Manipulación de sonido y archivos de sonido

Este capítulo cubre:

- Almacenamiento de sonido en computadores
- Carga y reproducción de archivos de sonido en ChucK usando SndBuf
- Trabajo con archivos stereo
- Modulo, un nuevo y útil operador aritmético
- Construcción de tu propia máquina de ritmos usando samples

Hasta el momento solo has usado osciladores y ruido para hacer tus sonidos y composiciones. Pero hay mucho más en la composición y producción de música que estos sonidos. Existen un sinnúmero de tupos de sonidos en el mundo y en la música. En este capítulo, te mostraremos cómo usar archivos de sonido en ChucK. Los archivos de sonido, como .wav y .aif, u otros archivos que tengas en tu computador o has visto en internet, son a menudo llamados samples, por razones que verás pronto. Los samples son una manera fácil y efectiva de construir elementos sónicos y musicales con mucha variedad y realismo. Usando samples, puedes acceder una gran colección de sonidos existentes que otra gente ha grabado y sintetizado a lo largo de los años.

En los siguientes capítulos aprenderas a sintetizar tus propios sonidos desde cero. Pero en este capítulo, usarás samples para llegar al punto que puedes crear tu propia máquina de ritmos con muchas nuevas habilidades de programación. Primero, le daremos un vistazo rápido a cómo el sonido es transformado de formas de onda en el aire a números digitales para

almacenamiento y transmisión por computador, llamado sampling. Luego profundizaremos en cómo usar samples en ChucK. También revisaremos nuevas maneras de usar arreglos e introduciremos un nuevo operador matemático llamado modulo.

4.1 Sampling: convertir sonido en números

Primero hablemos un poco de sonido. Definimos sonido como el viaje de compresiones fluctuando a través del aire, desde una fuente y hasta nuestros oídos. En los sistemas de computadores, el sonido es almacenado y reproducido en varios formatos digitales como números. Aunque no necesitas saber los detalles de todo esto para usar archivos de sonido en ChucK, es importante que sepas algunas cosas cuando cargas y reproduces archivos de sonido y cuando empieces a sintetizar sonidos en el futuro.

Para obtener sonido en un formato que tu computador y ChucK pueden entender, el sonido debe ser transformado en un flujo de números, que recibe el nombre de señal digital. El proceso de convertir una forma de onda sonora en una señal digital es llamado conversión análogo a digital. El dispositivo que hace esto se llamada conversor análogo-digital (ADC, por analog-to-digital converter), que es esencialmente lo opuesto al conversor digital-análogo (DAC) que has estado usando todo este tiempo para lograr que las señales digitales de ChucK sean emitidas por tus parlantes o audífonos. El ADC funciona por medio de la inspección electrónica de la forma de onda entrante (esto es llamado sampling o muestreo) a intervalos regulares, llamado el periodo de sampling o

muestreo (que equivale a 1/ tasa de sampleo o muestreo) y el redondeo al entero más cercano disponible para representar y almacenar esa amplitud. La figura 4.1 muestra este proceso, haciendo zoom en en periodo de "eee" de nuestro sonido "see" anterior. Hacemos aún más zoom al inicio de la forma de onda y mostramos unos pocos números enteros individuales que representan los primeros valores de samples.

En el capítulo anterior, cuando hablamos de arreglos, dijimos que los arreglos pueden almacenar lo que sea, y todo en tu computador está almacenado en un arreglo. El proceso mostrado en la figura 4.1 está haciendo exactamente eso, transformando los valores de la forma de onda en una secuencia de números y almacenándolos en un arreglo. Aquí los índices (0, 1, 2, ...) representan el tiempo en aumento, una cuenta por cada periodo de sampleo (1 / tasa de sampleo). Los niveles correspondientes a cada punto en el tiempo (sample) son enteros en muchos de los formatos de sonido como .wav y .aiff, pero ChucK los convierte en números de punto flotante en el rango +1.0 a -1.0.

Todas las formas de onda que hasta visto, incluyendo la de la figura 4.1, requieren tanto números positivos como negativos porque la línea de la mitad es el cero. Esto es generalmente cierto para audio, que es el viaje a través del aire de compresiones (presiones de aire mayores a la normal) y rarefacciones (menores a la normal), así que necesitas tanto números positivos para presiones de aire mayores que la presión ambiental del aire y negativos para presiones más bajas.

Entonces, cada número que representa un valor en el tiempo de una forma de onda recibe el nombre de sample y - quizás confusamente - una vez que todo el sonido es transformado en un torrente de números, esta colección de valores, indexados en el tiempo, también recibe el nombre de

sample. Cuando esa colección de valores es escrito para su almacenamiento en un disco, o como enlace en una página web, o quemado a un CD/DVD, es llamado un archivo de sonido. Otro término que deberías concoer es cuando un archivo de sonido es cargado a tu computador o sintetizador, donde a veces recibe el nombre de wavetable (tabla de onda.

Para reproducir audio, necesitas un DAC; que convierte los números de vuelta a samples de forma de onda continua, usualmente voltajes, en tus parlates o audífonos. Ya has usado bastante el objeto dac en ChucK para conectar tus osciladores y generadores de ruido a tus parlantes. ChucK, comibnado con el hardware de audio de tu computador, se encarga de convertir los archivos de sonido de vuelta a los voltajes necesarios para alimetnar tus audífonos o parlantes.

Sampling, tamaño de palabra, y tasa de sampleo

Los ingenieros necesitan considerar muchas aritas cuando crean y trabajan con archivo de uadio y especialmente cuando diseñan hardware para ADCs y DACs. Estos incluyen cuánto almacenamiento de computador debe ser reservado para cada valor de sample individual (llamado tamaño de palabra, típicamente 16 bits o 24 bits por sample) para música y 8 bits para sonidos de habla. Además, deben elegir cómo es sampleada la forma de onda (típicamente 44100 samples por segundo). Esto tiene implicaciones en la calidad y espacio en memoria/disco. Por ejemplo, tasas de sampleo más altas y más bits por sampe son generalmente mejores, hasta cierto punto. La calidad de sonido de un CD, a 2 bytes por sample, 2 canales y 44100 valores por segundo, consume cerca de 10 megabytes por minuto.

Para más información sobre sampling, formatos de archivos de sonido, entre otros, ve la referencia al final de este capítulo.

Ahora sabes un poco más sobre cómo el sonido llega a tu computador. Profundicemos más ahora y reproduzcamos archivos de sonido usando la unidad generadora incluida en ChucK llamada SndBuf.

4.2 SndBuf: carga y reproducción de archivos de sonido en ChucK

SndBuf, abreviación de sound buffer (buffer de sonido), es el UGen incluido en Chuck que permite cargar archivos de audio. Para hacerlo, debes empezar con un archivo de sonido o quizás una carpeta llena de archivos de sonido. Hemos provisto una serie de archivos de sonido que puedes usar para empezar a componer y que estarás usando a lo largo de este libro, para que todos los ejemplos funcionen sin que tengas que modificar algo. También es posible que agregues tus propios archivos de sonido y que hagas funcionar todo como tú desees. Primero configurarás una estructura de archivos en tu copmutador para que ChucK pueda encontrar los archivos de sonido. Luego aprenderás muchas maneras de cargar, reproducir y manipular archivos de sonido usando SndBuf.

