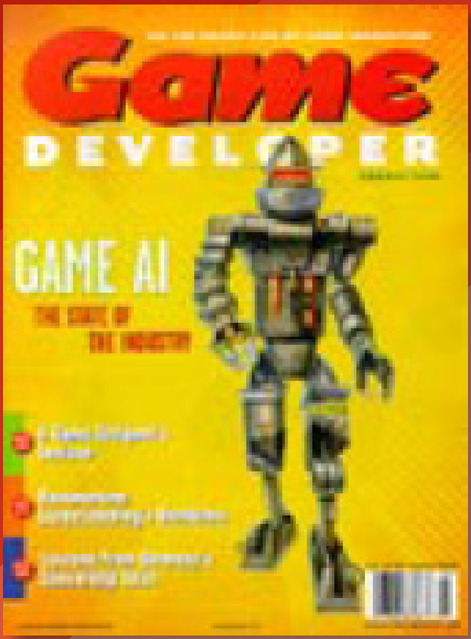
GAME DEVELOPER MAGAZINE

AUGUST 1999



Back

Where's Webster When You Need Him?

There's an old story (I chalk it up as an urban legend) about a young FBI employee who was put in charge of the bureau's supply department. He figured that an efficient way to cut costs would be to reduce the size of the memo paper used throughout the agency. When one of the memo sheets found its way onto J. Edgar Hoover's desk, he got upset over the change and wrote on the narrow margin, "Watch the borders." For the next six weeks, it was difficult to enter the country by road from either Canada or Mexico.

This tale is about as bogus as they come, but it illustrates what can happen when communication gets garbled. Efficient communication is tough. That's why so many Postmortem authors in *Game Developer* have cited "bad communication between team members" as a major factor that hindered their game projects. And perhaps no aspect of game development is as loosely defined and poorly documented as game design.

As game players, we often take for granted what makes a game "rock" or "suck," without digging deeper and analyzing what that means and what exactly evoked such a response. That's the luxury of the player. As developers though, we're expected to be masters of the art; to see many layers beneath the surface. We have to be familiar with the devices that make games challenging, fun, addictive, exciting, funny, scary.

Having a formalized vocabulary for game design elements and techniques would help the industry greatly. Other forms of media have their own lexicons — novelists, script writers, movie directors, and composers converse using well understood and accepted terminology. (A quick search on Amazon.com reveals dozens of dictionaries for filmmakers, fiction writers, poets, and other creative professionals.) With decades (if not centuries) of history behind these fields, perhaps this comes as no surprise. Now that we're a multi-billion dollar industry though, we should begin to designate formal terms for our craft, too.

This month we present an article on page 44 by Looking Glass's Doug Church which addresses the need for a game design vocabulary. The article takes the first steps towards the creation of a lexicon for game designers — a collection of "formal abstract design tools," as Church calls it — which can be applied to game designs across various genres and platforms.

There will be some who say, "Bah — what a bunch of pointy-headed nonsense. Don't turn game design into an academic exercise and start jargonizing things." That's definitely not the aim here — jargon for jargon's sake does no good. But try imagining what filmmaking would be like if every time the director wanted a zoom-in dolly-back shot, he had to say, "Roll the camera away from the actor while you simultaneously increase the lens magnification on him, just like Hitchcock did in that neat scene in *Vertigo*!" Without an effective vocabulary to discuss a concept, it can get pretty difficult to communicate your intentions and ideas.

Here's Where You Come In

If you read Church's article and want to submit some terminology of your own, our sister web site Gamasutra.com is launching an online lexicon to extend the concepts that Church presents. We invite you to submit terms and view what others have proposed. With a little inspiration from Noah Webster, we may all be able to take the art of game design to the next level. ■

[illegible]

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.2228 w: www.gdmag.com

Publisher
Cynthia A. Blair *cblair@mfi.com*

EDITORIAL

Editorial Director
Alex Dunne *adunne@sirius.com*

Managing Editor
Kimberley Van Hooser *kvanhoos@sirius.com*

Departments Editor
Jennifer Olsen *jolsen@sirius.com*

Art Director
Laura Pool *lpool@mfi.com*

Editor-At-Large
Chris Hecker *checker@d6.com*

Contributing Editors
Jeff Lander *jeffl@darwin3d.com*
Mel Guymon *mel@surreal.com*
Omid Rahmat *omid@compuserve.com*

Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook Verant Interactive
Susan Lee-Merrow Lucas Learning
Mark Miller Harmonix
Paul Steed id Software
Dan Teven Teven Consulting
Rob Wyatt DreamWorks Interactive

ADVERTISING SALES

Western Regional Sales Manager
Jennifer Orvik e: *jorvik@mfi.com* t: 415.905.2156

Eastern Regional Sales Manager/Recruitment
Ayrien Houchin e: *ahouchin@mfi.com* t: 415.905.2788

International Sales Representative
Breakout Marketing e: *breakout_mktg@compuserve.com*
t: +49 431 801703 f:+49 431 801797

ADVERTISING PRODUCTION

Senior Vice President/Production Andrew A. Mickus
Advertising Production Coordinator Dave Perrotti
Reprints Stella Valdez t: 916.983.6971

MILLER FREEMAN GAME GROUP MARKETING

Group Marketing Manager Gabe Zichermann
MarComm Manager Susan McDonald
Marketing Coordinator Izora Garcia de Lillard

CIRCULATION

Vice President/Circulation Jerry M. Okabe
Assistant Circulation Director Sara DeCarlo
Circulation Manager Stephanie Blake
Assistant Circulation Manager Craig Diamantine
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Joyce Gorsuch

INTERNATIONAL LICENSING INFORMATION

Robert J. Abramson and Associates Inc.
t: 914.723.4700 f: 914.723.4722
e: *abramson@prodigy.com*

Miller Freeman
A United News & Media publication

CEO/Miller Freeman Worldwide Tony Tillin
Chairman/Miller Freeman Inc. Marshall W. Freeman
President & CEO Donald A. Pazour
Executive Vice Presidents Darrell Denny, Galen A. Poss, Regina Starr Ridley
Sr. Vice Presidents Annie Feldman, Howard I. Hauben, Wini D. Ragus, John Pearson, Andrew A. Mickus
Sr. Vice President/Development Solutions Group KoAnn Vikören
Group President/Division SF1 Regina Ridley

BPA INTERNATIONAL

www.gdmag.com

More Than Just a Steady Paycheck to Some

Chris Hecker's Soapbox column ("So You Want to Make Movies? Good Riddance!" June 1999) echoed my long held belief about the status of the game development industry. Easily half of the artists I have worked with in the industry would have preferred to be doing something else. Whether it be film or traditional art, game development is just a steady paycheck to them while they "get their demo reel ready."

More than once I have had to explain to other artists in the field why I don't want to work in film. Many of them don't understand that my goal is to bring games to the level of film and television and finally to eclipse those media with interactivity.

This attitude is less prevalent among programmers. Still, you find more than a small share of programmers who are enamored of the technological aspects of game development: how many polygons can they push, or how much physics can they code. Technology can't help but push the games further, but programmers shouldn't sacrifice the game itself in the name of technology.

Even the game designers might rather be writers working on mainstream fiction instead of writing up design documents and honing the game balance.

The real key to achieving the game industry's equivalent of a *Casablanca* will be creating a game world that is as accessible as a film or novel, and is just as engaging. Such a game needs the industry's digital equivalent of Bogart and Bergman, the writing nuances of the Epsteins, and the directorial talents of a Michael Curtiz.

I am eagerly awaiting Dramaera's THE INSIDER (http://www.dramaera.com) to see if they can pull off real emotion in a computer game.

Carl L. Pinder
New World Computing,
a division of The 3DO Company
via e-mail

I found Chris Hecker's recent Soapbox on developers with ulterior aspirations interesting. I agree completely that complaining is annoying. I also would never openly talk of working for another company while under the employment of another (it's just bad manners).

However, as a game artist and a lover of independent film, I must disagree with many of his points. First and foremost, I don't ever think that a game will ever be revered as a work as great and as artistically poignant as say, films such as *Saving Private Ryan*, *Apocalypse Now*, *Forrest Gump*, *Schindler's List*, the *Godfather* movies, *Citizen Kane*, and *The Maltese Falcon*.

Don't keep it bottled up inside. E-mail saysyou@gdmag.com, or write to *Game Developer*, 600 Harrison Street, San Francisco, CA 94107.

