the drum tracks/grooves of hip-hop bangers and funk jams, I wanted to see how I could use computer programming to simulate a real drummer drumming. A great part about drum machines is their ability to be extremely precise - they will always play a programmed beat in perfect time and without a single mistake. However, this precision is very uncharacteristic of real human drummers, as metronome-perfect drumming is extremely challenging and unrealistic in a live music setting. Therefore, I wanted to find a way to create drum grooves with imperfect, and in some sense unpredictable, timing.

Another goal I had was to create something with programming which could play its own original beat. Like a drummer, the program would use its own notions of what exactly makes up a "beat" to create a new musical idea that might fit perfectly into a specific piece of music. I thought original beat creation would also be useful for rhythmic inspiration - as, for example, a program like this could help a music producer get out of writer’s block.

Design:
Deviating from Perfect Time:
In order to simulate a human drummer, I first collected some data on my own timing deviations (as I have experience drumming myself). To do so, I collected data in the following manner:

1. Listen to an 80 bpm metronome for four beats
2. Turn off the metronome and hit four beats on the website: all8.com/tools/bpm.htm
3. Record the average tempo of those four beats calculated by the metronome

Here is the data I collected:
From my data, I could see pretty clearly that the data followed a mostly normal looking distribution, so I calculated my sample’s mean and standard deviation, in terms of percent difference. My calculation for the mean gave me $E(X) = 0.0055$ percent difference from 80bpm (I chose percent difference as this would allow myself to use other tempos in my program - this is all under the assumption that percent difference in timing remains somewhat constant at different tempos). To calculate my sample variance, I used this value, along with a calculated value of $(E(X^2)) = 0.000475$, to get an estimated variance of $S^2 = ((E(X^2)) - (E(X))^2) = (0.000475 - 0.0055^2) = 0.000445$. So my standard deviation for a group of 4 beats was $\sqrt{0.000445} = 0.0211$ percent difference from the mean tempo.

To get the standard deviation of one hit, I multiplied the sample standard deviation of four hits by $\sqrt{4} = 2$, as a 4 "beat" sample is four times the sample size of one beat. This gave me a predicted percent difference standard deviation in tempo of 0.0421. This is the value I would use in my chucK (a music creation programming language) script to simulate random beat offsets.

I simulated pulling from a normal distribution through randomly picking an index of a 21 element array, as shown below in my code, as chucK does not have any libraries for normal distributions and because I wanted to create these deviations in "real time", while the script itself was running and
creating sound, like a real life drummer would. Each element in the 21 long array corresponds to an approximate z-score for a p-value in a normal distribution of .05*(index). All the math for keeping the beat centralized on the actual "metronome" of the beat is shown in the chucK code below.

Creating New Drum Beats From a Sample:

In order to create a program that generates beats, I first needed a sample of actual drum beats for the program to reference. To do so, I created a file called sampleBeats and put in drum beats from a classic drum book: A Funky Primer for the Rock Drummer by Charles Dowd as well as some classic funk beats from my favorite recordings. To make beat creation more focused and more manageable with a relatively small sample size, I classified beats solely with their bass and snare drum patterns, keeping hihat beats at a constant eight note pace and classifying snare and bass drum hits into a "linear"(only one of bass and snare can hit at a time) sixteenth note pattern. Each drum beat is a loop of 4 quarter notes. SampleBeats is recorded at the end of the document.

It was a challenge to come up with an algorithm to create a new drum beat, as my sample size was relatively small for calculating relevant statistics for 32 values (16 unique notes for both snare and bass) and it is hard to numerically determine what is actually "musical". I eventually decided to create an algorithm, shown in my python code below, that first calculates arrays of "prior" probability values - that is, within my samples, the probability that an instrument hits at a specific time. (This was under the artistic assumption that certain snare hits, bass hits, or rests simply sound better at certain times in a drum beat.) The program randomly chooses a sixteenth note to begin the beat’s creation, and bases the probability of that note’s value (snare hit, bass hit, or no hit) on the prior arrays’ values. Then, if a random value from 0 to 1 is less than the prior value of that instrument hitting at that specific note, then the program accepts that note to be played. If both the snare and bass random values pass these criteria, the program flips a coin to see which drum should hit at that note in order to preserve linear drumming in the created drum beat.