4.2.1 Organización de tus archivos de sonido

Cuando empieces a usar archivos de sonido en ChucK, primero tenemos que ayudarle a configurar la estructura de archivos de computador para que puedas seguir los ejemplos de este libro, así como adoptar buenos hábitos de organización de tu código, sonidos y canciones.

Como se muestra en la figura 4.2, recomendamos primero crear un directorio (carpeta) llamado chuck en tu computador donde grabar todos tus archivos. En este directorio también tienes que crear otro directorio llamado audio para almacenar tus samples. En la versión de ChucK instalada en tu computador, encontrarás un directorio llamado audio. Copia y pega este directorio a tu nuevo directorio chuck para que puedsas usar esos archivos de sonido mientras lees este capítulo y el resto del libro.

Ahora que tienes configurada una estructura de archivos en tu computador que hace fácil usar archivos de sonido, puedes usar ChucK para probarla, usando samples pregrabados del toque de un tambor ubicado en el directorio audio. El listado 4.1 muestra cómo lograr justo eso, usando SndBuf, que es una nuevo UGen generador de onisod que te permite cargar y reproducir samples. Aunque lo conectes al dac tal como lo habías hecho con otros UGen hasta el moento, cuando se comparar con nuestro oscilador (SinOsc, TriOsc, entre otros) y generadores de ruido, existen un númbero de cosas sobre SndBuf que necesitas conocer para usarlo.

Primero, necesitas decirlea a SndBuf que cargue un archivo y de dónde, en caso contrario no emitirá ningún sonidl. Lo logras por medio de la construcción de una dirección de archivo completa y un nombre de archivo. Puedes imaginártelo como una dirección en el sistema de archivos de tu computador, tal como cuando tienes que usar el menú de navegación para cargar a tus programas documentos, hojas de cálculo, o cualquier otro tipo de archivos de tu computador. Con ChucK, mientaras tengas todo configurado de forma correcta (como se muestra en la figura 4.2), puedes cargar archivos desde el interior de programas ChucK usando solo código ChucK y sin la necesidad de menús.

EScribe el código ChucK del listado 4.1 y grábalo como UsingSndBuf.ck (o

cualquier otro nombre que desees; solo recuerda el nombre que usaste) en el directorio chuck que creaste previamente. El códgigo en la línea (1) le pide a ChucK que arroje la dirección completa de archivo donde se encuentra este archivo de ChucK. .me es una palabra clave. y me.dir() retorna el directorio actual donde este archivo ChucK reside. Esta es la razón por la que es importante grabar tu archivo en algún lugar del directorio chuck. Graba la dirección del archivo en un string llamado path para ser usado después. En la siguiente línea crea una nueva variable tipo string e inicialízala con el nombre del sonido que vas a reproducir; incluyendo el nombre del directorio audio donde se ubica el archivo.

Luego crea otra variable tipo string llamada filename (2) para almacenar el nombre del archivo de sonido que vas a reproducir. Este nombre debe incluir el directior de audio que contiene tus archivos de sonido. Observa que aquí usamos el símbolo + de una manera nueva, para unir las dos variables tipo string llamadas path y filename para crear un nuevo string, que luego almacenamos en la variable filename. Esta es otra característica interesante de ChucK, donde el signo más puede sumar números o unir variables string automáticamente, basado en el tipo de datos en cuestión.

Listado 4.1 Carga y reproducción de archivos de sonido usando SndBuf

```
//Uso de SndBuf para reproducir un archivo de sonido
//por programador de ChucK, diciembre 2050
SndBuf mySound => dac;

//obtener la dirección del directorio
//(1) obtiene el directorio actual de trabajo
me.dir() => string path;
//archivo de sonido que queremos reproducir
"/audio/snare_01.wav" => string filename;
```

```
// signo + concatena strings
//(2) construye una dirección completa del archivo de sonido
path + filename => filename;

//indicarle a SndBuf que lea este archivo
//(3) hacer ChucKing de un string al método .read de SndBuf ca
filename => mySound.read;

//definir la ganancia
0.5 => mySound.gain;

//reproducir el sonido desde el princpio
//hacer ChucKing de un número al método .pos (de posición) de
0 => mySound.pos;

//hacer que el tiempo transcurra para poder escuchar el sonido
second => now;
```

NOTA No puedes usar el signo + para sumar un string y un float, incluso si el string puede parecerse a un número. Es así que "123.7" + 6.3 no es legal, pero dependiendo de si quieres que el resultado final sea string o float, puedes usar Std.atof("123.7") + 6.3 para obtener un float. Recuerda que Std.atof() convierte un string ASCII en un número de punto flotante. Puedes también usar "123.7" + Std.ftoa(6.3) (float a ASCII) para obtener un string, aunque no tenga mucho sentido, porque tendría dos puntos decimales.

Una vez que has ensamblado la dirección completa del archivo, puedes hacer ChucKing a tu SndBuf usando el método .read (3), y automáticamente cargará el archivo desde el disco y lo almacenará en el arreglo interno de tu SndBuf. Si esto no funciona, por ejemplo si no se

pudo encontrar el archivo porque la dirección era incorrecta o hiciste un error al escribir algo en el nombre del archivo, entonces ChucK imprimirá un eror en la ventana Console.

> Formatos de archivos de sonido y extensiones de archivo
>
> Entre los muchos tipos de formatos de archivo de audio, los más comunes son .wav (por wave o waveform, onda o forma de onda) y .aif (de audio interchange file o format, archivo o formato de intercambio de audio). SndBuf es capaz de entender estos y otros formatos. Basta con indicarle a SndBuf .read, y ChucK se encargará del resto, o arrojará un error si no es capaz de encontrar el archivo o leer el formato particular. ChucK todavía no lee archivos .mp3 comprimidos, así que si tratas de cargar uno, obtendrás un mensaje de "Format not recognized" (Formato no reconocido) en la ventana Console Monitor.

Puedes definir el volumen de tu SndBuf usando el método .gain. Como recuerdas, cada UGen en Chuck obedece al método .gain, así que siempre puedes usarlo para cambiar el volumen. Finalmente, para hacer que tu SndBuf se reproduzca, usa el nuevo método .pos (de position, posición) para apuntar el puntero interno de SndBuf al primer sample, indexado como cero, tal como cualquier otro arreglo. Después de eso, basta con hacer ChucKing de una duración (en este caso 1 segundo) a now, para permitir que el tiempo pase y el sonido sea generado. SndBuf se encarga del resto, entregándole samples individuales al dac para su reproducción a través de tus parlantes o audífonos.

Asumiendo que has escrito todo de forma correcta y que todos los archivos están en los lugares correctos, si haces click en el botón Add

Shred en el miniAudicle, ¡deberías escuchar un gople en el tambor llamado caja (snare drum)! Haz click en el botón unas cuantas veces adicionales. Podrías incluso tocar esto en vivo como baterista en una banda. Hacer click es un poco difícil - igual que tocar un tambor exactamente en el momento indicado - pero es posible.

