CHAPTER 3

Insert Quarter to Continue: 16-Bit and the Death of the Arcade

By the late 1980s, video games were no longer viewed as a novelty or passing fad. The Nintendo NES had sold over sixty million consoles, and the Commodore 64, whose success was built on games, had become the best-selling personal computer of all time, with over fourteen million units sold.[1] Games were big business, and competition was fierce. Notes Martin Campbell-Kelly in his history of the software industry, ''A few hit products, such as Brøderbund's *Choplifter* and Sierra On-Line's *Frogger*, stayed on the best-seller lists for more than a year, but the average life of a computer game was about 3 months and the average sales were less than 10,000 units'' (Campbell-Kelly 2004, p. 226). Games had a short life cycle, resulting in a constant desire for new titles coupled with very high risk. By the close of the decade, Atari had lost its grip on the industry, and responded by joining forces with Sega in an antitrust lawsuit against Nintendo, who by that time controlled more than 80 percent of the home console market (Oxford 2005). Nintendo held its developers under strict licenses that prevented them from developing for other systems, creating a monopoly atmosphere that shut out competitors. Nintendo ultimately lost the suit, and was forced to pay out over $25 million (Oxford 2005). It was not the first time game companies had ended up in court. One employee was cited as defining Nintendo's business as ''games and litigation'' (cited in Kline, Dyer-Witherford, and de Peuter 2003, p. 115). Part of the legal problem was the inability of the existing copyright laws to protect games, since in many countries, such as the United States, the law explicitly stated that it did not protect ''games,'' ''methods of operation'' (and the method of play), ''ideas,'' and ''utilitarian'' aspects of pictorial works.[2] Canada's first lawsuit, *Atari Inc. v. Video Amusements of Canada Ltd.*, came as early as

1982, and questioned whether copyright could in fact be applied to the source code of a game. Summarizes the Canadian law firm Davis and Company,

> The defendant's game units would flash the ''Atari'' logo, and when Atari sought an injunction the judge stated that there was clear evidence of copying. However, the defendant raised the issue as to whether existing Canadian copyright laws extended to game machines or pieces of equipment, EPROMS, source code or object code. Because this was a question of major importance to copyright law, and because damages would be an appropriate remedy should the plaintiff succeed at trial, the judge denied an injunction and stated that the issue of whether a video game was copyrightable should be sent to trial. [However] There is no record of this issue ever being sent to trial.[3]

The implications of the laws and the nature of the industry led to a climate of secrecy, protectiveness, and fear, made worse by the failing arcade coin-op industry. The arcades had previously been the first to launch new technologies, but the introduction of the next generation of games, the 16-bit machines, heralded an era of the home console and cast a near-fatal blow for the arcades. When Nintendo pulled out of the arcade market to focus on their console business in 1992, the coin-op industry was dealt another symbolic blow. In 1994, there were still some 860,000 arcade game units in the United States, earning $2.3 billion a year, but a decade later, those numbers would drop by more than half, to 333,000 and $866 million.[4] With significant advances made in the technology of home consoles, there was simply less of a reason for gamers to visit the arcades.

One of the major audio advances of the 16-bit era was the introduction of *Frequency Modulation* (*FM*) synthesis (see box 2.1, ''Sound Synthesis''). FM synthesis had been developed by John Chowning at Stanford University in 1967–68, and licensed and improved on by Yamaha, who would use the method for a variety of computer sound chips, as well as their DX series of keyboards. FM used a modulating wave signal to change the pitch of a second sound wave. Many FM chips used four or six different *oscillators* (creating the waveforms) for each sound, to generate more realistic sounding instruments than previously heard on sound chips.[5] These chips found their way into many of the arcade games of the mid-1980s, and into most computer soundcards of the era. Compared to the subtractive synthesis PSG chips of the 8-bit games era, FM chips were far more flexible, offering a wider range of timbres and sounds. Moreover, they allowed for more realistic sounding sound effects. The attributes of FM synthesis were particularly well suited to organ and electric piano sounds, pitched percussion, and plucked instrument sounds, and, as shown below, these instrument sounds dominated those games machines that relied on FM synthesis.

Arcades of the 16-bit era typically used one or more FM chips, as well as pulse code modulation (PCM) voice synthesis chips.[6] Some companies, such

as Nintendo and Konami, used custom-made speech chips for their arcade games. Walking around the arcade in the late 1980s, the machines would literally call out to players, begging to be played: SNK's *Guerilla War* (1987) had victims calling out, ''Help me!'' But these sounds were to be replaced with the new sounds for the next generation—the sounds of the home console.

NINTENDO AND SEGA: THE HOME CONSOLE WARS

The first 16-bit home console was released by the multinational communications corporation NEC in Japan in 1987 as the PC Engine. The PC Engine, or TurboGrafx16 as it became known in North America, was not true 16-bit, but rather had dual 8-bit processors. Nevertheless, it did have a 16-bit graphics chip as well as six-channel stereo sound, and was popular in Japan for a brief time. When it came time for a North American release, however, the TurboGrafx16 did not fare so well. Part of the reason for this lack of success was Nintendo's exclusive contracts, which restricted developers from producing games for other companies or console systems.[7] The TurboGrafx16 managed to sell close to one million units, but when true 16-bit systems were released, such as the Sega Genesis (or Mega-Drive as it was known in Europe), it was left far behind.[8] Sega had previously enjoyed some success in Europe with their 8-bit Master System, but could not compete with the NES in the North American market. Sega did, however, own a much bigger segment of the coin-op arcade market, and could rely on these games to sell its consoles. As David Sheff (1993, p. 352), author of a history of Nintendo's rise indicates,

> Sega's coin-op arcade games were key to the console's success; ''[Nintendo's] Yamauchi underestimated Sega, whose executives understood the importance of software to drive hardware sales ... Sega had simply taken the design of its 16-bit arcade machines and adapted it for Genesis. It could therefore boast not only such 16-bit features as high-definition graphics and animation, a full spectrum of colors [512] ... two independently scrolling backgrounds that created impressive depth-of-field, the illusion of three dimensions, and near CD-quality sound, but also a proven software catalogue: Sega's arcade hits.''

The Genesis produced many games ported from successful Sega arcade titles like *Space Harrier* (1985), *After Burner* (1987), and *Ghouls 'N Ghosts* (1988). The system originally came packaged with the arcade hit *Altered Beast* (1988), but soon took on Nintendo's popular *Mario* character head-to-head with their *Sonic the Hedgehog* (1991). The Genesis also had superior sound to the 8-bit NES. It had an equivalent PSG 3+1 chip to handle effects and the occasional music part (a Texas Instruments SN76489, which had been used in the ColecoVision), as well

as a Yamaha FM synthesis chip, a YM 2612, which had six channels of digitized stereo sound, and one PCM 8-bit sample channel (the same chip used in the popular Yamaha DX27 and DX100 music keyboards).

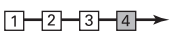
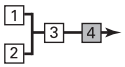
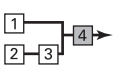
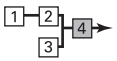
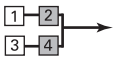
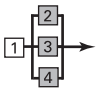
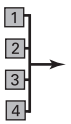
The Sega Genesis was the most popular game console to employ FM sound. Although sampled vocals still sounded somewhat scratchy on the machines (rather like a detuned radio), they were a significant improvement over those of the NES, and along with the Genesis' advanced graphics there was a clear distinction for fans between the games of the 8-bit era and those of the 16-bit era. Sound on the Genesis was rather difficult to program, however, requiring assembly language code to engage the two chips. Masato Yakamura, composer of *Sonic the Hedgehog*, recalls, ''Nowadays, you'd just be able to send the data through email, but at the time, I had to record onto cassette. *[laughs]* The sound engineer would then listen to them, and reproduce them for implementation on the Genesis. Then, they'd send me back a bare game chip, and then I'd listen to it and check it. It's sort of unthinkable, now, but at the time, we just did that over and over until we got to the finished product.''[9]

Instrument sounds and sound effects had to be coded by hand in a cumbersome format,[10] and thus sounds and instruments were used many times in different games to save coding new ones. The *Genesis Technical Manual* described the most common instrument sounds (see table 3.1). Each FM channel had four *operators* (waveforms) assigned, each with a frequency and an envelope with which to modify the input. Operator One (the oscillator, or carrier wave) could feed back into itself, creating a more complex waveform, and thus, a different instrument sound. The other operators served as modulating waveforms (known as *slots*). The oscillator would produce a basic sine-wave tone, and the modulators could then affect the output waveform, altering the complexity of the waveform and therefore its timbre. There were eight different algorithmic configurations of these four operators, presented with suggested instrument uses (table 3.1). Although they could be used for other instrument sounds, these suggested sounds became the most common on the Genesis.[11]

The result of the assembly programming and the eight algorithms meant that many of the same types of sounds were used over and over in games. The same ''fat square-wave-like organ sound'' is used in the *Shadow of the Beast II* (Psygnosis 1992) game discussed below, as well as other Psygnosis games, like *Fatal Rewind* (1991), while the ''flute-like sound'' from the same game was also used in *Misadventures of Flink* (1993), and so on.