After it decides the value of the first sixteenth note, the algorithm proceed in linear order from that random note. Meaning, it then calculates a conditional probability of the previous note being its exact value for a bass drum or snare drum hit in the next note (I did this under the artistic assumption that drum hits in beats both rely on the frequency of hits at that specific time - the prior - and the sound or lack of sound that occurs before the note - the conditional). The reasoning for these conditional probabilities being based simply on one note directly before instead of instead a larger amount of notes was again from my sample size - as it would be all too common of an occurrence to find no recorded beat with very specific note placements. I calculated these conditional probability values by searching for beats in sampleBeats that have the snare, bass drum, or neither hit at the previous note, and then finding how many of those arrays have a specific hit in the following note. The overall probability it uses to determine the value of that note (in a manner symmetric to the one explained above), would then be the average of the prior value and this conditional probability value. It loops this process until all notes have been determined.

Creating Music:
In the screen recording, I generate 10 beats with my beat creation method, and then generate a pseudo-
song for fun with an emphasis on the created beat. The melody is hard coded and the drum beat in the song is the last of the 10 created beats. The song itself will have the effect of live "timing offsets" for each instrument in the drum beat, as well as for the melody instruments.

The Future:
I wholeheartedly enjoyed bringing a more logical and statistical mindset into musical creation, and I really hope to create more music with these ideas in the future. Hopefully, with more time and more sampling, I can attempt to generate whole songs that demonstrate both musical ideas that have inspired joy in the past and also new combinations of ideas that bring something refreshing to music.

Code on the Following Pages
import random

bass_beats = []
snare_beats = []

# reading beats from the sample and putting them in the arrays
with open("sampleBeats.txt") as file:
    beat = ""
    beat = file.readline()
    while(beat):
        bass_beat = [0] * 16
        bass_index = 0
        snare_beat = [0] * 16
        snare_index = 0
        for i in range(0, len(beat)):
            if beat[i] == '0':
                if bass_index != 16:
                    bass_beat[bass_index] = 0
                    bass_index += 1
                else:
                    snare_beat[snare_index] = 0
                    snare_index += 1
            elif beat[i] == '1':
                if bass_index != 16:
                    bass_beat[bass_index] = 1
                    bass_index += 1
                else:
                    snare_beat[snare_index] = 1
                    snare_index += 1
        bass_beats += [bass_beat]
        snare_beats += [snare_beat]
        beat = file.readline()
for i in range(0,16):
    pBassBeat[i] /= len(bass_beats)
    pSnareBeat[i] /= len(snare_beats)

# creating new beats semi-randomly with priors and conditional prob
newSnare = "["
newBass = "["
condition = ""
beat = [[],[]]
def makeString(pBass, pSnare):
    global newBass
    global newSnare
    global condition
    global beat
    yesBass = pBass > random.random()
    yesSnare = pSnare > random.random()

    if yesBass and yesSnare:
        # randomly selecting in event of tie
        if random.random() > 0.5:
            newBass += "1,"
            newSnare += "0,"
            condition = "bass"
            beat[0] += [1]
            beat[1] += [0]
        else:
            newBass += "0,"
            newSnare += "1,"
            condition = "snare"
            beat[1] += [1]
            beat[0] += [0]
    elif yesSnare:
        newBass += "0,"
        newSnare += "1,"
        condition = "snare"
        beat[1] += [1]
        beat[0] += [0]
    elif yesBass:
        newBass += "1,"
        newSnare += "0,"
        condition = "bass"
        beat[0] += [1]
        beat[1] += [0]
    else:
        newBass += "0,"

newSnare += "0,"
condition = "none"
beat[0] += [0]
beat[1] += [0]