4.2.2 Looping (repetición automática) de tus samples

Como has visto, el objeto SndBuf tiene un número de métodos distntos para controlar sonido. Recuerda que nuestros osciladores tenían métodos .gain y .freq, y como has visto, SndVuf tiene un objeto .gain, pero no posee un método .freq. La razón de esto es que como SndBuf contiene una grabación, no tienes idea de qué frecuencia, o si incluso tiene una, contiene esta grabación. Pronto verás cómo hacer que la altura del sonido de un SndBuf aumente o disminuya.

También has visto que SndBuf tiene un método .pos, que es usado para definir la ubicación donde empiezas la reproducción de un archivo, llamado la cabeza de reproducción. Una manera de pensarlo es imaginarte tu archivo de sonido como una grabación en vinilo. El método .pos sería donde ubicas la aguja en el disco. Observando los samples almacenadas en el SndBuf de la figura 4.3, nota que cada sample sucesivo yendo hacia la derecha se corresponde exactamente con los valores de sample de nuestra forma de onda "eee" de la figura 4.1. Otra manera de verlo es recordando que tu archivo de sonido está almacenado en un gran arreglo, y que el método .pos te indica en cuál índice del arreglo debes empezar, como se muestra en la figura 4.3.

En la figura 4.3, la posición cero (con valor cero de sample) es el principio del sample, el momento cero del archivo de sonido. La posición etiquetada como 1 (valor 40) contiene el sample un T (periodo de sampleo) posterior. Cada posición es el sample un T más tarde, y así hasta el final del archivo de sonido.

¿Y si quiesieras tocar tu sample SndBuf una y otra vez, quizás en un bucle? Puedes hacerlo si repetidamente retornas la cabeza de reproducción a cero usando el método .pos. Mientras estás en esto, vas a añadir otras características expresivas, como un objeto de paneo, que recuerdas del capítulo 2. Como ya lo has hecho antes, ubicarás el control de tu sonido dentro de un bucle while, como se muestra en el listado 4.2. Graba este archivo en tu directorio chuck una vez que lo hayas escrito. En el listado 4.2, usas números aleatorios para definir el volumen/ganancia de tu archivo de sonido (1), y defines una posición de paneo aleatoria (2). Como es costumbre ya, cerarás un nuevo volumen y una nueva posición espacial para cada toque (4).

Para aún mayor variedad musicak, usarás un método adicional incluido en el objeto SndBuf. Uno de los aspectos más expresivos de trabajar con archivos de audio es obtener el control de la tasa de reproducción del sample. Piensa de nuevo en el reproductor de discos: cuando un disco es reproducido más rápidamente de lo normal, el sonido sube en altura. Similarmente, cuando se reproduce a una velocidad menor, el sonido baja en altura. En una sola línea de código, puedes alterar y experimentar con este parámetro para obtener una gran variedad de expresión en tus composiciones. Lo haces usando el método .rate, para obtener un valor aleatorio entre .2 y 1.8. La tasa de reproducción original es 1.0 (3). Arriba de 1.0 será más rápido y por lo tanto más alto, y bajo 1.0 será más lento y por lo tanto más grave.

Tú defines el tiempo a esperar (5), antes de repetir el bucle. Aquí esperas 500 :: ms (1/2 segundo) entre cada nuevo golpe. Puedes aumentar este número y observar que tu reproducción en bucle se hace más lenta, hazlo más corto y escucha como acelera.

Listado 4.2 Uso de un bucle para repetir la reproducción de un archivo de sonido

```
//Reproduce un sonido repetidamente en un bucle
//por programador de ChucK, 12 de enero 2017
//Conecta un SndBuf a través de un objeto de paneo Pan2 al DAC
SndBuf mysound => Pan2 pan => dac;

//obtener la dirección del archivo y cargarlo en la misma línea
me.dir() + "audio/cowbell_01.wav" => mySound.read;

//reproduce nuestro sonido una y otra vez en un bucle infinito
while (true)
{
  //ganancia, tasa (altura), y paneo aleatorios cada vez
  //(1) Ganancia aleatoria para el archivo de sonido
  Math.random2f(0.1, 1.0) => mySound.gain;
  //(2) Posición de paneo aleatoria
  Math.random2f(-1.0, 1.0) => pan.pan;
  //(3) Tasa aleatoria (velocidad y altura)
  Math.random2(0.2, 1.8) => mySound.rate;

  //(re)comienza el sonido configurando la posición a 0
  //(4) Configura la posición a 0 para que empiece a sonar
  0 => mySound.pos;

  //avanza el tiempo para poder escucharlo
  //(5) Espera un poco mientras toca
  500.0 :: ms => now;
```

```
}
```

> Controlando y deteniendo tus bucles con Replace Shred y Clear VM
>
> Reuerda que para hacer un cambio, como el rango de números aleatorios o el tiempo que usas para ChucKing a now para controlar la sincronización de tus sonidos en bucle, necesitas hacer click en el botón Replace Shred para detener el bucle antiguo y reemplazarlo con el nuevo. Como esto es un bucle infinito (while (true)), cuando has finalizado y estás listo para continuar, tienes que detener lo que está siendo ejecutado haciendo click en Clear VM.

4.2.3 Reproducción de tus samples en reversa

Como esto tuvo un sonido tan interesante, no nos detengamos aquí en nuestra experimentación con samples. ¿Y si quisieras reproducir un sample en reversa? Esta es una técnica útil que puede añadir un motif expresivo adicional a tus composiciones, y eso es usado para efectos sonoros, efectos musicales especiaesl, diseño sonoro, entre otros.

El listado 4.3 muestra cómo lograr esto, usando un diferente sonido, un tipo de platillo llamado hi-hat. Después de declarar un SndBuf y conectarlo al dac, cargas tu archivo de sonido hi-hat usando me.dir() y otros métodos que ya hemos visto (1).

Para reproducir un archivo de sonido en reversa, necesitas hacer dos cosas. Lo primero es ubicar la posición de la cabeza de reproducción al final del archivo, un arreglo interno en SndBuf. ¿Pero cómo sabemos cuántos samples hay en el archivo de sonido? SndBuf tiene un método

llamado .samples() que arroja el número de samples en tu archivo de sonido, y lo almacenas en una variable entera llamada numSamples (2). Otra forma de pensar esto es que le estás preguntando a SndBuf, cuán grande es tu arreglo interno que almacena el archivo de sonido (similar al método .cap() de los arreglos normales)? Primero debes reproducir el sonido una vez hacia adelante y a velocidad normal; observa que aquí estás usando numSamples para avanzar el tiempo en exactamente el número correcto (3). Usando el método .samples(), puedes ahhora usar el número como un límite superior con el método .pos para definir dónde empieza la posición de la cabeza de reproducción (4), como se muestra en la figura 4.4.

Lo segundo que debes hacer para reproducir un archivo en reversa es definir .rate para que vaya en reversa en vez de hacia adelante. Como debes haberlo adivinado, se logra haciendo que el valor de la tasa sea negativo -1.0 (5) para ir en reversa a la velocidad correcta. También puedes usar -.2 para ir más lento y en reversa, -2.0 para ir en reversa al doble de velocidad, entre otros.