Conversely, we already have many games that are a far superior experience to films such as *Starship Troopers* and *Independence Day*, both of which suffered from over-emphasis on special effects, while leaving the acting and script as a secondary consideration. A good example of this would be an intensive game like UNREAL. That game's story line and rich background do more for your sense of wonder and imagination than these poetically devoid big-screen bombs achieved. Second, if Hecker only refers to games as a medium of interactivity, then he needs to open his eyes sometime at Siggraph. Games are not the end-all final destination for interactivity. Interactive fine art that senses the viewer and plays on body location is happening right now. He obviously needs to get out more often.

The bottom line is, I think his take on games as a higher art form is comical at best. How serious can we take an industry whose best seller stars a heroine with anti-gravity boobs, and little fan boys keep pictures of her that they got from a few "galleries"? Comic books have also had their war cry to be recognized as a higher art form. You almost can't find a comic today without some ridiculously proportioned woman drawn in it.

Technology will bring us to a melding of cinema and interactivity soon. Why would you condemn those people who would leave the strict game format for broader fields?

Perhaps the game industry can make a classic game that everyone knows and loves like, say, the board game classic Monopoly. And to a point that has happened a few times already, namely with AGE OF EMPIRES and QUAKE. But to think that a PC game will ever be held in the same light as a great movie is little more than a delusion of grandeur.

Tony Castelluci
via e-mail

AUTHOR CHRIS HECKER RESPONDS. *My short response to you is this: I'm taking bets. Cash money. Ten thousand dollars due in 50 years — you pay me if games have made it into the big leagues as an art form. If they haven't, I pay you. What do you say?*

My longer and less flippant response is that for every potentially new art form, there are proponents and detractors. Time will tell who's right, of course, but I think games are qualitatively different from comic books in that they are not the hybrid of two existing art forms. While comic books are a blend between written language and pictorial art, games are not "in between" any two existing art forms — they expand the envelope of art via interactivity.

As for Siggraph-style "interactive fine art," I think that's a totally valid use of interactivity, and you have a good point. I do think these works will relate to games in the same way experimental art films relate to more mainstream films. That is, these interactive fine art pieces will have a smaller audience and get less attention than mainstream games and experiences.

As a friend pointed out to me, the term "games" might be a limiting factor in and of itself. Maybe we should all work towards figuring out a better name that gives our industry a little more room in which to grow. Regardless of the name, however, I'll bet the medium will ultimately find its rightful place alongside movies, painting, music, and the like. Are there any takers out there?

Back

BIT Blasts

News from the World of Game Development

New Products

by Jennifer Olsen

Gaming All the Way to the Bank

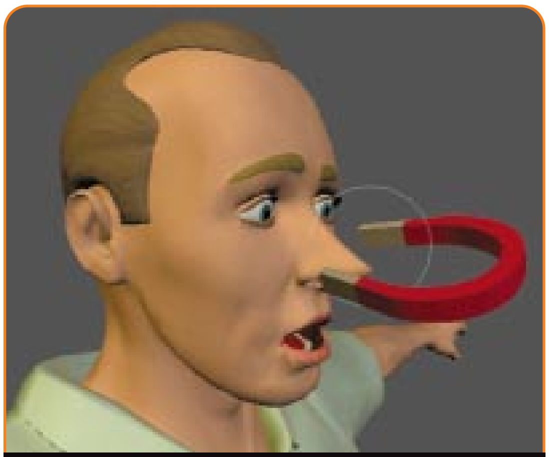
ERICSSON HEWLETT-PACKARD TELECOMMUNICATIONS (EHPT) has developed Jalda, a non-proprietary system for handling Internet payment transactions, which EHPT hopes will enable game developers to find innovative ways to bring in (read: milk) revenue from their online games. In case you haven't heard, online gaming has quickly secured its place as one of the fastest growing slices of that big, juicy e-commerce pie, with total revenues projected to rise tenfold to almost $700 million in 2003.

Jalda is EHPT's response to demand from both consumers and content providers for simple and secure Internet payment systems, enabling transactions spanning from collecting micro-payments to awarding tournament prize money into the right player's account. Developers and users can structure different pay-for-play modes, such as per-click, per-minute, per-game, on up to huge multiplayer tournaments. (Just imagine each of those pay-per-clicks as change jingling in your pocket.)

Jalda's developers tout its security, which uses RSA Public Key Infrastructure, a cryptographic technology that claims to eliminate fraud using digitally signed transactions whereby each party in the transaction has his own electronic ID. Better still, when someone pirates your game, Jalda's soft certificate goes with it, and recipients of pirated games start paying when they start playing.