Once the instruments were developed, the musical routines (sequences) could be coded, often using loops and transpositions to save time and system memory. In the ''Labyrinth'' section of *Sonic the Hedgehog*, for instance, there is repeated use of various sequences that are transposed, which are part of an overall ABAC song pattern. Typically, one channel handled percussion, one bass, one the melody, and others were used for filler chords, or arpeggios. The multiple

Table 3.1
Eight algorithmic configurations of the FM chip's operators, from the *Sega Genesis Technical Manual*. Slots are indicated by shading.

Algorithm #	Layout	Suggested uses
0		Distorted guitar, ''high hat chopper,'' bass
1		Harp, ''PSG''-like sound
2		Bass guitar, electric guitar, brass, piano, woodwinds
3		Strings, folk guitar, chimes
4		Flute, bells, chorus, bass drum, snare drum, tom-tom
5		Brass, organ
6		Xylophone, tom-tom, organ, vibraphone, snare drum, bass drum
7		Pipe organ

channels were also used to create various effects, which were common on the Genesis. The most popular effect seen was a kind of *double-tracking*,[12] in which a slight delay between channels ''fattens up'' the instruments, creating a fuller sound. A more significant delay could be used to create phasing effects, or even a kind of flanging.

Song structures on the Genesis were not significantly different than those of its 8-bit predecessors or those of other 16-bit systems. Typically, level gameplay had the longest loops, while boss music had the shortest loops, often eight to ten bars, as in figure 3.1, which also used a double-tracking effect, and illustrates very typical usage of the YM chip's six channels.

Transitions between the loops on the Sega Genesis also did not significantly differ from those of the 8-bit games. The most common transitions were still hard cuts, although there were also some rapid fade-outs of the original cue before the

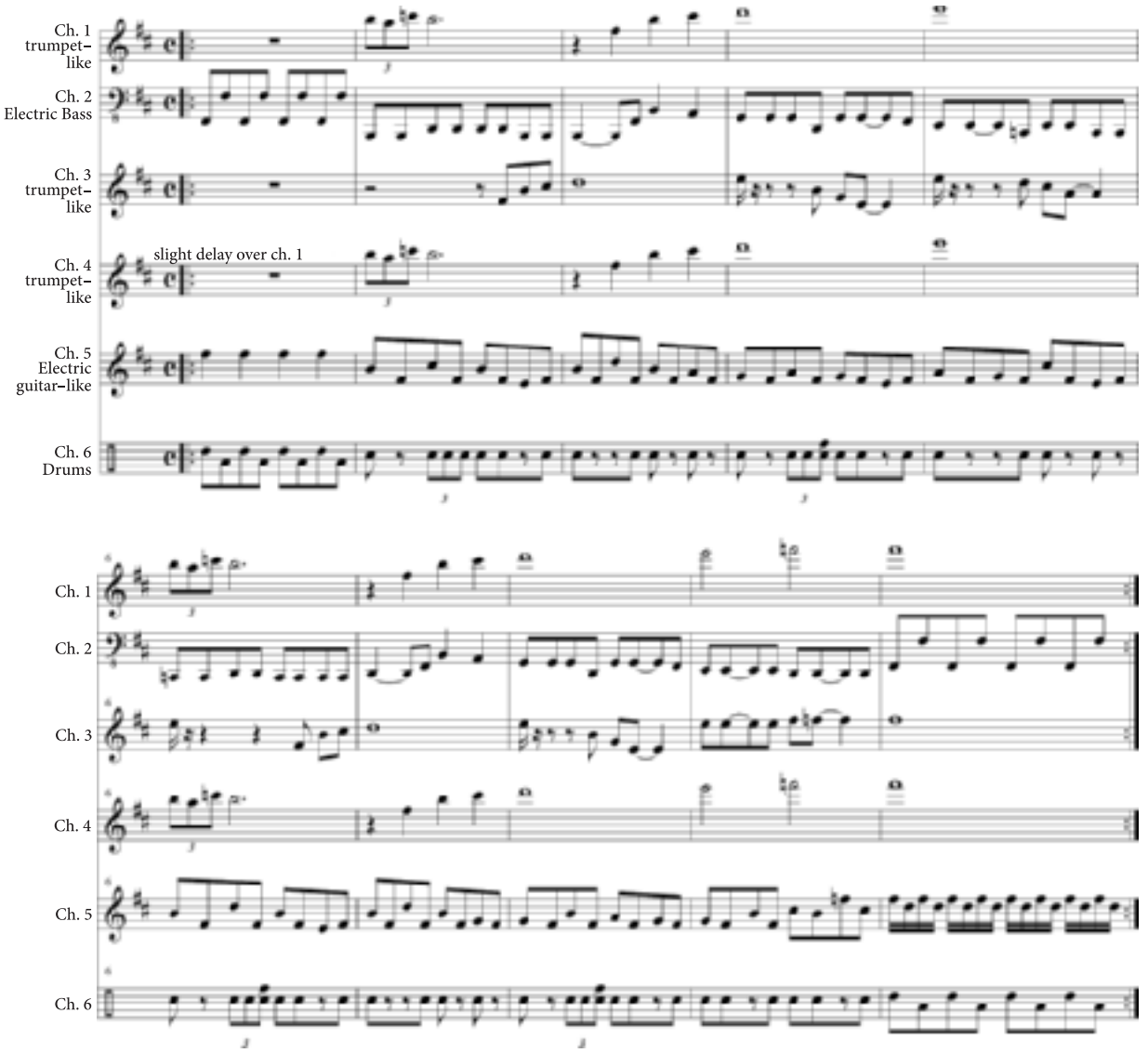


Figure 3.1
Boss music from *Sonic the Hedgehog* (Masato Nakamura, Sega, 1991), showing very typical use of the Genesis chip's channels.

new cue began. It could be argued, in fact, that apart from the increased quality of the sound (in terms of more ''realistic'' sounding instruments), and the extra channels, there was little significant change from the music of the 8-bit games. In terms of sound design, however, Sega's digital sample channel enabled a far more advanced repertoire. Sound effects became more realistic sounding, and vocal samples were far in advance of their 8-bit predecessors. A few Sega games managed to experiment with interactivity between the music and the player in some way. *Toejam and Earl: Panic on Funkotron* (Sega, 1992), a music side-scrolling platform game, for instance, allowed the player to select the initial game music (by John Baker), and had the player ''jam out'' to the percussion in the funky score in a simple *Simon*-type memory game at several stages in gameplay. In a few other games, the player had some minor interaction with the music, but the most interesting example came from the *Looney Tunes* game, *Desert Demolition* (Blue Sky, 1995). Reminiscent of *DigDug* discussed in chapter 2, the player selected a character to control (Wile E. Coyote or the Road Runner), and the sound then *mickey-moused* the character (that is, followed the character's movements), speeding up or slowing down, starting and stopping with the player's actions. The music was very simplistic, but nevertheless the game represented one of the most interactive soundtracks produced on the Genesis.