Listado 4.3 Reproduciendo un archivo de sonido en reversa

```
//Reproducción de sonidos en reversa
//por programador de ChucK, 4102 oiluj
//La fecha está al revés, tal como el sonido dentro de poco
SndBuf mySound => dac;

//(1) Construye la dirección del archivo y para cargar en SndB
me.dir() + "/audio/hihat_04.wav" => mySound.read;

//(2) Averigua cuán largo es el sonido (en número de samples)
mySound.samples() => int numSamples;
```

```
//reproduce una vez el sonido hacia adelante
0 => mySound.pos;
//(3) deja que el sonido sea reproducido
numSamples :: samp => now;

//y una vez hacia atrás
//(4) Hace que la posición del buffer sea el final del archivo
numSamples => mySound.pos;
//(5) Hace que la reproducción sea en reversa
-1.0 => mysound.rate;
// Reproduce el archivo completo, pero en reversa
numSamples :: samp => now;
```

4.2.4 Manejo de múltiples samples a la vez

La habilidad final a aprender en esta sección es cómo reproducir múltiples SndBuf al mismo timepo. Existen dos maneras para hacerlo, dpendiendo de cuán largos sean tus sonidos y cuáles sean tus necesidades de composición. Una forma es usar solo un SndBuf y volver a cargar los diferentes sonidos. Digamos que tienes tres diferentes grabaciones de cajas, y quieres intercambiarlas durante una composición. Puedes hacerlo con un arreglo de strings para almacenar las direcciones de los archivos y nombres (1), como se muestra en el listado 4.4. Ahora en cualquier momento puedes acceder a las direcciones de los archivos de sonido y cargarlos en memoria en cualquier punto durante el programa. Como un ejemplo simple aquí, en un bucle infinito, generas un número aleatorio (2) y lo usas para decidir cuál archivo será cargado y reproducido (3).

Observa aquí que lo que estás haciendo es levemento distinto para reproducir estos archivos, ya que no tienes que explícitamente definir el

puntero .pos a cero. Dentro de SndBuf, cuando se carga un archivo automáticamente se define el puntero .pos a cero y la tasa .rate a 1.0. Entonces, todos los archivos de sonido se reproducirán automáticametne una vez que son cargados, siempre y cuando el tiempo avance.

Listado 4.4 Reproduccion de distintos archivos de sonido usando un SndBuf

```
//Reproduce múltiples sonidos
//por programador de ChucK, julio 2023
SndBuf snare => dac;

//construye y puebla un arreglo de direcciones de archivos de s
//(1) Crea y puebla un arreglo de nombres de archivos de sonid
string snare_samples[3];
me.dir() + "/audio/snare_01.wav" => snare_samples[0];
me.dir() + "/audio/snare_02.wav" => snare_samples[1];
me.dir() + "/audio/snare_03.wav" => snare_samples[2];

//bulce infinito
while (true)
{
  //escoge un número aleatorio entre 0 y el número de archivos
  //(2) Escoge un número aleatorio entre 0 y 2
  Math.random2(0, snare_samples.cap() - 1) => int which;

  //carga ese archivo
  //(3) Carga el archivo aleatorio asociado
  snare_samples[which] => snare.read;

  //Deja que el tiempo transcurra para la reproducción
  0.5 :: second => now;
}
```

El método del listado 4.4 puede funcionar sin problemas, pero también puede causar clicks si otros procesos de tu computador se interponen en la carga de los archivos siendo cargados a tiempo y reproducidos o si los archivos son muy largos. Existe una mejor manera. ¿Reucerdas cuando en el capítulo 3 dijimos que podías hacer arreglos para almacenar cualquier tipo de datos? Aquí usaremos esto para declarar un arreglo de SndBufs (1) y precargarlos con diferentes archivos de sonido (2), como se muestra en el siguiente listado. De esta forma, todos los archivos de sonido son cargados antes de que entres al bucle principal, y puedes evitar clicks evitando cargar los archivos mientras algún sonido esté en reproducción

Listado 4.5 Reproducción de distintos archivos de sonido (método 2) usando múltiples SndBufs

```
//Reproducción de múltiples sonidos (método 2)
//por programador ChucK, agosto 2023
//(1) Declara un arreglo de thres SndBufs
SndBuf snare[3];

//por diversión, hagamos paneo a izquierda, centro y derecha
snare[0] => dac.left;
snare[1] => dac;
snare[2] => dac.right;

//precarga todos los archivos de sonido al principio
//(2) Carga los tres sonidos diferentes
me.dir() + "/audio/snare_01.wav" => snare[0].read;
me.dir() + "/audio/snare_02.wav" => snare[1].read;
me.dir() + "/audio/snare_03.wav" => snare[2].read;

//bucle infinito
while (true)
{
```

```
    //escoge un número aleatorio entre 0 y número de archivos -
    Math.random2(0, snare.cap() - 1) => int which;

    //Reproduce ese sonido
    0 => snare[which].pos;

    //Permite que suene
    0.5 :: second => now;
}
```

Puedes haber notado que los resultados sonoros del código del listado 4.5 son los mismos que los del listado 4.4 Esto es común en la programación, donde, tanto con los bucles for como while, viste que existen múltiples maneras de obtener los mismos resultados. Sin embargo, existen casos donde el método del listado 4.5 puede no ser el deseado. Un ejemplo puede ser si estás usando archivos de sonido muy grandes (como canciones completas, que no recomendamos por el momento) o si estás corriendo ChucK en una máquina con poca RAM. ChucK core en algunas máquinas muy pequeñas, como Macs y PCs muy antiguos, e incluso en Raspberry Pi, un computador Linux muy barato y pequeño. En casos como este, precargar todos los archivos que necesitas podría ser imposible. Podrías quedarte sin memoria, antes de que todo sea cargado. ¡Pero ahora conoces ambos métodos y puedes escoger cuál usar!

Hemos visto cómo usar Pan2 (listado 4.2) y cómo lograr paneo extremo - conexión directa a la izquierda, centro y derecha, como se muestra en el listado 4.5 - para obtener efectos stereo usando SndBuf. ¿Pero qué ocurre con archivos stereo? ¡No hay problema! Sigue leyendo.

4.3 Archivos de sonido stereo y

reproducción

El código que hemos visto hasta el momento trata los archivos de sample como si fueran de un solo canal, o mono (abreviación de monoaural, lo que significa "un oído"). No obstante, tienes dos oídos y usualmente escuchas sonidos y música usando dos parlantes o audífonos con dos casos. Recuerda que ya has estado usando una unidad generadora Pan2 para mover ruido entre tus parlantes o audífonos izquierdos y derechos. Ahora vas a querer ser capaz de usar archivos de sonido stereo, de dos canales. Los archivos de sonido stereo, en un sentido, vienen pre-espacializados, así que no necesitas usar Pan2 para ubicarlos en el campo espacial sonoro. Un archivo stereo bien producido tendrá un sentido de espacio más auténtico que lo que puede lograrse con un archivo mono procesado por Pan2. Eso sí, es más difícil cambiar la espacialización una vez que el archivo ha sido grabado, haciendo que los ajustes de paneo dinámicamente sean mucho más difíciles con código.

