y Jamie McCornack

Small Sounds

One common and terribly annoying shareware feature is Big Sound. You download a 900K game from your favorite online service (and it takes a while, and you're paying for the time), save it on the 20 meg currently free on your GameHD partition, and discover you really got a 200K game with a 700K ‘snd’ resource for the splash screen background music. Grumble. Are you going to save that game? Are you going to recommend it to your friends? Are you going to send in the requested $15 registration fee? Neither am I.

I may take particular offense if that 30-second background clip came off the Tank Girl soundtrack CD. I figure I’m paying by the byte, so the shareware author is charging me $11.67 for 30 seconds of music and $3.33 for the rest of the game, and I could take my $11.67 down to Tower Records and buy the whole album! Oh yeah, and I doubt the shareware author is paying any licensing fee to Stomp, and the tone is kind of flat, like the CD is playing on a stereo and the game developer held the Mac microphone in front of the speakers.

ETHICS ALERT!!! IF YOU FIND ETHICS OFFENSIVE, DO NOT READ THE FOLLOWING PARAGRAPH!

Well, I’m not going to get heavy into the ethical issue here, but the issue exists, and I personally resist paying someone for content stolen from someone else. I’m tolerant of little snatches snipped from hither and yon (and so is the music industry, as demonstrated on your local rap music station), as per the copyright doctrine of Fair Use, but taking somebody’s performance and song as content for your own game is Going Too Far. The pros either license music, hire a music developer, or compose their own music in-house. If I hear the theme from The X-Files (or even The Twilight Zone) playing as background music, there’d better be a licensing notice attached, or I’ll be deeply unmoved by insistence that I demonstrate my personal integrity by mailing money to the game developer. And now, back to programming issues, which are already in progress…

Once you’ve chosen your music content, you’ve got to get it on your game. There are four basic ways you can do this.

Many big publishers prefer CD-ROM games with streaming audio. This technique offers fabulous sound quality and long, complicated background music, but interferes with other data transfer. If a streaming audio background is playing (possibly in conjunction with a QuickTime movie), the next level isn't loading, so streaming audio can slow down gameplay.

For major publishers, streaming audio has one big benefit: since it demands huge files (around 2-1/2 meg per minute) and since CDs are cheap to manufacture and have lots of room, streaming audio provides some brute — force copy protection. Go ahead, put on half an hour of music — it won't increase the cost of pressing the CD, and 75 megabytes of music will surely deter those loathsome pirates from dragging your entire game onto their hard disks. If you're doing a CD-ROM game, this is an avenue well worth investigating.

MIDI is gaining popularity, though more with multimedia than true games, because it provides a whole lot of music from teensy little files, and since a MIDI player is built into QuickTime, MIDI is relatively easy to implement (okay, relatively easy — may I refer you yet again to Tricks of the Mac Game Programming Gurus, ©1995 Hayden Books for explanatory text, and tools on the CD in the back of the book). The disadvantage is a limited range of instruments and one instrument per channel. "Limited" is a relative term too, and along with the usual bagpipes and glockenspiel, you get gunshots and bird chirps and a variety of arcane tone-producing devices. For many music projects, QuickTime MIDI is more than sufficient.

The third technique is to put the whole song in a 'snd' file. This used to be popular with floppy disk commercial games, and still shows up in a lot of shareware. Unfortunately, this technique hogs storage space and RAM, and any compromises made for space will compromise sound quality.

Voice doesn't demand a lot of tone quality, and if your speaking game characters are robots, air traffic controllers, or cowboys who spend a lot of their time smoking those little cigars, 8-bit sound at 5.5 khz sample rate will do just fine. But if you want music, 8-bit at 22 khz is about the minimum the ear will accept. That works out to one minute of music on one 1.4 meg floppy. That doesn't leave much space for the game, does it?

The fourth technique is the subject of this month's column; it's called sequencing. It involves breaking music down into little pieces for storage, and letting the program reassemble those pieces for playback. It's rather the way that speech works (or speech programs), by taking a few dozen distinctive sounds and stringing them together into words, then stringing the words together into sentences…

Remember when you first started noodling around on the guitar, and discovered that 90 percent of popular music could be played with just three chords? Most music is embarrassingly repetitive. Background music is particularly simple stuff; we're not playing the New World Symphony here, we don't want music so complex and fascinating as to distract from the game. A couple dozen four-beat bars can be assembled in enough different ways to keep the game player unaware of hearing things which were heard before.

The best real-life example is John Calhoun's Glider Pro, published by Casady & Greene. It's a good example because it has pleasant and hummable background music, and because it goes on and on without repeating itself, and it's a great example because John is a straightforward guy who doesn't hide his talents behind custom resource types. You can open Glider Pro with ResEdit or Resourcerer, and you'll find the 'snd' files laid out in front of you. Along with the usual bangs and kablooeys you'd expect to find in a game, you'll find a number of music snippets. If you play them one at a time, they don’t sound like much, but assembled together (with some of the snippets played just once or twice, but with many snippets played many times in many different orders) they provide a catchy little tune — a tune which is long enough that when it repeats, it seems friendly and familiar instead of oh-no-here-we-go-again.

To show the basics of sequencing, we’re going to dust off the clams again. ClamMusic and ClamMusicBig are identical to HelloWorld6 (which you’ve grown to know and love over the last couple months), with the addition of some startup music.