#start beat generation from a random note
random_start = random.randint(0,15)
probs = [[0] * 16, [0]*16]
for sixteenth_note in range(0, 16):
    real_note = (random_start+sixteenth_note) % 15
    #first note is based just on prior - which is prob of beat calc above
    if sixteenth_note == 0:
        probs[0][real_note] = pBassBeat[real_note]
        probs[1][real_note] = pSnareBeat[real_note]
    else:
        priorBass = pBassBeat[real_note]
priorSnare = pSnareBeat[real_note]
total_count = 0
condit_bass_count = 0
condit_snare_count = 0
#calculating the conditional prob of a snare/bass based on last note value
if condition == "snare":
    for beat_index in range(0, len(snare_beats)):
        if snare_beats[beat_index][(real_note-1) % 15] == 1:
            total_count +=1
        if snare_beats[beat_index][real_note] == 1:
            condit_snare_count +=1
        if bass_beats[beat_index][real_note] == 1:
            condit_bass_count += 1
elif condition == "bass":
    for beat_index in range(0, len(bass_beats)):
        if bass_beats[beat_index][(real_note-1) % 15] == 1:
            total_count +=1
        if bass_beats[beat_index][real_note] == 1:
            condit_bass_count +=1
        if snare_beats[beat_index][real_note] == 1:
            condit_snare_count += 1
else:
    for beat_index in range(0, len(snare_beats)):
        if snare_beats[beat_index][(real_note-1) % 15] == 0 and bass_beats[beat_index][real_note] == 1:
            total_count +=1
        if snare_beats[beat_index][real_note] == 1:
            condit_snare_count +=1
        if bass_beats[beat_index][real_note] == 1:
            condit_bass_count += 1
if total_count == 0:
    probs[0][real_note] = priorBass
    probs[1][real_note] = priorSnare
else:
    bayes_psnare = condit_snare_count/total_count
    bayes_pbass = condit_bass_count/total_count
    probs[0][real_note] = (priorBass+bayes_pbass)/2
    probs[1][real_note] = (priorSnare+bayes_psnare)/2

for i in range(0,16):
    makeString(probs[0][i],probs[1][i])

newBass = newBass[0:(len(newBass)-1)] + "]"
newSnare = newSnare[0:(len(newSnare)-1)] + "]"
ewBeat = "[" + newBass + "," + newSnare + "]"
print("bass beat created: " + newBass)
print("snare beat created: " + newSnare)

unique = True
for i in range(0, len(snare_beats)):
    if snare_beats[i] == beat[1] and bass_beats[i] == beat[0]:
        unique = False
if unique:
    print("This beat is NOT part of the training data")
else:
    print("This beat IS part of the training data")

f = open("beat.txt", "w")
f.write(newBeat)
f.close()
g_f.set(200, 1); // set default drum attack

fun void setFilter(float f, float Q)
{
    s_f.set(f, Q);
}

fun void setDecay(float decay)
{
    g_fb.gain(1.0 - 1.0 / decay); // decay unit: samples!
}

fun void setAttack(float attack)
{
    g_f.freq(attack); // attack unit: Hz!
}

fun void hit(float velocity)
{
    velocity => i.next;
}

class BassKjzBD101
{
    Impulse i; // the attack
    i => Gain g1 => Gain g1_fb => g1 => LPF g1_f => Gain BDFreq; // BD pitch envelope
    i => Gain g2 => Gain g2_fb => g2 => LPF g2_f; // BD amp envelope

    // drum sound oscillator to amp envelope to overdrive to LPF to output
    BDFreq => SinOsc s => Gain ampenv => SinOsc s_ws => LPF s_f => Gain output;
    g2_f => ampenv; // amp envelope of the drum sound
    3 => ampenv.op; // set ampenv a multiplier
    1 => s_ws.sync; // prepare the SinOsc to be used as a waveshaper for overdrive

    // set default
    80.0 => BDFreq.gain; // BD initial pitch: 80 Hz
    1.0 - 1.0 / 2000 => g1_fb.gain; // BD pitch decay
    g1_f.set(100, 1); // set BD pitch attack
    1.0 - 1.0 / 4000 => g2_fb.gain; // BD amp decay
    g2_f.set(50, 1); // set BD amp attack
    .75 => ampenv.gain; // overdrive gain
    s_f.set(600, 1); // set BD lowpass filter

    fun void hit(float v)
    {
        v => i.next;
    }
fun void setFreq(float f)  
{
    f => BDFreq.gain;
}

fun void setPitchDecay(float f)  
{
    f => g1_fb.gain;
}

fun void setPitchAttack(float f)  
{
    f => g1_f.freq;
}

fun void setDecay(float f)  
{
    f => g2_fb.gain;
}

fun void setAttack(float f)  
{
    f => g2_f.freq;
}

fun void setDriveGain(float g)  
{
    g => ampenv.gain;
}

fun void setFilter(float f)  
{
    f => s_f.freq;
}

Shakers hhat => JCRev shk => dac;
Shakers snare => JCRev snr => dac;
Brass bass => JCRev bss => dac;
Moog melody => JCRev meldy => dac;
Moog melody2 => JCRev meldy2 => dac;
SnarekjzBD101 snare2;
snare2.output => dac;
BassKjzBD101 bass2;
bass2.output => dac;
//mixes for instruments
2 => shk.gain;
0.01=> shk.mix;
6 => snr.gain;
0.01 => snr.mix;
30 => bss.gain;
0.04 => bss.mix;
4 => meldy.gain;
0.40 => meldy.mix;
10 => meldy2.gain;
0.40 => meldy2.mix;
1.4 => float bps;
1000 / bps / 4 => float sixteenth_note;