La carga de archivos de sonido stereo en ChucK es casi lo mismo que con archicos mono, pero en vez de usar SndBuf usas SndBuf2, como se muestra en el listado 4.6. La añadidura del 2 en el nombre indica que la unidad generadora es stereo, esto es, tiene dos canales de salida. Todo debería ser bastante standard hasta ahora, excepto por una nueva función/método que usas (1), para obtener una duración para avanzar el tiempo. EL método .length() de SndBuf y SndBuf2 retorna una duración que es exactamente igual al tiempo requerido para reproducir ese archivo. Entonces, puedes usarlo para preguntarle a SndBuf2 cuánto tiempo sonar y hacer ChucKing de esto inmediatamente a now.

Listado 4.6 Carga y reproducción de archivos de sonido stereo usando

SndBuf2

```
//Carga y reproducción de archivos de sonido stereo
//por programador ChucK, septiembre 2023
// Hace un SndBuf2 stereo y lo conecta al dac (¡se hace stereo
SndBuf2 myStereoSound => dac;

//se carga un archivo stereo usando .read
me.dir() + "/audio/stereo_fx_01.wav" => myStereoSound.read;

//y lo hace sonar por el tiempo apropiado
//(1) ¡Nueva función / método! .length retorna la duración exa
myStereoSound,length() => now;
```

Observa que ya no estás usando un objeto Pan2 para controlar la espacialización porque SndBuf2 produce una salida de dos canales. Al conectarlo directamente al dac, automáticamente ocurren las cosas correcas (izquierda a izquierda, derecha a derecha, tal como las dos salidas de Pan2).

Observa tamvién que si hubieras conectado tu SndBuf2 a una unidad Gain antes del dac, el resultado hubiera sido diferente, Cuando una salida de dos canales es conectada a una entrada de un canal, los dos canales son mezclados para ajustarse a la entrada. Esto elimina cualquier efecto de espacialización en la señal stereo original, por lo que mezclar la salida de SndBuf2 generalmente anula el sentido de usar SndBuf2. Si quieres lograr esto, carga un archivo stereo en un SndBuf regular, y los dos canales serán mezclados internamente dentro del objeto SndBuf y almacenados en un arreglo mono interno.

No obstante, aún puedes usar UGens Gain para controlar el campo sonoro

stereo en otras maneras, como se muestra en el listado 4.7. Aquí combinamos tu conocimiento de arreglos y unidades generadoras para crear un arreglo de UGens, uno por cada canal. Usando un arreglo de twos UGens Gain, creas un control de balance stereo, que es una manera de ajustar el paneo de una señal stereo. ¿Pero cómo funciona esto?

Después de hacer un SndBuf2 stereo y cargar un archivo stereo (1), haces un arreglo de dos UGens Gain, para ser usados para volumen izquierdo/derecho (2). Luego conectas el canal izquierdo de tu SndBuf2 al Gain cero (bal[0]) y luego al canal dac.left (3). Haces lo mismo con dac.right y bal[1].

Listado 4.7 Paneo stereo con archivos de sonido stereo usando SndBuf2

```
//Carga y paneo de archivos de sonido stereo
//por programador ChucK, octubre 2023

//declara y carga un archivo stereo
SndBuf2 myStereoSound;
//(1) Crea un archivo SndBuf2 streo y carga un archivo stereo
me.dir() + "/audio/stereo_fx_03.wav" => myStereoSound.read;

//usaremos estos para paneo en stereo
//(2) Hace un arreglo de Ugens Gain para control stereo de volu
Gain bal[2];

//conecta todo en stereo
//(3) Conecta el izquierod al izquierdo y el derecho en el dere
mySteroSound.chan[0] => bal[0] => dac.left;
mySteroSound.chan[1] => bal[1] => dac.right;

//configura nuestro archivo para que se repita para siempre
//(4) Repite automáticamente
```

```
1 => myStereoSound.loop;

while(true)
{
  //escoge una tasa aleatoria de reproducción y un paneo aleat
  //(5) Define una tasa aleatoria (altura y tiempo)
  Math.random2f(0.2, 1.8) => myStereoSound.rate;
  //(6) Define un balance aleatorio (paneo)
  Math.random2f(-1.0, 1.0) => float balance;

  //convierte el balance en ganancias izquierda/derecha entre (
  //(7) Implementa el control de balance stereo
  (balance + 1) / 2.0 => float rightGain;
  1.0 - rightGain => float leftGain;

  leftGain => bal[0].gain;
  rightGain => bal[1].gain;

  0.3 :: second => now;
}
```

A continuación haces algo nuevo (4) con tu SndBuf2, que también lo puedes hacer con tu SndBuf regular mono. Haciendo ChucKing de un 1 al método .loop, le dices a SndBuf que quieres que toque en bucle para siempre. Normalmente, cualquier SndBuf se reproduce solo una vez y se queda al final, esperando que hagas que .pos vuelva a cero. Pero al configurar .loop como 1, le estás diciendo al mecanismo interno de SndBuf que cuando el puntero de posición llega al final del arreglo y todos los samples han sido reproducidos, debería volver a 0 automáticamente y empezar a tocar de nuevo. Observa que también podrías usar esto para reproducción en reversa (tasas negativas), y que cuenado el puntero llega a 0, automáticamente vuelve al último sample y repite la reproducción en

reversa para siempre.

Ahora vayamos a la sección de paneo. Dentro del bucle infinito, defines una tasa de reproducción aleatoria (5), y creas y defines (aleatoriamente) una nueva variable llamada balance (6). Tal como con Pan2, cuando haces que la variable balance se incline hacia -1.0, el canal izquierdo se vuelve más prominente; si el balance se inclina hacia 1.0, hace que el canal derecho sea más prominente. Una vez que el balance es definido aleatoriamente entre -1 y 1, y haces los cálculos para convertir eso en ganancias, entre 0 y 1, para controlar los canales izquierdo y derecho (7).

PRUEBA ESTO En papel, define balance a -1.0; luego haz los cálculos para llegar a los valores de leftGain y rightGain. Haz lo mismo para balance definido como 0.0 y balance definido como 1.9. Observa cómo mover balance de -1.0 a 1.0 causa que el paneo se mueva de izquierda a derecha. Haz esto también en código ChucK para que puedas escuchar la diferencia a medida que cambias balance.

Como viste anteriormente cuando precargabas un arrego de UGens SndBuff, los arreglos de UGens son una técnica común para crear y manipular patches de síntesis multicanal en ChucK.

4.4 Ejemplo: construcción de una máquina de ritmos

Ahora que has visto formas de usar reproducción de samples en ChucK, combinemos todo esto para hacer una máquina de ritmos que haga bailar, con múltiples archivos de sonido. Empezemos con el bombo (kick drum, tambor en un set de batería tocado con un pedal) y una caja (snare drum),

conectados al dac (1), a través de un mezclador UGen Gain que llamaremos master, como se muestra en el listado 4.8. El objeto UGen Gain te permite hacer un control de ganancia en cualquier parte de tu cadena de audio, en este caso antes de la salida al dac, para que puedas controlar el volumen de tu composición completa por medio de modificar master.gain. Podrías recordar que todos los UGens permiten control de ganancia, pero también proveen un punto de conexión para mezclar múltiples fuentes y otras funcioens que aprenderás más adelante.