Perhaps the most distinguishing feature of Sega Genesis audio was the adoption of progressive-rock stylistic traits. Occasionally, graphics on the Sega made some reference to progressive rock, such as those of *Shadow of the Beast II*, which recalled the Roger Dean album covers for Yes. More significant, though, is that elements of progressive rock instrumentation entered the Genesis sound. Since the chip could somewhat accurately mimic the common progressive rock instruments, it is perhaps not surprising that a progressive rock sound was used. Rather than using a guitar lead sound like that common to the Commodore 64, the Genesis more commonly used keyboard instrument sounds, which had contributed significantly to the progressive aesthetic, used primarily for supplying background chords and for stating thematic material (Macan 1997, pp. 33–34). Providing the rhythm section were bass guitar and drums, and these were often accompanied by various percussion sounds (such as woodblocks, bells, chimes, and so on, ''to create tone colours rather than emphasize the beat'') (ibid., pp. 37–38). Flute and violin sounds were then used 'in an obbligato role (most often to supply instrumental commentary between vocal stanzas) or as an alternate lead instrument to the keyboards and guitar' (ibid., p. 37). A keyboard-like lead, organ sounds, and unusual wood-block or bell/chime percussion were staples of Genesis music, and flute and violin sounds were also quite common, particularly in epic role-playing games, like *Phantasy Star* (Sega, 1988). There were also examples in shoot-'em-ups, like *Vapour Trail* (Riot, 1991), puzzle games, such as *Fatal Rewind*'s Pink Floyd-esque beginnings (Psygnosis, 1991), and platform games like *Shadow of the Beast II*, which used a fat synth sound (slightly resembling an

organ), a flute-like sound, and a bell, with unusual percussive accents (music by David Whittaker).

Song structure on the Genesis (and to some extent other platforms of the era), like progressive rock, deemphasized ''full-blown tunes in favour of other types of melodic material. Large-scale structure in instrumental progressive rock results from the repetition of brief melodic ideas or chord progressions, and the combination of such fragments to produce larger, more complex units'' (Macan 1997, p. 44). Indeed, it was a common element in some game music of the 16-bit era and beyond to avoid anything too ''catchy'' that might become annoying after many repetitions, in favor of various smaller melodic riffs which, collectively, could often be played like a longer epic soundtrack, with each tune thematically and instrumentally tied to each other. It was also quite common to use ground bass, a recurring bass line with variations overtop. As Macan (1997, p. 44) indicates in his study of progressive rock, these variations were commonly four- or eight-bar fanfare melodic riffs. On the Genesis, some games were entirely composed using this concept, such as *Pirates of Dark Water* (Sunsoft, 1994), in which each distinct level in the game had a different theme, but within each level, all the music shared one bass line, with two or three melodic variations, as in figure 3.2, which shows a simple sixteen-bar loop made up of two variations over the bass.

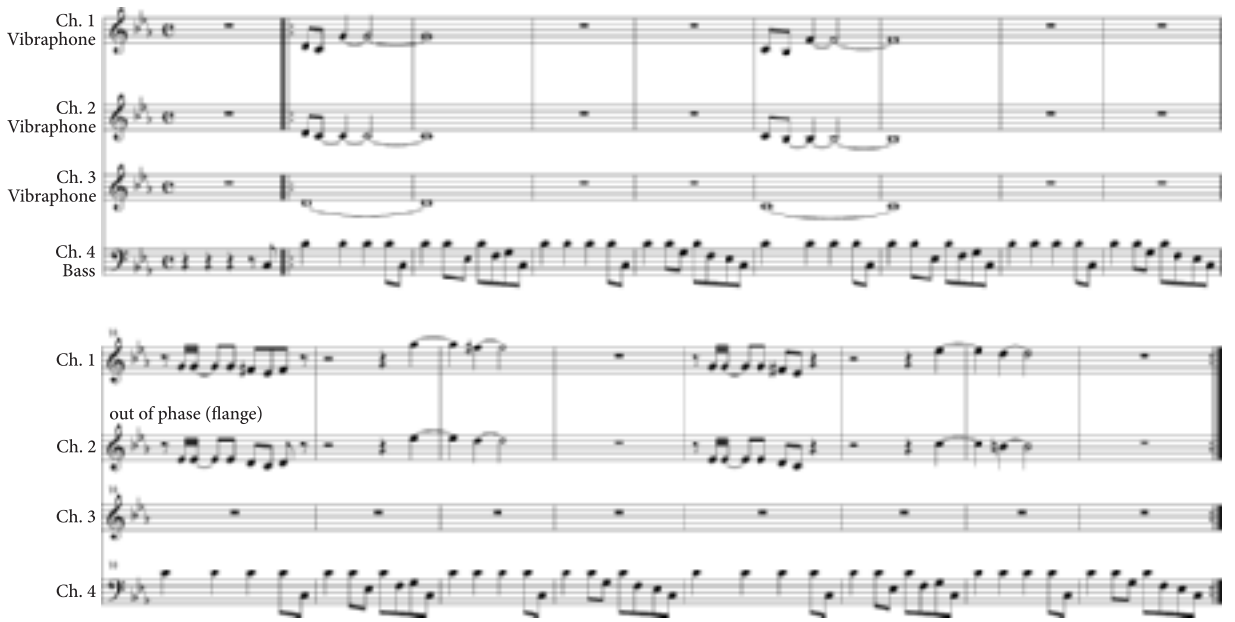


Figure 3.2
Pirates of Dark Water, ''Citadel level'' (Tom Chase and Steve Rucker, Sunsoft, 1994).

One final element worth noting common to both progressive rock and the music of the Sega Genesis was the use of modal harmony, exotic modes, and chromaticism. It is perhaps not surprising that progressive rock—with themes commonly drawn from science fiction, mythology, and fantasy—might use non-Western or archaic tonal elements to signify these ''other'' places and worlds in which the songs were set. Equally unsurprising, then, is the adoption of similar modal elements and chromatic notes in video games, which were also commonly set in fantasy realms, as can be seen in the examples above as well as, more clearly, in figure 3.3, from *Shadow of the Beast II*, which had several pentatonic melodies.

With the Genesis leagues ahead of the NES in capabilities, Nintendo realized that they would have to build a 16-bit system to compete. By 1991, they had developed their Super Famicom, or Super Nintendo Entertainment System (SNES). The SNES was arguably superior in graphics and sound capabilities to the Genesis, and managed to sell 46 million units over the course of the product's lifetime.[13] The SNES sound module consisted of several components, the most important of which was the Sony SPC-700. The SPC-700 was an 8-bit co-processor, with an attached 16-bit Sony *digital signal processor* (see box 3.1, ''DSP''), and a 16-bit stereo DAC. The DSP was essentially a *wavetable synthesizer* that supported eight stereo channels at programmable frequency and

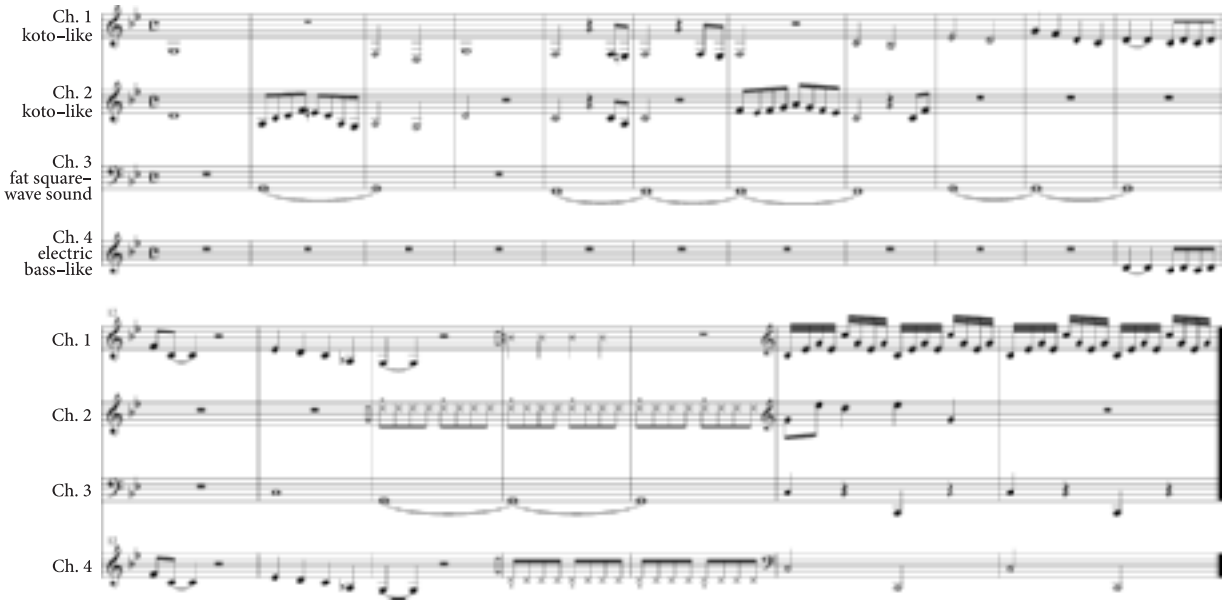


Figure 3.3
Shadow of the Beast II, ''Prison level'' (Tim Wright, Psygnosis, 1992).