Jalda's API is available for download from the Jalda web site.

■ Ericsson Hewlett-Packard Telecommunications AB
Stockholm, Sweden
+46 (8) 685-2000
http://www.jalda.com/eips


Manipulating faces is simple with Poser 4's magnetic deformers, but it sure looks painful.

Strike a Pose

METACREATIONS has introduced Poser 4, its next-generation figure posing and animation tool, hoping to provide an affordable package for newcomers to 3D animation and a specialized pre-visualization tool for high-end developers.

The upgrade introduces features such as a redesigned user interface, new animal figures, a swappable 3D clothes wardrobe, and new morph targets for generating characters. Advanced texture controls allow users to incorporate transparency and reflectivity into a scene, apply a texture style to an entire figure or body part, and create effects such as wispy hair using transparency maps.

Poser also comes with an improved library of 3D figures, while new magnetic and wave deformers allow users to sculpt 3D figures by manipulating body and facial surfaces. Poser 4 also supports multiple deformers for customizing faces by allowing photos to be applied as texture maps. Users can generate new figures within Poser 4 by importing 3D geometry, breaking it into body parts, and then generating the new figures using inverse kinematics.

Poser 4 has a suggested retail price of $249 and upgrade discounts are available.

■ Metacreations Corp.
Carpinteria, Calif.
(805) 566-6200
http://www.metacreations.com

New Toy Box for Digital Artists

RIGHT HEMISPHERE has released Deep Paint 3D, which, despite the potential for confusion over sequential numbering, is the successor to 4D Paint, its Windows-based 3D paint program.

Deep Paint's tool set has been redesigned to include features resembling airbrush, oil, watercolor, colored and charcoal pencils, felt pens, chalks, pastels, gouache, acrylics, impasto, and texture and image spray paints, among others. The addition of Phong rendering allows for improved 3D rendering quality and speed by enabling users to see in real time the variations in paint texture shininess.

With support for .3DS and .LWO file formats, Deep Paint integrates easily with other modeling systems, including 3D Studio Max, Softimage, Lightwave, and Maya. It also supports Photoshop plug-ins and a two-way interface with Photoshop.

Deep Paint 3D runs on Windows 95/98/NT and Intel or Alpha processors. The Intel version of Deep Paint 3D has a suggested price of $795, the Alpha processor version is $995.

■ Right Hemisphere Ltd.
Auckland, New Zealand
+64 (9) 309-3204
http://www.righthemisphere.com

Industry Watch

by Alex Dunne

3DFX SUES CREATIVE. In the aftermath of the divorce between 3dfx and its former board-making customers, some legal stuff has hit the fan. 3dfx filed suit against Creative Labs and its parent company, Creative Technology, claiming that Creative breached a licensing agreement and infringed on 3dfx copyrights by incorporating 3dfx Glide source code into Unified, Creative's new technology for running Glide-only games on Creative's TNT and TNT2-based cards. (Unified consists of a software layer that translates Glide calls into the corresponding commands in Direct3D, plus extensions that support Creative's Graphics Blaster RIVA TNT.) If Creative did use 3dfx code for this purpose, that would be a no-no — the Glide license agreement stipulates that licensees can't use or modify 3dfx source code so that it operates with non-3dfx hardware. 3dfx also asserted a claim against Creative Technology for unpaid amounts owed for 3dfx products.

MS EMBRACES EAX. Microsoft announced a licensing agreement with Creative Technology for a number of Creative's EAX audio effects. The licensed effects, including flange, chorus, EQ and environmental reverberation, will be incorporated by Microsoft into the next version of DirectX. While Creative officials admitted that the licensing agreement with Microsoft "has no direct financial impact on Creative," it looks to be a strategic win for Creative, which has been trying to make EAX an industry standard for 3D audio effects, thereby increasing sales of Creative audio hardware.

Interplay will get North American distribution rights to Monolith's Odium.

EIDOS HAS A GOOD YEAR. Eidos announced results for the year ending March 31, 1999, and it had some nice news for investors. The company saw revenues increase by 65 percent to over $364.3 million for the full year, resulting in a pre-tax profit of $50.3 million (compared to $0.16 million in 1998). Ian Livingstone, Eidos's chairman, cited the continued successful development of existing franchise properties such as Tomb Raider, Gex, Championship Manager, and Thief: The Dark Project for the company's successful year. Nineteen new titles were launched by Eidos during the year, and eight games (including catalogue titles) achieved sales in excess of 350,000 units.