Una vez que has creado y conectado en tu código, solo necesitarás conectar nuevos SndBufs adicionales al UGen Gain master (2). Esto es porque ya has conectado el Gain master al dac. Después de conecetar tus Sndbufs, puedes cargarlos con los archivos de sonido adecuados (3). Luego entras al bucle infinito y tocas el bombo (0 => kick.pos) por sí mismo en el tiempo 1 (4), y luego tocas ambos tambores juntos en el tiempo 2 (5), por medio de hacer ChucKing de 0 a tanto kick.pos como snare.pos.

Listado 4.8 Construir una máquina de ritmos con SndBufs en ChucK

```
//Máquina de ritmos, versión 1.0
// por programador con ritmo, 31 de diciembre 1999

//SndBufs para bombo (kick, bass drum) y caja (snare)
//(1) SndBuf a mezclador Gain master y a dac
SndBuf kick => Gain master => dac;
//(2) Otro sndBuf al mezclador Gain master
SndBuf snare => master;

//carga algunos archivos
//(3) Carga tus archivos de sonido
me.dir() + "/audio/kick_01.wav" => kick.read;
me.dir() + "/audio/snare_01.wav" => snare.read;
```

```
while(true)
{
  //En el tiempo 1, toca solo el bombo
  //(4) En los tiempos impares, solo toca el bombo
  0 => kick.pos;
  0.6 :: second => now;

  //Toca ambos tambores en el tiempo 2
  //(5) En cada tiempo par, toca tanto el bombo como la caja
  0 => kick.pos;
  0 => snare.pos;
  0.6 :: second => now;
}
```

4.4.1 Añadir lógica para tener distintos tambores en distintos momentos

¡Ya estás haciendo música! Pero puedes hacer mucho mejor música usando un bucle for y algunas declaraciones lógicas. ¿Te acuerdas de ellas? Te dijimos que cobrarían mayor importancia posteriormente. Cambia el bucle while anterior por lo que se muestra en el siguiente listado. También agrega una nueva variable duration para tempo (4), y reemplaza el bucle while con un bucle for (5) y pruebas lógicas (6) (7).

Listado 4.9 Mejorando el bucle while de tu máquina de ritmos

```
//Máquina de ritmos, versión 2.0
//por programador con ritmo, 32 de diciembre, 1999

//SndBufs para bombo (kick, bass drum) y caja (snare)
//(1) Bombo al mezclador Gain master y al dac
```

```
SndBuf kick => Gain master => dac;
//(2) La caja también va al mezclador
SndBuf snare => master;

//carga algunos archivos
//(3) Carga tus archivos de sonido
me.dir() + "/audio/kick_01.wav" => kick.read;
me.dir() + "/audio/snare_01.wav" => snare.read;

//declara una nueva variable de tempo
//(4) La duración del tempo es la separación entre los tiempos
0.15 :: second => dur tempo;

while(true)
{
  //(5) Bucle sobre una "barra" de 16 tiempos
  for (0 => int beat; beat < 16; beat++)
  {
    //reproduce el bombo en los tiempos 0, 4, 8 y 12
    //(6) Toca solo el bombo son el algunos tiempos específicos
    if (beat == 0 || beat == 4 || beat == 8 || beat == 12)
    {
      0 => kick.pos;
    }
    //reproduce la caja en los tiempos 4, 10, 13 y 14
    //(7) Toca solo la caja en algunos tiempos específicos
    if (beat == 4 || beat == 10 || beat == 13 || beat == 14)
    {
      0 => snare.pos;
    }
    tempo => now;
  }
}
```

NOTA: Puedes cambiar los números de if (beat == N) para revisar cómo

cambia el patrón en general

En el listado 4.9 usas un bucle for (1) para tocar 16 tiempos cada vez. Músicos y compositores llaman a esto un compás. Usas operadores lógicos para y expresiones como if y || (or) para tocar los distintos tambores en diferentes tiempos dentro de tu compás; tocas el bombo en algunos tiempos (2) y la caja en otros (3). Prueba cambiando los números en las condiciones beat == #, haciendo click en el botón Replace Shred, y observa como el patrón cambia. Puedes volverte muy bueno en hacer click en el botón en el momento correcto, para que tu patrón de batería cambie, pero conservando el tiempo básico. En los próximos capítulos aprenderás técnicas para usar el fuerte sentido y control de tiempo de ChucK para lograr este tipo de temporización sobre la marcha de forma exacta.

> Bucles y otras estructuras de control pueden ser anidadas
>
> Nota que en el listado 4.9 se ha anidado un bucle for dentro un bucle while, y lo hemos hecho también en ejemplos anteriores en otro capítulos. Esto no presenta ningún problema, por supuesto, y es realmente poderoso. Puedes anidar bucles for dentro de bucles for, bucles while dentro de bucles while o for, ifs dentro de ifs, ifs dentro de elses, con cuánta profundidad quieras. Mientras mantengas tus llaves derechas y alineadas, para lo que miniAudicle te brinda ayuda, puedes saber con mayor facilidad en qué parte cada bloque empieza y termina.

4.4.2 Control la reproducción de los tambores usando arreglos lógicos

Usa tu conocimiento sobre arreglos para controlar cuando la batería toca,

como se muestra en el listado 4.10. Ahora añade otro sonido de batería (1), un sonido de platillo hi-hat. Aquí cambias el bucle for por uno while; nuevamente, existen múltiples formas de resolver problemas en ChucK. Defines dos arreglos (2) que contienen valores lógicos, o 1 o 0, para decirle a la batería si toca o no en ese tiempo. Entonces aquí el índice en el arreglo es el número de tiempo. Eso hace que tu lógica sea muy simple: en vez de todos esos || o condiciones, lees la variable lógica desde el arreglo en el índice adecuado de tiempo, y te indica si debes o no tocar ese sonido. Usas un arreglo para el bombo y uno para la caja, mientras que el sonido hi-hat es tocado en cada tiempo. No obstante, sería muy fácil introducir un tercer arreglo hatHits[] y someter ese platillo a su propia declaración lógica también. Observa que de nuevo usas la función .cap() para determinar el tamaño de uno de tus arreglos.