Box 3.1
Digital Signal Processing (DSP)

DSP refers to the processing of a signal (sound) digitally. This subject is worthy of a book in itself (see Zölzer, Amatriain, and Arfib 2002 for a more detailed look at these effects). Common game sound effects processing includes:

Echo delayed signals of 50 milliseconds or above. Echo is useful for creating the effect of an enormous cavernous environment.

Reverberation (Reverb) recreates the reflection of sound waves off a solid surface, typically with delayed signal of 50 ms or below. Reverb is used for creating indoor environments.

Chorus a delayed sound added to original with constant delay. Chorus may have one or more delayed sounds, resulting in the effect of multiple voices. Creates several ''layers'' of the same sound, adding depth.

Time stretching altering the speed of a sound without adjusting the pitch.

Compression reducing the dynamic range of a sound. Compressors make loud sounds quieter, and the quiet sounds louder. Compression has been used to extremes in games, although dynamic range is improving. See Bridgett 2008 for more information on dynamic range in games.

Equalization (EQ) attenuating (reducing or eliminating) or amplifying various frequency bands in a sound; used in mixing.

Filtering specific frequency ranges can be emphasized or attenuated. Band-pass (reducing high and low frequencies) creates a ''telephone voice'' effect. Low-pass filters cut high frequencies, while high-pass filters cut low frequencies. Used in mixing, and for creating effects (such as the aforementioned telephone voice).

volume, effects such as reverb, filters, panning, and envelope generators, and had a preset stock of MIDI instruments (see below for an explanation of MIDI). Unlike previous synthesis models used in computer sound, wavetable synthesis used preset digital samples of instruments, usually combined with basic waveforms of analog synths (see box 2.1, pp. 10–12). It was, therefore, much more ''realistic'' sounding than FM synthesis,[14] and with the addition of software to convert MIDI data into files executable by the sound processor, it was much more user-friendly.

Despite appealing to the same market, Nintendo had cultivated an image quite different from that of Sega. The marketing practice of Sega has been summarized as ''smearing its rival as infantile and boring and building it own 'cool' image'' (Kline, Dyer-Witherford, and de Peuter 2003, pp. 130–131). Sheff (1993, p. 358) describes the differences, for instance, in the design of the visual style of the consoles: ''The Genesis, in black, was the outsider, the heavy metal of videogame machines. The SNES, sleekly styled in gray, was commercial and pop.'' This distinction was similar to that of the music and sound effects, which typi-

cally had a different style between consoles. Even though the SNES had wavetable synthesis sound, it still maintained a distinctly ''chip-tune'' and poppy feel, relying on the aesthetic of the 8-bit era, for the most part using squarewave-like sounds rather than relying on more ''traditional'' musical instruments.[15] Whereas the Genesis had more progressive-rock stylistic traits, the SNES leaned toward more popular music of the early 1990s, including dance (such as *Adventures of Dr. Franken* [Elite, 1993]), hard rock, such as *Biker Mice from Mars* (Konami, 1994, which used three electric guitar sounds, three drumkit sounds, and a bass), and even hip-hop, such as *Barkley Shut and Jam* (Accolade, 1993, sound by Rudy Helm and Richard Kelly).

Sega's initial success in the home console market had been reliant to some extent on celebrity-endorsed games, such as *Michael Jackson's Moonwalker* (Sega, 1991) and *Joe Montana Football* (Sega, 1990). This marketing concept was not lost on Nintendo, who released many games with movie tie-ins (*Cliff Hanger* [Sony, 1993], *Demolition Man* [Acclaim, 1995], *Judge Dredd* [Acclaim, 1995], and so on).[16] This celebrity marketing also spilled over into the music, as the SNES relied on more cover songs and licensed music than its predecessors, employing various electronic music artists to write soundtracks (such as Eurodance band 2Unlimited's soundtrack for *Biometal* [European and US versions, Activision, 1993], and Psykosonik's soundtrack for the SNES game, *X-Kaliber 2097* [Activision, 1994]). *Ken Griffey Baseball* (Nintendo, 1994), even included a Jimi Hendrix–style American national anthem. Even classical music was more frequently used on the Super Nintendo, such as *Adventures of Dr. Franken*'s dance version of Beethoven's *Moonlight Sonata*, or *Air Cavalry*'s (Cybersoft, 1995) use of Wagner (a clear reference to *Apocalypse Now*, considering the setting of the game as a helicopter-war game), shown in figure 3.4.

Although there were clear differences in instrumentation, the channel usage on the SNES was quite similar to that of the Genesis. Chords (in octaves) were often simulated by adding an overtone on one channel to fill out the sound. As on the Genesis, the extra channels that the 16-bit consoles had over their 8-bit predecessors were commonly used for fattening up the sound with chords, or to create an ADT-style effect. Dynamic sound activity on the SNES was comparable to that of its predecessors, and the musical structures remained very similar to those of the 8-bit era, despite the increased potentials of the chips. Boss music typically had at most three short sections which repeated, gameplay levels had longer cues, and genres still maintained the same influence over the length of sections and songs.[17] There were, of course, a few exceptions, and some interesting approaches to dynamic activity were seen in a handful of cases. The music for *Super Mario World* (Nintendo, 1990), for instance, used a layering technique when the player's character, Mario, would hop on turtle Yoshi's back, which is discussed further in chapter 8. Although there were not many significant advances made in the construction and playback of game music in the 16-bit

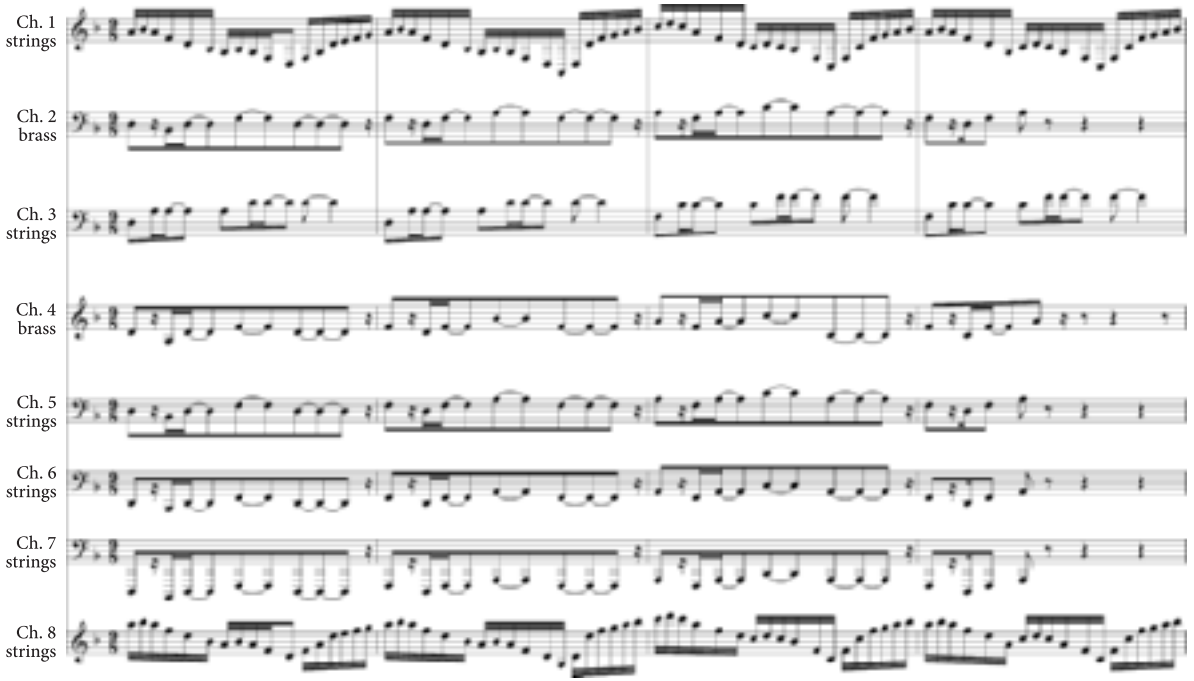


Figure 3.4
Air Cavalry's take on Wagner's *Ride of the Valkyries* (Cybersoft, 1995).

consoles, there were to be some interesting developments in the music of PC games.