The music is InfoHighway1, which is a sample from Optical Media International’s audio capture software for CD-ROM drives, Disc-To-Disk. It’s a very useful program for us game developers. You load an audio CD in your CD-ROM drive, Disc-To-Disk asks you which track you want to capture, what sample rate, what file format you want (including ‘snd’ files, among many others), and whether you want 8- or 16-bit sound. Click “save” and voilá! Music on your Mac, ready to install in your own programs.

Of course, some people won’t be able to resist using Disc-To-Disk to make Eric Clapton the background guitarist for their upcoming remake of Missile Command, but that’s not really what it’s for. Most licensable music comes on audio CDs, and if you look in the back of a multimedia magazine, you’ll find ads from a veritable plethora of music content providers. They offer entire albums full of 30 second to three minute music beds and cues, described in such commercial music terminology as, “Medium tempo groove with laid-back rock drums," and, “Fast-paced music bed with strings and rhythm section.” The licensing agreements generally give you non-exclusive rights to use the music in your own product, providing your product isn’t another sound or music library, and the fees are commonly in two digits ($10 to $99).
The fee for InfoHighway1 (“Computer-techno rhythmic bed with whooshes and streaks”) is included with the purchase of Disc-To-Disk, so no fair snagging it off ClamMusicBig for your own game unless you’re an Optical Media International customer.

ClamMusicBig adds the entire InfoHighway1 track to the HelloWorld resource file as ‘snd’ resource #4000. It is saved in 8-bit 22 khz format to keep it from getting too out of hand, but it’s still 727,692 bytes. The ClamMusicBig application has grown from HelloWorld6’s 213k to 924k, which seems excessive to me for the addition of a half-minute computer-techno rhythmic bed, even allowing for the whooshes and streaks.

Still, it was an awfully easy piece of programming. I added…

```
#define	rMusicSndID
4000
```
…to the declarations, commented out the previous background sound…
```
PlayASound(rFootstepSndID, kLowSoundPriority);
```
…from ClemLoop(), and added…
```
PlayASound(rMusicSndID, kLowestSoundPriority);
```
…right after the InitAll() call in the main function. It sure was easy, and it merely quadrupled the size of the application.
ClamMusic is a bit more complex, but only a bit. Instead of one huge ‘snd’ resource for the music, it has seven little ones, numbered…
```
#define	rDown1SndID
4001
#define	rDown2SndID
4002
#define	rMedium1SndID
```

```
4003
#define   rMedium2SndID
4004
#define   rUp1SndID
 4005
#define   rUp2SndID
 4006
#define   rFade1SndID
4007
```

…each around 19k, four beats, and .866 seconds long. I'll admit there was some trial and error involved in coming up with the .866 second figure. I used Macromedia's SoundEdit Pro to pick the InfoHighway1 file apart, and relied heavily on the visual feature (the spikes that showed up when individual instruments went dit-dit-dit) to measure and time what four beats were, but still there were a couple hours of cut-and-paste before I got satisfactory results. Satisfactory to me, that is; if you're not up at 3 a.m., your satisfaction may demand higher artistic standards.

So here we have a couple of music snippets that go down, and a couple that go up, and a couple that don't change much in pitch from beginning to end, and one that is quieter than the rest. And we have some new functions, named…

```
void DoBackgroundMusic0(void);
void DoBackgroundMusic1 (void);
void DoBackgroundMusic2 (void);
void DoBackgroundMusic3 (void);
void DoBackgroundFadeOut (void);
```

…of which DoBackgroundMusic0() is typical:

```
void DoBackgroundMusic0 (void)

 AddSoundToQ(rFade1SndID);
 AddSoundToQ(rFade1SndID);
 AddSoundToQ(rFade1SndID);
 AddSoundToQ(rMedium1SndID);
 AddSoundToQ(rUp2SndID);
 AddSoundToQ(rMedium1SndID);
 AddSoundToQ(rMedium2SndID);
 AddSoundToQ(rDown1SndID);
 AddSoundToQ(rMedium1SndID);
 AddSoundToQ(rMedium2SndID);
```

Collecting all these stanzas together, we have…

```
void DoBackgroundMusic (void)

 DoBackgroundMusic0();
 DoBackgroundMusic1();
 DoBackgroundMusic2();
 DoBackgroundMusic3();
 DoBackgroundMusic2();
 DoBackgroundMusic1();
 DoBackgroundMusic1();
 DoBackgroundMusic2();
 DoBackgroundFadeOut();
```

…right after the InitAll() call in the main function. Sound familiar?

All in all, ClamMusic makes about 90 calls to AddSoundToQ() (a Feeping Creatures routine from MGWSound.c), which stacks up all those music snippets and plays them in order. Thus we get 90 four-beat bars of music, with only seven four-beat bars worth of storage. I think when you run the ClamMusic application, you'll find it captures the feel of InfoHighway quite nicely—though it's admittedly weak regarding hummability—and the application size on disk is only 345k.

The code and resources are in the Code Folder, along with CodeWarrior 68K C projects for you Metrowerks users. For Disk-To-Disk sales information, call Optical Media International at 800-347-2664 (it costs around a hundred bucks). Nope, I'm not connected with them, I'm just a satisfied user. If you have questions about this column (or ideas for it, or a product you feel would be appropriate for review) you can find me at MacGameDev on AOL (that's macgamedev@aol.com from other services).