"beat.txt" => string filename;
fun void findBeats(int bassBeat[], int snareBeat[])
{
    FileIO file;
    file.open(filename, FileIO.READ);
    if (!file.good()) {
        << "error opening " + filename >>;
        me.exit();
    }
    file.readLine() => string beat;
    2 => int startBIndex;
    36 => int startSIndex;
    for (0 => int i; i < 16; i++) {
        if (beat.charAt(startBIndex + 2*i) == 49) {
            1 => bassBeat[i];
        }
        if (beat.charAt(startSIndex + 2*i) == 49) {
            1 => snareBeat[i];
        }
    }
    //hi hat
    11 => hhat.preset;
    0.3 => hhat.energy;
    0.3 => hhat.decay;
    1 => hhat.objects;
    1.2 => hhat.noteOff;
    [1,0,1,0,1,0,1,0,1,0,0,1,0,1,0] @ => int hhatTimings[];
/snare
12 => snare.preset;
1 => snare.energy;
0.3 => snare.decay;
100 => snare.objects;
1.2 => snare.noteOff;
int snareTimings[16];

//bassdrum
87 => bass.freq;
0 => bass.volume;
0.8 => bass.lip;
0.2 => bass.slide;
0.1 => bass.rate;
1.2 => bass.noteOff;
4 => int bduration;
int bassTimings[16];

//melody
87 => melody.freq;
.99999 => melody.filterQ;
.65 => melody.filterSweepRate;
.9 => melody.afterTouch;
1.2 => melody.noteOff;
6 => int mduration;

//melody2
200 => melody2.freq;
.99999 => melody2.filterQ;
.65 => melody2.filterSweepRate;
.9 => melody2.afterTouch;
1.2 => melody2.noteOff;

//snare2
snare2.setFilter(3500,2);
snare2.setDecay(2800);

//bass2
bass2.setFreq(87);

findBeats(bassTimings, snareTimings);
// calculated estimated value of .0055 percent dif
0.0055 => float mean_pdif;
// calculated sample variance of .00178 -> std dev of .0421
0.04 => float dev_pdif;
// gaussian estimation - every index of array is aprox num of std devs away from mean for (index)*5% cdf p
[-2.5, -1.645, -1.28, -1.035, -0.84, -0.675, -0.525, -0.385, -0.25, -0.125,
0, .125, .25, 0.385, .525, .675, .84, 1.035, 1.28, 1.645, 2.5] => float gaussEstimate[];

//initial wait till begin
2*sixteenth_note => float wait;

// Calculates Time Difference in ms by using the gaussian p-value array and
// picking a random integer value from 0 to 20.
fun float findTimeDif()
{
    Std.rand2(0, 20) => int normSimulation;
gaussEstimate[normSimulation] => float offset;
    mean_pdif + (offset * dev_pdif) => float percent_dif;
    <<< "Last note's deviation from real time in ms: " + percent_dif*sixteenth_note >>>;
    return percent_dif*sixteenth_note;
}

/* playhiHat, snare, and bass use the helper function findTimeDif along with the timing arrays
from the beat.txt python script creation to
play their instruments at specific notes with time offsets */
fun void playHihat(int timings[], Shakers hihat)
{
    findTimeDif() => float next_offset;
    0 => int time;
    wait + next_offset => float next_wait;
    next_wait::ms => now;

    while (true)
    {
        if (timings[time]) {
            1.2 => hihat.noteOn;
        }

        if (++time == 16) {
            0 => time;
        }

        if (timings[time]) {
            next_offset => float current_offset;
            findTimeDif() => next_offset;
            sixteenth_note - current_offset + next_offset => next_wait;
        } else {
            