PERSONAL COMPUTERS GET MUSICAL

Recognizing that gamers and musicians wanted better sound from their PCs, add-on third-party FM soundcards began to develop in the mid-1980s. Soundcards were designed with the gamer and composer in mind: they generally had a joystick game port that could double as a MIDI port with an adapter. As well, lines in and out for speakers, headphones, home stereos, and microphones were often included, creating advances in not just playback, but also in composition for the home user. Although earlier soundcards had been sold,[18] the first popular PC soundcard was produced by the small Canadian company AdLib Multimedia in 1986. AdLib based their card on the nine-channel Yamaha FM chip, YM3812, which was a later version of the popular YM3526 used in many arcade games. To boost sales, the AdLib card was packaged with a MIDI sequencer program

equipped with 145 preset voices (*Visual Composer*), and an FM synthesis program to design sounds or instruments (*Instrument Maker*).

Soon after the development of AdLib soundcards, a Singaporean company called Creative Technology (now Creative Labs) developed their own soundcard, known originally as the Creative Music System, but marketed in North America by Radio Shack as Game Blaster. The name itself suggests the importance of gaming in the development of new technologies for PC audio. Although Game Blaster had twelve channels, it used the more primitive PSG subtractive synthesis chips, and therefore was quickly abandoned in favor of the Sound Blaster. The Sound Blaster was essentially a copy of the AdLib card, using the same FM chip, but with added digital audio capabilities for sampling and a game port. Sound Blaster quickly became the standard for game sound after a drop in price, although the cards were not without problems (see Weske 2000). One of the difficulties with the Sound Blaster cards was the fact that sound channels had to be mixed down into one or two output channels, resulting in a loss of resolution in the sound. If a file wanted to play eight simultaneous channels, this meant that the sound came out sounding muddied.[19] A more advanced card, the Gravis Ultrasound, using a ''GF1'' Gravis-specific chip, was designed with this problem in mind, and was capable of supporting both MOD and MIDI files (see below), with dedicated digital audio outputs for thirty-two channels. The card worked by loading instrument patches into its RAM, rather than the traditional ROM, which meant that whatever samples the user desired could be uploaded to the card. Unfortunately, however, the chip did not support FM synthesis, and as such was not compatible with all of the games that had been developed for Sound Blaster.[20] With such a variety of soundcards on the market, programming music for games became more problematic. By 1989, at the behest of Microsoft, Yamaha made their FM chips available on the open market, so that a standard sound format for PCs could be created.[21]

Another significant development in soundcards came from synthesizer manufacturer Roland. The MT-32, or ''Multi-Timbre Sound Module,'' released in 1987, was designed as a MIDI soundcard, capable of 32 simultaneous voices, with 128 preset instruments. What made it different from its competitors, however, was that it used the more advanced wavetable synthesis. Roland was given a serious boost when they struck a deal with Sierra On-Line. As well as becoming a reseller of their products, Sierra would adopt both Roland's MT-32 soundcard and the AdLib as standards for their games, beginning with *King's Quest IV* (1988).[22] As they had done in the 8-bit era with the PCjr, Sierra would once again use games to showcase the capabilities of new audio hardware components, this time by bringing on board film and television composer William Goldstein (for *King's Quest IV*), and, later, Jan Hammer (for *Police Quest III*, 1991), among others, to compose for the games. The most significant advance of the 16-bit era

sound, however, was the adoption of the Musical Instrument Digital Interface (MIDI) protocol.

MIDI was defined in 1983 to allow musical devices (synthesizers, keyboards, sequencers, mixing desks, computers, etc.) to be compatible in a standardized format. Only code was transmitted, rather than actual sounds, meaning file size was very small—a distinct advantage for games, which taxed the memory of the machines. A MIDI command might, for instance, tell a synthesizer when to start and stop playing a note, at what volume and what pitch, and what voice (instrument) or sound to use. Initially, some of this information would vary greatly depending on the devices used, which complicated programming for soundcards, but in 1991 a General MIDI (GM) standard was agreed upon. This standard laid out a template for 128 instruments and sound effects, so that the same number setting would be the same on any MIDI device. For example, a command to ''play number 39'' would always result in a slap bass. There were also various MIDI controllers, including volume, tempo, duration, key pressure, and so on, each of which contained a specific number assigned to that control.[23]

The advantages of MIDI for games composers were great. No longer encumbered with the awkward tunings or difficult programming languages, composers could instead write their music on music keyboards. The most important advantage of MIDI, however, was the fact that the audio file consisted of only code, rather than recorded digital audio files (which would come later), and thereby would take up little of a game's limited amount of RAM. There were, however, several complaints about the General MIDI standard. First, the selection was limited to 128 instruments, and some of these were taken up with the seemingly ridiculous sound effects such as ''bird tweet'' (123) and ''helicopter'' (125), which may have been useful for some game sound designers, but were rarely usable for musicians. Roland responded to the limitations of the GM standard by creating General Standard (GS) MIDI later that same year (1991), which would allow for 128 variations of each of the 128 available MIDI channels.

Another problem with MIDI, however, was the fact that most MIDI hardware playback devices sounded quite different. A slap bass on one person's soundcard might sound very different from a slap bass on another person's soundcard, and what sounded good on a composer's specialized soundcard might sound very poor on a cheap card. The timbre, volume, or sound quality would vary (even the form of synthesis might be different, as shown above). Although for composers of console games this was not a problem, since the consoles had the same hardware in every unit, for PC users, this meant widely varying music playback quality. A few solutions were tried: many game composers would write songs using Roland's Sound Canvas, and this became a standard by which to compare MIDI cards. Another solution was to revisit the idea of ''System Exclusive Data'' (SysEx), which had previously been used by some sound devices, and meant that each sound device would have an ID code specific to that device. The playback

device could then read this code and know exactly what hardware configuration to use—in other words, what sounds and what effects were on which channel, so that number 39, voice 16 might mean slap bass with reverb on a specific card. To address compatibility issues, the MIDI device would always default to the standard GM instrument. The trouble with this method was, of course, the time involved in programming for the myriad different devices. Despite these difficulties, MIDI was quite flexible, and led to some quite original ideas, such as the developments at LucasArts, discussed below.

MIDI AND THE CREATION OF IMUSE

LucasArts' iMUSE technology for PC adventure games, developed in 1991 and patented in 1994, was a unique approach to game sound that was leagues ahead of its competitors. What was original about iMUSE was the introduction of dynamic audio components into a scripting language to allow for more appropriate music in games. Like many other PC adventure games developers (such as Sierra On-Line), LucasArts (then LucasFilm Games) developed their own software engine for building adventure games. Called SCUMM (Script Creation Utility for Maniac Mansion), the first engine was created initially for *Maniac Mansion*, an 8-bit game released on the PC and later ported to the Nintendo NES. The SCUMM engine became the basic building block of many of LucasArts' popular games of the late 1980s and early 1990s, including *Indiana Jones and the Last Crusade* (1989), *Loom* (1990), *The Secret of Monkey Island* (1991), and *Day of the Tentacle* (1993). SCUMM was a scripting technology that enabled the use of other add-on scripting advances, such as INSANE (Interactive Streaming Animation Engine, for video), and FLEM (object naming). iMUSE, the ''Interactive Music Streaming Engine,'' was developed by composers Michael Land and Peter McConnell to allow them to create more dynamic music.[24] Explained composer Michael Land: ''The thing that's hard about music for games is imagining how it's going to work in the game. The iMUSE system was really good at letting the composer constantly test out the various interactive responses of the music: how transitions worked between pieces, how different mixes sounded when they changed based on game parameters, etc. Without a system like that, it's much harder to conceive of the score as a coherent overall work'' (cited in Mendez 2005).

The developers of iMUSE identified various difficulties with MIDI that were specific to composers of dynamic audio: ''Although music and sound effects are an important part of the game's 'feel,' the technological progress which has been made in this area has been relatively limited. The use of technology from the music industry, such as synthesizers and the Musical Instrument Digital Interface (MIDI), has yielded an increase in the quality of the composition of music in

computer entertainment systems, however, there has been little technological advancement in the intelligent control needed to provide automated music composition which changes gracefully and naturally in response to dynamic and unpredictable actions or the 'plot' of the game'' (Land and McConnell 1994, p. 2).