SAVING UP FOR PSX2. While consumers "ooh" and "ahh" over the stunning graphics and processing power of Sony's recently-announced PlayStation 2, game developers may secretly be wincing at what looks to be an expensive platform for which to develop. Some companies are starting the bankroll-building process. Case in point: 3DO just announced plans to sell $34.5 million in new shares, which will be used in part to fund its PSX2 games. 3DO plans to release at least six titles for the new PlayStation around the time the new console ships.

MONOLITH FARMS OUT DISTRIBUTION. Apparently looking to focus more of its resources on game development rather than distribution, Monolith entered an exclusive agreement with Interplay which gives the latter North American distribution rights to three of Monolith's upcoming RPGs, Rage of Mages II: Necromancer, Septerra Core and Odium. All three titles are scheduled to ship for the PC this fall.

GAME RATINGS REDUX. In the wake of recent tragedies like the Columbine shooting and the subsequent debate that has ensued over the media's role in adolescent violence, Senators Joseph Lieberman (D–Conn.) and John McCain (R–Ariz.) introduced legislation to create a uniform media rating system. Under the proposed legislation, entertainment media industries would have six months to develop an across-the-board rating system for videogames, television and music. The warning labels would have to reflect the nature, context, intensity of violent content, and age appropriateness of the media product, Sen. Lieberman's office said. It's not clear to what extent the RSAC or ESRB ratings currently used by game publishers will influence such a system. In related news, Rep. Henry Hyde (R–Ill.) proposed legislation that would restrict youth access to videogames, movies and other media containing violent and sexually explicit material.

BRIAN HOOK left id Software for Verant Interactive, developer of the successful online game EverQuest.

CD-WRONG? Looking to divest itself entirely from traditional CD-ROM games and devote itself towards its "high-growth Internet entertainment community" business, Interactive Magic sold its entire CD-ROM product line to Ubi Soft. Interactive Magic will retain the online rights to these CD-ROM products, and operate the Internet versions of the products exclusively on the company's games sites, including iMagic Entertainment Network, GameHub, and MPG-Net. ■

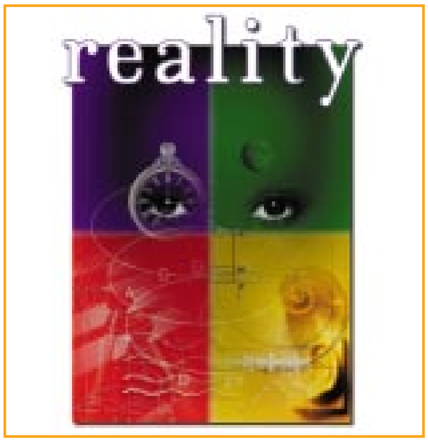
UPCOMING EVENTS CALENDAR

Siggraph '99

LOS ANGELES CONVENTION CENTER
Los Angeles, Calif.
August 8–13, 1999
Cost: $25–$760
http://www.siggraph.org/s99

ECTS '99

OLYMPIA CONFERENCE CENTRE
London, England
September 5–7, 1999
Cost: variable
http://www.ects.com



Seer Systems' Reality 1.5

by Andrew Boyd

The experience of receiving a package containing "Reality" is itself almost worth the purchase price. True, in this case it refers to a Windows 98, host-based software synthesizer of that name, but there's no reason anyone else needs to know that. As far as your programmers, producers and management need to know, the audio department has just received a box of reality — and the philosophical ramifications are staggering.

Reality (the product, not the epistemological construct) seems like a promising solution to some common problems in game sound and music production. Specifically, when designing effects there never seems to be enough unique sources, ways to transform sounds, or tools to let you really get at creative new sounds. And when scoring music using synths and samplers (which most of us in games do), there's never enough polyphony — more modules are always better. Reality offers up a massively customizable synthesizer and sample playback engine that can address these concerns in a single product, and do it inexpensively, leveraging equipment you probably already own.

Reality is a 128-voice, 16-part multitimbral, fully editable synthesizer realized wholly in software. It employs just about every synthesis method you've ever heard of, and a few you might not have. It supports sample playback (with full multi-sample and layering functionality, and full compatibility with SoundFont 2.0), four-operator FM synthesis (using anything for the operator waveforms), subtractive "analog" style synthesis, and 20 different types of physical modeling algorithms such as Simple Mallet, Plucked, Modal, and others. It has a sophisticated modulation matrix, programmable envelope generators, powerful resonant filters (2-pole to 16-pole), and an enormous capacity for patch data.