Listado 4.10 Uso de arreglos para mejorar aún más tu máquina de ritmos

```
//Máquina de ritmos, versión 3.0
//por programador con ritmo, 31 de diciembre, 1999

//SndBufs para bombo (kick, bass drum) y caja (snare) y hi-hat
//(1) Los SndBufs kick, snare y hihat van al mezclador y al da
SndBuf kick => Gain master => dac;
SndBuf snare => master;
SndBuf hihat => master;


//carga algunos archivos
me.dir() + "/audio/kick_01.wav" => kick.read;
me.dir() + "/audio/snare_01.wav" => snare.read;
me.dir() + "/audio/hihat_01.wav" => hihat.read;

0.3 => hihat.gain;
```

```
0.15 :: second -> dur tempo;

//partituras (arreglos) para decirle a la batería cuando tocar
//(2) Arreglos para controlar cuando tocan kick y snare
[1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0] @=> int kickH
[0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1] @=> int snareH


while (true)
{
  0 => int beat;
  while (beat < kickHits.cap())
  {
    //toca el kick basado en el valor del arreglo
    if (kickHits[beat])
    {
      0 => kick.pos;
    }
    //toca el snare basado en el valor del arreglo
    if (snareHits[beat])
    {
      0 => snare.pos;
    }
    //siempre toca el hihat
    0 => hihat.pos;
    //avanza el tiempo para que el sonido sea emitido
    tempo => now;
    //Incrementa al siguiente tiempo
    beat++;
  }
}
```

PRUEBA ESTO Añade un arregllo hatHits lleno de 1s y 0s y añade el

código para tocar el hihat, condicional a cada entrada del arreglo. Cambia los 1s y 0s en todos los arreglos para hacer tus propios patrones de batería. Aún más, prueba cambiando las duraciones de los patrones, cambiando los largos de los arreglos. Asegúrate de que todos tengan el mismo largo después de que hayas hecho los cambios, o podrías obtener un error de "Array index out of bounds" (índice del arreglo fuera de rango).

Has mejorado tu máquina de ritmos usando lógica y estructuras de bucle y poniendo variables lógicas en arreglos, correspondientes a patrones de batería que tus bucles internos pueden tocar automáticamente. Pero hay mucho más que puedes hacer. A continuación, aprenderás un operador matemático que puede ser extremadamente útil para hacer patrones musicales.

4.5 Una nueva herramienta matemática/musical: el operador modulo

Digamos que quieres tocar un tambor en particular cada cuatro tiempos, o un tiempo por medio cada N tiempos, con N arbitrario. Para darte otra herramienta de ChucK y hacer que tu máqiuna de ritmos - y composiciones en general - sea aún más flexible, vamos a introducir un nuevo operador matemático llamado modulo. Denotado por el símbolo %, modulo es un tipo de división. A diferencia de la división normal, modulo retorna el resto de una operación de división. Por ejemplo 9 % 4 retorna el número 1, porque 9 dividido por 4 es 2 con un resto de 1. Similarmente, 14 % 4 retorna 2 (3 con un resto de 2, pero modulo solo retorna el resto).

El programa en el listado 4.11 imprimime y "sonifica" tus números modulo para ayudarte a ntender mejor cómo funciona el operador modulo. La sonificación es la muestra de datos o estado de procesos a través de sonidos, siendo ChucK una herramienta muy buena en eso. Aquí usas un par de sonidos click (uno agudo, otro grave) cargados en SndBufs (1). También defines y asignas una variable llamada MOD (2), que será usada (y luego cambiada) para hacer una demostración del operador modulo.

> Una nota sobre estilo de programación
>
> En el listado 4.11, defines y asignas un entero llamado MOD a usar por el resto del programa, sin modificarlo mientras el programa está corriendo. Puedes cambiar la definición inicial, que cambia el sonido, pero MOD es una constante durante cualquier ejecución del programa. Es común en programación llamarle a estas variables constantes con nombres usando solo letras mayúsculas. Es así que MOD es el nombre que hemos seleccionado para esta variable. En nuestro último ejemplo de este capítulo, declararemos y usaremos otra variable constante llamada MAX_BEAT. Mantente atento a su aparición, y recuerda que puedes hacer esto en tus propios programas para hacerlos de más fácil lectura.

En un bucle for con una variable contador llamada beat ascendente desde cero (3), tocas el click más agudo en cada tiempo (4). No obstante, toca el click grave solo cuando la condiciñón beat % MOD (en este caso, beat % 4, que se le e como "beat modulo cuatro") sea igual a cero (5). Esto será cierto en cada tiempo MOD-ésimo, y verás el resultado del sonido sonificado. Sigue la cuenta y ovserva los resultados impresos en la consola.

Listado 4.11 Uso del operador modulo

```
//Matemática de moudlo para músico
//por allguien matemático y musical, 11/11/11

//Crear y cargar un par de SndBufs para sonificar modulo
//(1) Cargar dos archivos de sonido distintos
SndBuf clickhi => dac;
SndBuf clicklo => dac;
me.dir() + "/audio/click_02.wav" => clickhi.read;
me.dir() + "/audio/click_01.wav" => clicklo.read;

//define un número para nuestro modulo
//(2) Modulo limita a MOD
4 => int MOD;

//(3) Compás de 24 tiempos
for (0 => int beat; beat < 24; beat++)
{
  //imprime el tiempo y tiempo modulo MOD
  <<< beat, "modulo", MOD, " is: ", beat % MOD >>>;

  //(4) sonido agudo en cada tiempo
  0 => clickhi.pos;

  //(5) Sonido grave solo en cada MOD-ésimo tiempo
  if (beat % MOD == 0)
  {
    0 => clicklo.pos;
  }
  0.5 :: second => now;
}
```

Escuchando cómo esto suena, debería quedar claro que modulo puede

tener usos musicales; de otro manera, no la habríamos introducido. Modulo puede jugar un rol importante en la construcción de máquinas de ritmo, secuenciadores (bucles que tocan baterías, notas, lo que sea) y composiciones en general. Es así que tienes otra herramienta para usar al determinar cuándo la batería u otros elementos deben reproducirse y también para computar índices cíclicos, como los que podrías necesitar para leer arreglos una y otra vez.

4.6 Uniendo todas las partes: tu máquina de ritmos más genial hasta el momento

Unamos todas las partes que hemos cubierto y hagamos un último asombroso ejemplo de máquina de ritmos para este capítulo. Usarás un número de sonidos de batería a través de SndBufs, y harás paneo stereo usando ambos Gains y el objeto Pan2. El siguiente listado muestra el comienzo de tu programa, donde configuras todo esto.

Listado 4.12a SndBufs y conexiones de paneo para tu gran máquina de ritmos

```
//Máquina de ritmos, versión 4.0
//por un programador con ritmo, 1 de enero, 2099
//Aquí usaremos Modulo % y aleatoreidad para tocar tambores

//define Gains para izquierda, centro, derecha
//declara un arreglo de gains maestros
//izquierda [0], centro [1] y derecha [2]
Gain master[3];
master[0] => dac.left;
```

```
master[1] => dac;
master[2] => dac.right;

//declara SndBufs para los sonidos de batería
//se conectan a distintas posiciones de paneo
//conecta kick al centro
SndBuf kick => master[1];
//conecta snare al centro
SndBuf snare => master[1];
//conecta hihat a la derecha
SndBuf hihat => master[2];
//conecta cowbell a la izquierda
SndBuf cowbell => master[0];

//usa un objeto Pan2 para los aplausos (claps)
//conéctalos a las posiciones de paneo
SndBuf claps => Pan2 claPan;
//conecta el canal izquierdo (0) al gain maestro izquierdo
claPan.chan(0) => master[0];
//conecta el canal derecho (1) al gain maestro derecho
claPan.chan(1) => master[2];

//carga todos los archivos de sonido
me.dir() + "/audio/kick_01.wav" => kick.read;
me.dir() + "/audio/snare_01.wav" => snare.read;
me.dir() + "/audio/hihat_01.wav" => hihat.read;
me.dir() + "/audio/cowbell_01.wav" => cowbell.read;
me.dir() + "/audio/clap_01.wav" => claps.read;
```

Una vez que hayas declarado todos tus sonidos de batería, conectados a ganancias, objetos de paneo y el dac, y una vez que hayas cargado todos tus archivos de sonido, necesitas declarar variables que usarás en tu bucle principal para controlar tu batería. En el siguiente listado declaras un arreglo para controlar lógicamente los golpes al cencerro (1). Luego

declaras otras variables globales que usará la batería.