MIDI sequencing had been designed for linear music, or at best very limited looping. Music was intended to be played back sequentially in the order set in the MIDI sequence. There were no ''provisions for branching or conditional messages,'' and nor could ''the playback system be re-configured to respond differently to the performance data'' (Land and McConnell 1994, p. 2). The music could not, in other words, respond intelligently to game events, and the result was, typically, quite jarring transitions between sequences of music in a game (the hard cuts seen in previous game music):

> For example, suppose that there is a high-energy fight scene occurring in the game which, at any time, may end in either victory or defeat. In existing systems there would likely be three music sequences: fight music (looped), victory music, and defeat music. When the fight ends, the fight music would be stopped, and either victory or defeat music would be started. The switch from the fight music to the victory or defeat music occurs without taking into account what is happening in the fight music leading up to the moment of transition. Any musical momentum and flow which had been established is lost, and the switch sounds abrupt and unnatural. (Ibid., p. 5)

iMUSE sought to solve these problems. The iMUSE patent described several components to the software: at its core was a database containing musical sequences. These musical sequences contained *decision points*, or markers, within the tracks. These decision points marked places where changes or branches in the performance could occur based on a condition in the game in real time. When the sound driver reached a decision point, it evaluated the conditions that had been set, and determined the appropriate action. Actions included the ability to turn on or off (or alter the volume of) one or more musical instrument parts, the transposition of musical parts, the ability to change instrument selection, jumping to new parts, looping one or more sequences, delaying the execution of a sequence, detuning an instrument, panning, changing the speed of a sound, and so on. Actions would not take place unless the sound driver encountered a decision point and the correct condition was met. MIDI messages were thus given new sets of conditional controllers by the composer: specifically, hooks and markers, which were given identification numbers with corresponding values in the musical sequence. In hook messages, an action was specified such as ''jump to a new location within the sequence,'' ''transpose this instrument part,'' ''increase the volume,'' and so on. In marker messages, the action was defined by the use of triggers. When a sound played matched the sound specified by the trigger, and a marker having a corresponding identification was encountered, the commands were executed.

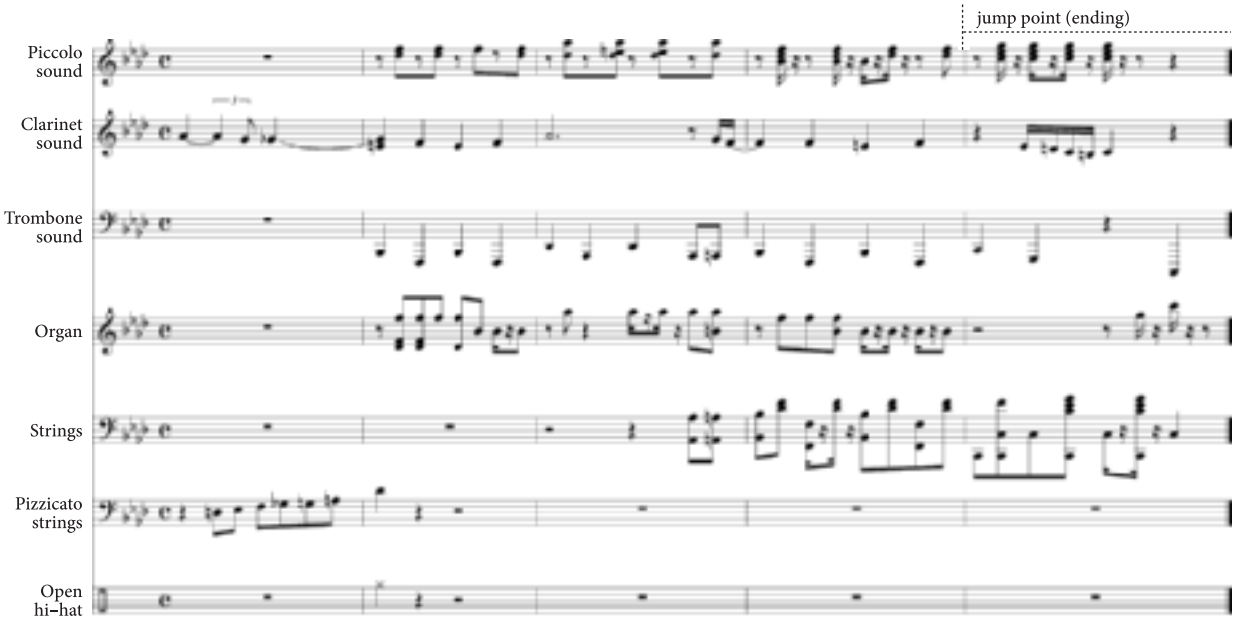


Figure 3.5
Day of the Tentacle (Clint Bajakian, Peter McConnell, and Michael Z. Land, LucasArts, 1993): Nurse Edna's room, showing the jump to the end of the sequence.

For example, in *The Day of the Tentacle*, when characters in the game enter or leave locales, there are changes in music cues. The player's character Bernard walks around the hallway of a hotel to the accompaniment of a hotel theme song. While Bernard walks around the hotel, a long, looped sequence is played. When the player chooses to open the door on Bernard's right, Bernard changes locale and enters Nurse Edna's room. Upon entering the room, the cue must change from that of the general hotel hallway music to Nurse Edna's cue. If Bernard then leaves Nurse Edna's room, a transition point is chosen, and now, instead of transitioning immediately back to the hallway theme, the music jumps to the very end of the Nurse Edna loop, playing the ending as a brief transitional sequence before transitioning to the start of the hallway theme (figure 3.5).

This transitioning from room to room, then, represents two different approaches to musical transitions: the first is a marker trigger that waits until an appropriate point before transitioning to the new sequence. The second uses a jump hook in the sequence and plays another part of the sequence before transitioning. A jump hook is used in a different way in *Sam & Max Hit the Road*. While the two characters (private investigators, one a rabbit, one a dog) stand in front of their office building, the ''street theme'' plays. When the player selected for Sam and Max enter their car, the theme appears to speed up. In actuality, the

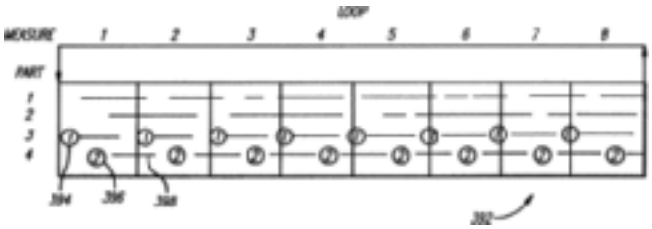


Figure 3.6
Loop sequencing in iMUSE, showing an eight-bar loop with conditional responses (Land and McConnell 1994).

MIDI file has two versions of the theme, one at slow speed, and a second cue for the fast speed. The first jumps to the appropriate stage in the second cue when the theme is to speed up.

Loop parameters could also be set within sequences, which defined start and end points, as well as playback parameters (number of repetitions, and so on). The composer, in this case, composed musical sequences, and then conditionalized those sequences, making decisions about which segments should play continuously, change, be enabled or disabled, loop, converge, branch, jump, and so on.

For example, figure 3.6, from the patent document, shows an eight-bar loop sequence (called ''392''), which consists of fight music, victory music, and defeat music. The first two channels (parts 1 and 2 listed on the left) contain no messages, and therefore continuously loop. Part 3 is played only when a fighter is winning, and part 4 only when the fighter is losing. Parts 3 and 4 contain hook messages (numbered 394 and 396) at musical phrase changes. Musical phrases are designated as number ''398,'' and are sequences that must be played completely. As the fight sequence begins, parts 1 and 2 play and loop. If the fighter begins to lose, hook message 396 is invoked, causing part 4 to begin playing at the next marker point (between phrases, at the ringed number 2s). If the player begins to recover from the fight, 396 is disabled. If the player now begins to win, 394 is enabled, causing the third channel to begin to play at the next marker point. Thus,

> the fight music, rather than playing along unresponsively, can be made to change [the] mood of the game in response to the specific events of the fight. For example, certain instrument parts can signify doing well (for example, a trumpet fanfare when a punch has been successfully landed), while others can signify doing poorly (for example, staccato strings when the fighter has taken a punch).... Also, it may be desirable to transpose ... the music as the fight reaches its climax. This can also be done either immediately under entertainment system control, or by hook message (if it is necessary that the transposition occur only at an appropriate point in the music

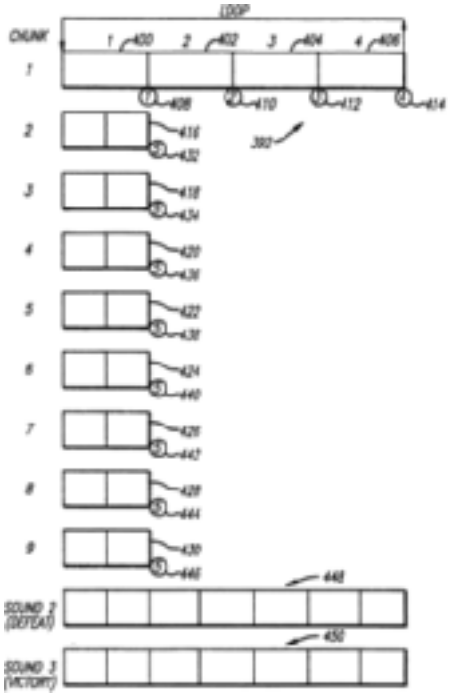


Figure 3.7
"Jump to" transition in iMUSE (Land and McConnell 1994).

