ChucK Assignment 4

19 April 2012

In this lesson you will explore the Synthesis Toolkit (STK) instruments in ChucK, and learn to use and control randomness.

- **You need to have your ChucK Manual open for this lesson.**

1 STK Instruments

- In the early 1990s Perry Cook began developing the Synthesis Tool Kit (STK) at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University. It's a collection of software instruments that model physical instruments, or aid in creating synthetic sounds. ChucK includes the STK as UGens. They are listed in the ChucK manual on pages 137-170.
- You may notice that we have already used some of them: ADSR, Echo, PitShift. Today we're going to focus on the physical modeling instruments. These are synthetic instruments that attempt to imitate real instruments, not just in sound but in actual sound production method.

1. Look at the listing for StkInstrument on page 137. All STK instruments share some common parameters:
    - **noteOn** - trigger note on
    - **noteOff** - trigger note off
    - **freq** - set frequency in Hz
    - (We can ignore controlChange, since ChucK give us better controls, as you'll see below.)
2. Try this sample STK code:
```
ModalBar myBar => dac;
440 => myBar.freq;
1 => myBar.noteOn;
1::second => now;
1 => myBar.noteOff;
1::second => now;
```
3. The only significant difference between the above code and the oscillators you were working with before is the .noteOn and .noteOff methods. These are similar to the **ADSR** envelopes we used last week. All STK instruments have their envelopes built-in.
4. In the above example I used ModalBar, one of my favorite STK instruments. Turn to page 146 to read a little more about it. You'll see that there are numerous parameters specific to ModalBar: stickHardness, strikePosition, etc. The **preset** parameter lets you choose from the 9 different Modal Presets listed on page 146.

```
0.15 => myBar.stickHardness;
3 => myBar.preset;
60 => Std.mtof => myBar.freq; //middle C
1 => myBar.noteOn;
1::second => now;
1 => myBar.noteOff;
1::second => now;
```

5. Experiment with several different STK instruments. Don't be disappointed if they don't sound good right away–STK instruments are not meant to be perfect right out of the box! See if you can create interesting new sounds by pushing an instrument to extremes–very low notes on the Flute, for instance.

2 Randomness

6. Computers allow us to control every detail of a musical event, but sometimes that's an overwhelming task. Iannis Xenakis used controlled randomness to determine the low-level details of some of his pieces so he could focus his efforts on the big picture. ChucK has several methods for generating randomness.

    - **Std.rand2(0,10)** returns a random integer between 0 and 10.
    - **Std.rand2f(0,10)** returns a random float between 0.0 and 10.0.
    - **Std.randf()** returns a random float between -1.0 and 1.0.
    - The special keyword **maybe** is sometimes 1, sometimes 0. This is most useful in **if** statements.

7. Randomness is most commonly used to control parameters:

```
TubeBell myBell => dac;

repeat(10)
{
 Std.rand2(72,96) => Std.mtof => myBell.freq;
 1 => myBell.noteOn;
 1500::ms => now;
}
```

8. Randomness can also be used to make decisions. Let's replace the **1500::ms =¿ now;** with a random choice between two times:

```
TubeBell myBell => dac;
repeat(10)
{
 Std.rand2(72,96) => Std.mtof => myBell.freq;
 1 => myBell.noteOn;
 if (maybe) 500::ms => now;
 else 1500::ms => now;
}
```

9. Computer randomness isn't really random at all–it's a fancy algorithm for creating *pseudo-random* numbers. Don't worry–as far as we're concerned, they are good enough! And it gives us one nice advantage–we can choose our own "seed" value, and then every time we use that seed, we get the same repeatable series of random numbers. Try running this a few times:

```
Std.srand(830); // my random seed
repeat(10)
{
 <<< Std.rand2(1, 100) >>>;
}
```

Do you get the same series of "random" numbers each time?

10. The random number generators Std.rand2(), rand2f(), and randf() are designed to create a **flat distribution.** That is, every number is equally likely to occur. But we can gain some control over the randomness with a **weighted distribution.**

11. Let's say we want random numbers over a range of 0-100, but with *low numbers more likely to be chosen than high.* This is a **low-weighted** distribution. One good method is to pick **two** random numbers, then compare them and select the lower of the two:

    ```
    repeat(20)
    {
     Std.rand2(0,100) => int a;
     Std.rand2(0,100) => int b;
     if (a<b) <<< a >>>;
     else <<< b >>>;
    }
    ```

    Run this a few times. Notice how low numbers are much more common? How would you change this to a **high-weighted** distribution?

12. A **triangular** distribution is weighted toward the middle; that is, over a range of 0-100, numbers close to 50 are most common. To create this distribution, you again choose two random numbers, but now you average them together!

    ```
    repeat(20)
    {
     Std.rand2(0,100) => int a;
     Std.rand2(0,100) => int b;
     <<< (a+b)/2 >>>;
    }
    ```

Assignment 4: STK and Randomness

Create a short piece that uses one or more STK instruments and randomness. Save your ChucK code as a .ck or .txt file and upload it to the assignment page on Blackboard by this Tuesday, April 24.