Listado 4.12b Configuración de variables para tu gran máquina de ritmos

```
//(1) Arreglo para controlar el toque del cencerro
[1,0,1,0, 1,0,0,1, 0,1,0,1, 0,1,1,1] @=> int cowHits[];

//contorla el largo global de nuestros "compases"
//en mayúsculas, ¿te acuerdas?
cowHits.cap() => int MAX_BEAT;

//numero modulo constante para controlar kick y snare
4 => int MOD;

//control general de velocidad
0.15 :: second => dur tempo;

//contadores, tiempo (beat) dentro de cada compás (measure), y
0 => int beat;
0 => int measure;
```

En el bucle principal, mostrado en el siguiente listado, de tu gran máquina de ritmos, vas a usar una mezcla de todas las técnicas que has usado hasta ahora: un bucle, un contador de tiempos, lógica y condicionales, arreglos y el operador modulo.

Listado 4.12c Bucle principal para realmente tocar patrones de batería

```
//Bucle infinito principal de batería
while (true)
{
  //reproduce el bombo en todos los tiempos principales (0, 4,
```

```
//(1) Usa MOD 4 para reproducir el bombo cada cuatro tiempos
if (beat % 4 == 0)
{
  0 => kick.pos;
}
//después de un rato, reproduce la caja en los tiempos 2, 6,
//(2) reproduce la caja solo en algunos tiempos específicos
if (beat % 4 == 2 && measure % 2 == 1)
{
  0 => snare.pos;
}
//después de un rato, reproduce aleatoriamente el hihat o el
//(3) reproduce el cencerro y el hihat solo después del prim
if (measure > 1)
{
  //(4) reproduce el cencerro, controlado por un arreglo
  if (cowHits[beat])
  {
    0 => cowBell.pos;
  }
  //(5) si no es el cencerro, entonces el hi-hat
  else
  {
    //(6) el hi-hat tiene una ganancia aleatoria
    Math.random2f(0.0, 1.0) => hihat.gain;
    0 => hihat.pos;
  }
}

//después de un rato, reproduce aplausos aleatoriamente espa
//(7) reproduce los aplausos solo en algunos compases y tiem
if (beat > 11 && measure > 3)
{
  //(8) los aplausos tienen un paneo aleatorio
  Math.random2f(-1.0, 1.0) => claPan.pan;
  0 => claps.pos;
```

```
    }
    //(9) espera un tiempo
    tempo => now;
    //(10) y luego actualiza el contador de tiempo (MOD MAX)
    (beat + 1) % MAX_BEAT => beat;
    //(11) incrementa el contador de compases en cada nuevo compa
    if (beat == 0)
    {
      measure ++;
    }
}
```

USas un arreglo para controlar la reproducción del cencerro (1) en el listado 4.12b. Usas modulo para tocar el bombo en los tiempos 0, 4,8 y 12 (2) y la caja en los tiemops (2, 6, 10 y 14) (3), pero tocas la caja solo durante los compases impares - más sobre esto pronto. Para darle una estructura de composición a tu canción en batería, usas más lógica condicional (declaraciones if), comprobando el número de compás; que aumenta en uno cada MAX_BEAT tiempos, para comprobar si se deben tocar otros instrumentos. Por ejemplo, una vez que el contador de compases es mayor que 1 (4) empiezas a tocar o cencerro (5), según lo determinado por el arreglo (5), o hihat con ganancia aleatoria (6), (7). Cuando el contador de compases es mayor a 3, añades aplausos con paneo aleatorio (9), pero solo en los tiempos 12 a 15 (8).

Después de avanzar el tiempo según la duración tempo (10), todos los tambores pueden sonar, si es que son gatillados al hacer ChucKing de 0 a .pos, usas el operador matemático modulo para actualizar tu contador de tiempos (11). Al incrementar beat, y luego tomar el modulo de ese número con MAX_BEAT y grabando de vuelta ese valor en beat, automáticamente vuelves beat a cero cuando alcanza MAX_BEAT. Si el resultado de esa

operación modulo en beat es cero (12), entonces incrementas el contador de compases, para que puedas usarlo en tu lógica para darle a tu canción una creciente tensión en la composición (3), (4), (8).

4.7 Resumen

Ahora sabes cómo hacer sonidos realísticos en ChucK, usando SndBuf para cargar y reproducir archivos de sonido, o samples. Aprendiste cómo cargar y reproducir sonidos usando lo siguiente:

- El método .read de SndBuf, en conjunto con me.dir(), para cargar un archivo de sonido desde la memoria
- El método .pos para definir tu posición de reproducción
- El método .rate para controlar cuán rápido y cuán hacia adelante o hacia atrás se reproduce un archivo
- SndBuf2 para archivos stereo
- Uno o varios SndBufs para múltiples archivos de sonido
- El operador matemático modulo

Ahora que hemos programado nuestro primer secuenciador de batería, tu programación para composición puede avanzar a un nuevo nivel de diversión y expresión. Fue un ejemplo bastante complejo, pero fue hecho a partir de todo lo que hemos usado hasta el momento en nuestra travesía de aprender ChucK. Los resultados son decididamente asombrosos. Te motivamos a juntar más items que has aprendido hasta el momento y modificar este ritmo techno expresivo para solidificar el conocimiento que has obtenido hasta ahora.

Podrás estar preguntándote si existe una manera de factorizar un poco de ese código, algún tipo de atajo o método para agrupar cosas, para hacer tu

código más fácil de leer y reusable. Resulta que esta es la dirección que tomaremos a continuación. Hemos estado hablando de métodos (funciones) todo el tiempo, como cuando defines .gain o .freq o .pos. A continuación, vas a aprender cómo escribir tus propias funciones para hacer tu vida más fácil y darle superpoderes a tus programas. Nos vemos en el capítulo 5.

Otras lecturas recomendadas

- Ken Pohlmann, Principles of Digital Audio, 6th Edition (McGraw Hill, 2010).
- Perry Cook, Real Sound Synthesis for Interactive Applications (A K Peters LTD, 2002).
- Ken Steiglitz, A Digital Signal Processing Primer (Addison Wesley, 1996).
- Robert Bristow-Johnson, "Wavetable Synthesis 101, A Fundamental Perspective", AES 101 Conference, 1006. Disponible en http://musicdsp.org/files/Wavetable-101.pdf.