> sequence). The resulting fight music will change mood and character along with the intensity and excitement of the fight, but in a smooth and aesthetically natural way, much like the music would follow the action in a feature length motion picture. (Land and McConnell 1994, p. 5)

When the fight is won or lost, the victory or defeat sequence must be cued from the current looped fight music sequence. This process could be accomplished in two ways with iMUSE. In figure 3.7, the top four measures ("Chunk 1," 400–406) represent one continuous looped sequence, each number representing a two-bar segment of the eight-bar sequence in figure 3.7. There are two jump hook messages (408–414) placed at the end of each of these measures. One message (448) represents a transition to defeat, and the other a transition to the victory cue (450). Chunks (sequences) 2 through 9 represent the various options for transitions, to provide the connection between the end of the measure (400–406) to the beginning of the victory or defeat cues. If the player wins the fight, the command to start the victory cue is queued up, and the engine waits to hit the next marker point (at the end of measure 400 to 406) before beginning the appropriate transition cue, followed by the victory cue.

The theme song for the town of Woodtick on Scabb Island from *Monkey Island 2: LeChuck's Revenge* offers an example of such transitions. Woodtick (a series of docks and ships) has a town theme song that loops whenever the character walks around the docks. When the player enters one of the ships, the music transitions to a ''ship theme,'' of which there are several variants depending on which ship the character enters. The main melody of the ship stays the same, but the instrumentation varies. The last two bars of the sequence offer a transition point: if the player has left the ship, the song returns to the town theme. If the player has not left the ship, the variation loops. The song always plays out its loop when returning to the town theme.

In the swamp scene of *Monkey Island 2*, an example of iMUSE's layering of instruments is heard. In this scene, the player's character must walk from the edge of a swamp, climb into a coffin, and row the coffin across two screens to the ''International House of Mojo,'' a voodoo shop in the middle of the swamp. As the player progresses from the edge of the swamp to the House of Mojo, instruments are added to the theme (figure 3.8). Here I have separated the times at which the different instruments are heard (although they exist at all times in the

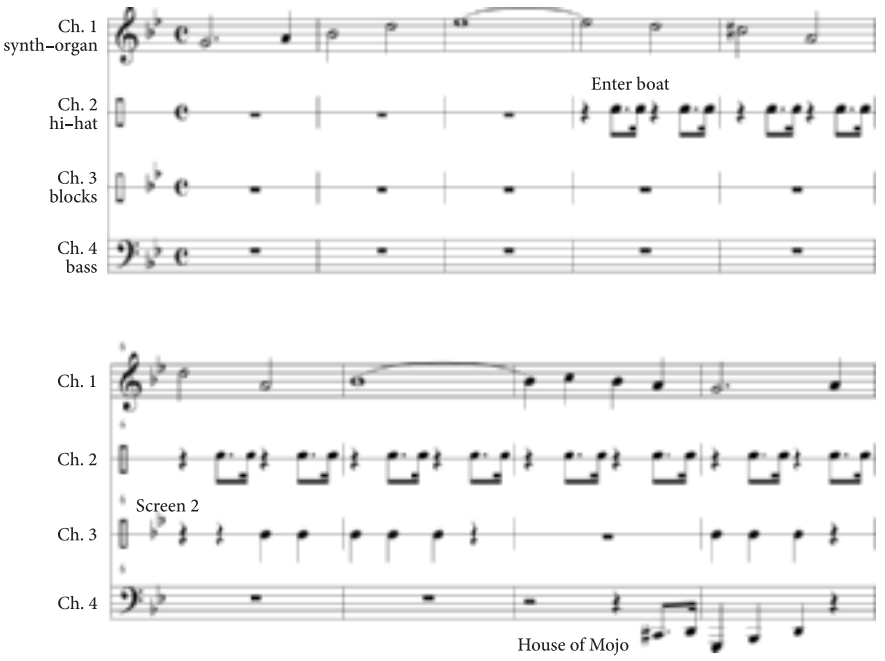


Figure 3.8
Approaching the ''House of Mojo'': *Monkey Island 2: LeChuck's Revenge* (Clint Bajakian, Peter McConnell, and Michael Z. Land, LucasArts, 1991) showing layering of instruments in iMUSE.

MIDI file, whether ''switched on'' or ''switched off''). The first channel plays an eerie synth-lead, somewhere between an organ and a choir sound, while the player is standing at the edge of the swamp. This instrument remains constant. When the player climbs into the coffin, the second channel, an open hi-hat, is added. The player then steers the coffin to the edge of the next screen, whereupon a (very quiet) block sound is heard which seems to be at random intervals. Finally, when the player happens upon the House of Mojo, a bass line begins. Unlike other channels of the song, the bass line always begins at the start of its sequence, so there may be a slight delay before it begins.

At this stage, LucasFilm Games had been in competition with Sierra On-Line, a maker of popular graphic adventures. Recalls Mike Ebert, ''Everyone just wanted to make something better than *Kings Quest* or *Space Quest* from Sierra . . . There was a strong sense of competition against Sierra. I don't know how Sierra felt about us, but in our minds they were the arch enemy'' (cited in Wallis 2007). The SCUMM engine that housed iMUSE was created as a response to Sierra's games, notes creator Ron Gilbert: ''One day I saw *Kings Quest I*, and it was an 'ah-ha!' moment. After a few minutes of playing it, I was totally frustrated by the silly parser. They made this huge leap from text to graphics, but kept the most frustrating part of text adventures. From that frustration, *Maniac Mansion* was born'' (cited in Fish 2005, p. 94). Advances such as iMUSE helped LucasArts to compete in a highly volatile market, but more importantly, drew attention to the possibilities of dynamic audio in games. Many game magazine reviews of the era mentioned the music and the iMUSE engine as selling factors for these games.[25] One reviewer, for instance, notes of *Monkey Island 2*, ''It allows the game's music to respond smoothly and spontaneously to unpredictable player choices, creating at the same time a wonderful feeling of continuity in the music'' (Linkola 1997). iMUSE helped to set a precedent for music to be more responsive to the player's actions, truly distinguishing the music of games from that of linear media.

AMIGA AND THE MOD FORMAT

Although LucasArts had been advancing game music with MIDI-based developments, there were also interesting developments in another format, on the Commodore Amiga. The Amiga had initially been an offshoot of Atari. It was developed by Jay Miner, who had worked on the Atari VCS and created his own company, Amiga, which was taken over by Commodore in 1984. The Amiga computer went through many different models, but despite the many improvements as the models progressed, the sound hardware remained essentially the same. The sound chip in the Amiga was known as Paula, capable of four-channel 8-bit stereo, which could produce fairly complex waveforms over nine octaves (20 Hz to 29 KHz). Sound could be synthesized on the fly or stored as digital samples

and played back through the DAC in stereo. Each channel had its own volume control but no envelope generator, although each channel was capable of modulation, which allowed for tremolo and vibrato effects.

Despite being designed as a games computer, there were initially few utilities available to create sound on the Amiga. Most developers at game companies therefore created in-house programs to sequence music.[26] One such program was created by Karsten Obarski, a programmer and composer at the German reLINE Software game company (see Carlsson 2008 for more on this history). Based loosely on the earlier sequencing techniques of Rob Hubbard's approach on the Commodore 64, Obarski's *Ultimate SoundTracker* sequenced patterns and would export them to assembly language for the composer. The software was released as a commercial product in 1987 and quickly became a standard for games music on the Amiga. It was, however, quite limited, in that it viewed the use of channels in a quite strict fashion, as melody, accompaniment, bass, and lead, and supported only sixteen instruments (samples). Swedish programmers Pex ''Mahoney'' Tufvesson and Anders ''Kaktus'' Berkeman released an update to the software in 1989 known as *NoiseTracker*, which allowed for thirty-two instruments and was more open in terms of channel usage.[27] Later versions of *SoundTracker* used what became known as the module format, or MOD, which included both patterns and instruments in the same file.