        }
    }
fun void playSnare(int timings[], Shakers snare, SnarekjzBD101 snare2)
{
    findTimeDif() => float next_offset;
    0 => int time;
    wait + next_offset => float next_wait;
    next_wait::ms => now;

    while (true)
    {
        if (timings[time]) {
            1.2 => snare.noteOn;
            snare2.hit(0.6);
        }
        if (++time == 16) {
            0 => time;
        }
        if (timings[time]) {
            next_offset => float current_offset;
            findTimeDif() => next_offset;
            sixteenth_note - current_offset + next_offset => next_wait;
        } else {
            sixteenth_note - next_offset => next_wait;
            0 => next_offset;
        }
        next_wait::ms => now;
        1.2 => snare.noteOff;
    }
}

fun void playBass(int timings[], Brass bass, BassKjzBD101 bass2)
{
    findTimeDif() => float next_offset;
    0 => int time;
    wait + next_offset => float next_wait;
    next_wait::ms => now;

    while (true)
{  
  if (timings[time]) {
    1.2 => bass.noteOn;
    bass2.hit(.8);
    bduration::ms => now;
    1.2 => bass.noteOff;
    sixteenth_note - next_offset - bduration => next_wait;
  } else {
    sixteenth_note => next_wait;
  }
  if (++time == 16) {
    0 => time;
  }
  if (timings[time]) {
    findTimeDif() => next_offset;
    next_wait + next_offset => next_wait;
  } else {
    0 => next_offset;
  }
  next_wait::ms => now;
}

// Similar to play hihat, etc except specific frequency hard
// coding for a predetermined melody
fun void playMelody(Moog melody, Moog melody2) {
  [174.61,0,174.61,174.61, 174.61,0,174.61,174.61, 174.61,0,174.61,207.65, 0,261.63,311.13,0, 277.2,0,277.2,277.2, 277.2,0,277.2,277.2, 261.63,0,261.63,233.08, 0,207.65,196,0] =>
    float notes[];
  findTimeDif() => float next_offset;
  0 => int time;
  wait + next_offset => float next_wait;
  next_wait::ms => now;
  0 => int melody_times;
  while (melody_times < 5)
  {
    if (notes[time] != 0) {
      notes[time] => melody.freq;
      if (melody_times > 1) {
        notes[time] * 2 => melody2.freq;
        1 => melody2.noteOn;
      }
    }  
    1.2 => melody.noteOn;
    mduration::ms => now;
    1.2 => melody.noteOff; 
  }
if (melody_times > 1) {
    1 => melody2.noteOff;
}
sixteenth_note - next_offset - mduration => next_wait;
} else {
    sixteenth_note => next_wait;
}
if (++time == 32) {
    0 => time;
    melody_times++;
}
if (notes[time] != 0) {
    findTimeDif() => next_offset;
    next_wait + next_offset => next_wait;
} else {
    0 => next_offset;
}
next_wait::ms => now;
}

// Timings for the song
(sixteenth_note * 16 * 2) => float pause;
(pause)::ms => now;
spork ~ playSnare(snareTimings, snare, snare2);
spork ~ playHihat(hhatTimings, hhat);
spork ~ playBass(bassTimings, bass, bass2);
pause::ms => now;
playMelody(melody, melody2);
pause*2::ms => now;

SampleBeats.txt

[[1,0,1,0, 0,0,0,0, 1,0,0,1, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]
[[1,0,0,1, 0,0,0,0, 1,0,0,1, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]
[[1,0,0,1, 0,0,0,0, 1,0,1,0, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]
[[1,0,1,0, 0,0,0,0, 1,0,0,1, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]
[[1,0,0,1, 0,0,0,0, 0,0,1,0, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,1,0, 1,0,0,0]]
[[1,0,0,1, 0,0,0,0, 0,0,1,0, 0,0,0,0],[0,0,0,0, 1,0,0,0, 1,0,0,0, 0,0,1,1]]
[[1,0,0,1, 0,0,0,0, 1,0,1,0, 0,0,1,0],[0,0,0,0, 1,0,0,0, 0,1,0,0, 1,0,0,0]]
[[1,0,0,1, 0,0,0,0, 1,0,1,0, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]
[[1,0,0,1, 0,0,0,0, 1,0,0,1, 0,0,0,0],[0,0,0,0, 1,0,1,0, 0,0,0,0, 1,0,0,0]]
[[1,0,0,1, 0,0,0,0, 1,0,1,0, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]

[[1,0,0,1, 0,0,1,0, 1,0,1,0, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]
[[1,0,0,1, 0,0,1,0, 1,0,0,1, 0,0,0,0],[0,0,0,0, 1,0,0,0, 1,0,1,0, 0,0,0,0]]
[[1,0,0,1, 0,0,1,0, 1,0,1,0, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]
[[1,0,0,1, 0,0,1,0, 1,0,1,0, 0,0,0,0],[0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0]]

SampleBeats.txt