Most game music on the Amiga was written in the MOD format. Compositions were created using a software program known as a tracker, and resulting music was stored in MOD. A tracker program would store data on the notes, volume setting, effects, and instrument (like MIDI), but could also include digital samples of the instruments in the actual file, limited only by the size of the file (the 880 kB floppy disk). MOD files had an advantage over MIDI, then, in that music or other sound events would sound as the composer/sound designer intended, and in that more possible sounds were opened up for the authors to use. Not only this, but the ease of sampling meant that much more realistic sound effects could be used. Some games, such as *Jill of the Jungle* (Epic MegaGames, 1992, sound by Dan Froelich) included a ''Noisemaker,'' the fun option of allowing the player to hear the sounds in advance of their in-game action by pressing different keys on the keyboard. MOD files were also easier to program for nonmusicians—like many game programmers—and in particular made it easy to sequence repetitive loops.[28] In fact, the tracker format of using blocks to represent music in a linear fashion became a standard that has been incorporated into the design of today's sequencing software. The MOD format was easily adaptable to game sound, in that patterns (sequences) could be arranged to change volume, jump to other sequences, start or stop instruments, and so on. The ease with which MOD files could be constructed no doubt played an important role in their success. iMUSE may have been very progressive, but it was a proprietary tool, and there seemed neither the capabilities nor the desire to construct such software on other systems

at the time. However, MOD was not without its difficulties. Although MOD files dominated the Amiga games scene, the MOD format never really caught on for gaming in general because of the required file size and the fact that there was no standard. There were nearly twenty different formats, which would allow for different sampling rates and different numbers of tracks. Although there were a few game companies outside the Amiga scene that used a tracker format (Epic MegaGames, for instance), the majority used the better-supported MIDI.

By using the Windows program *Modplug Tracker*, MOD files can be opened and explored, so the user can hear the samples individually and view the sequencing data. Using *Modplug* to open one of the files from the *Shadow of the Beast II* game, discussed above on the Genesis, we can see how the tracker format lays out the song (figure 3.9). The white horizontal squares listing the numbers ''8,'' ''9,'' ''10,'' etc. are the patterns (sequences) that make up this particular song. This song plays pattern 8, followed by pattern 9, pattern 10 twice, pattern 11, and pattern 12 twice, before looping. Pattern 10 is open in figure 3.9, showing the 64 notes in the pattern (each pattern in this song has 64 beats). Each row is played in sequential order following the 64 numbers in the leftmost column.

The four channels of available sound are shown in columns. The channel information shows the note to be played based on the sample referred to. Samples were transposed and all set to C5 originally, and these were then transposed to the notes indicated. The second number in each column tells the tracker which sample to use for the note. There are only four sounds in this sequence: Channel one and two both contain sample #07, a thin electric guitar–like sound (called ''g1 lead2''), Channel three has sample #08, a fat bass, and Channel four has sample #01, pitched percussion.

Deconstructing MOD files in this fashion shows the many different approaches to song construction in Amiga games. Typically far more complex than the songs of the 8-bit games, songs in Amiga games had a highly varied structure, but usually followed the same basic principles seen in the 8-bit games (shorter loops with fewer distinct sections for battle music, and longer loops with more distinct sections for in-game music). Some tracks, such as the in-game music of *Jurassic Park* (Ocean, 1993), had over forty different patterns.

CONCLUSION

The 16-bit era marked a time when game audio was no longer viewed as an afterthought, but was, rather, the turning point of many interesting developments in the history of games. Many of the technological audio innovations were now coming from within the games studios and were pushed by the game composers, sound designers, and sound programmers, rather than coming from outside sources. The competitive nature of the games industry, however, meant that

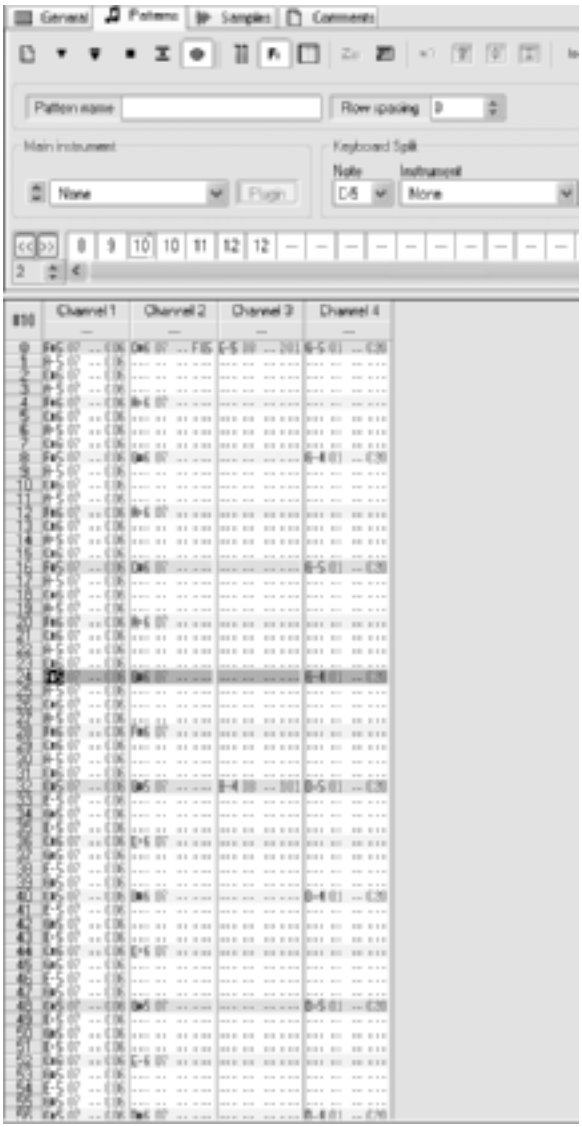


FIGURE 3.9
Shadow of the Beast II (Tim Wright, Psygnosis, 1992) tracker MOD details.

many of the most important advances—such as iMUSE—remained proprietary, in the hands of those who created and patented the ideas. Getting an edge over a competitor was more important than creating standards by which the industry could develop. It is perhaps ironic that iMUSE would work on the back of MIDI, which itself had been a rare example of industry collaboration. As Théberge (1997, p. 85) notes, ''the 'innovative' aspect of MIDI was that it was planned as a non-proprietary device (no one owns a patent for either the hardware or the software portions of MIDI).'' The proprietary nature of software and hardware developments led to the creation of competing standards, a problem that still exists today. The nature of rapid technological change pushed games toward advancing as hastily as possible, but in a secretive environment. As Denis Dyack, founder of games company Silicon Knights noted recently, ''technology would change so rapidly, if you didn't get your game out as soon as possible, within six months, id [software games company] could come out with a new engine and make you look like crap and you'd be dated. So there's this technology curve that people were just completely intimidated by'' (cited in Brightman 2007). There were, then, increasing tensions between getting a game out as quickly as possible, and trying to innovate to get an edge.

The development of the General MIDI standard and advancements in sound hardware enabled improvements in sound quality and polyphony, as well as increased memory, allowing for more music and longer songs.[29] Higher-fidelity sampling allowed for more realistic sound effects, and the MOD format brought ease of organization for the countless sequences that were involved in a game, while at the same time it helped to reduce memory requirements for the music, by basing songs on sequencing data. More importantly, it represented a change in game software toward the creation of musician-friendly products. Whereas previously most creators of game music had been programmers (as shown in chapter 2), the creation of MIDI and MOD enabled the influx of more musicians into game development. As a result, sound became a more integral part of games in this era, in such a way that turning off sound was sometimes detrimental to gameplay, as was seen in *Toejam and Earl: Panic on Funkotron*. LucasArts' *Loom* was also such a game, reliant on the player memorizing key sequences of music. Of course, sound effects had always played a useful anticipatory role in game audio, but now music had begun to take on some of this role. Although there were examples of explorations in dynamic audio not discussed (notably Origin Systems' *Wing Commander* [1990], composed by George Sanger, in which music responded to gameplay states), nowhere is this trait as well illustrated in the 16-bit era than in LucasArts' iMUSE games. As shown, the iMUSE design recognized that game music had to be different from standard linear music, and that an adaptable, dynamic score would greatly enhance gameplay. Transitions between spaces in gameplay could now be smoother, with the music matching and responding to what was occurring in the game. The blueprint had now been set for the directions in which game audio was to evolve.