Much like other synths, Reality organizes its sounds into banks (called "Banksets"), which contain patches called "Programs," which can be a patch or a combination of regions referencing one or more patches. Banksets can also store MIDI files, which can be played back inside Reality's simple sequence player. Reality can either be used as a stand-alone MIDI sound module (as I used it), or it can integrate directly into a Windows sequencer such as Cakewalk and provide a fairly seamless production environment. Reality can output audio using any Windows sound card, though cards with DirectSound drivers will exhibit vastly more acceptable latency behavior than cards without, and Reality can also capture its output into a .WAV file.

Interface

Launching Reality brings up a main workspace in which everything in the program takes place. There are three primary modes of operation: Bankset, Program, and Options. The first two are pretty self-explanatory, the last is where MIDI channel assignments, global adjustments, and effects editing take place. The look of Reality's interface is pretty standard Windows 9x fare, almost to a fault. Frankly, it looks rather like a semi-finished demo application. Its buttons, tabs, dialog boxes, and drop-down lists have little in the way of specialized graphics to ease the feeling that programmers — not musicians or artists — designed it, whether that's true or not. I ran Reality on a desktop set at 1024×768, and some pages use more than the available screen space, requiring awkward scroll bars, while others use far less, resulting in odd-shaped expanses of Windows gray. On the other hand, there are some handy touches: a browser-like pair of forward and backward buttons let you bounce quickly between frequently used screens; global volume, reverb, and chorus level adjustments are available on floating toolbars; and meters measuring left and right output levels and instantaneous CPU usage appear automatically, too.

Although it isn't the most attractive program to grace a display, Reality's interface does an admirable job of translating the program's astonishingly complex synthesis engine into language, components, and structure that users can comprehend. It doesn't try to hide the complexity with presets or black-box controls, but rather presents a window to it using a logical set of structures, which are split up with pages and settings. The hierarchy is straightforward and it leads the user through the process of designing a sound in a fairly sensible, if awfully dry, manner.

For instance, within the Program editing environment, you get a pull-down menu to select the algorithm type, and each algorithm has a tab-selectable Parameters page, which shows only those settings relevant to that algorithm. Also available are the tabs leading to the low frequency oscillator and Envelope pages that are common to all programs. The various connections involved in the modulation matrix are not always as obvious as they could be — some sort of graphical representation beyond just tabbed pages with drop-down lists would be very useful (The "cords" feature in E-Mu's Emulator samplers comes to mind as a great solution to this problem). But for the most part, the structure of a sound is laid out in a straightforward manner. It's essentially a good, highly-specialized editor/librarian application talking to a really sophisticated synthesizer.

In Use

I installed and tested Reality on a 333MHz Pentium II running Windows

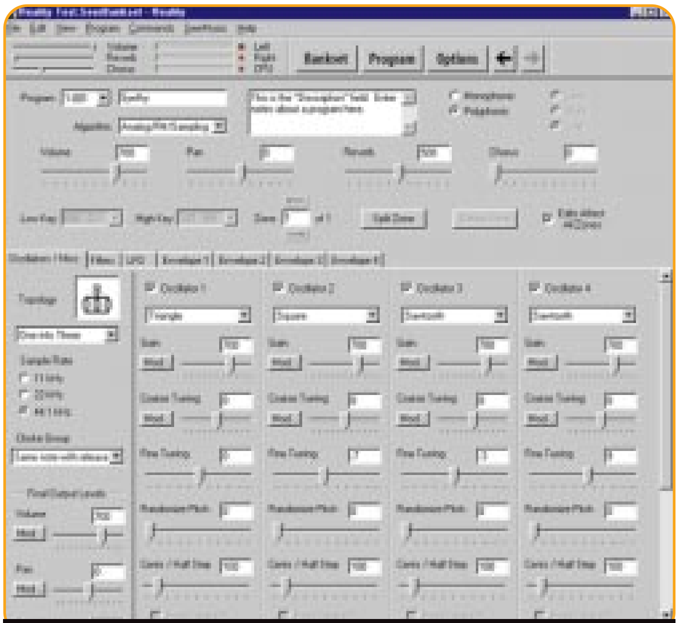
Andrew Boyd is Sound Design Manager for Stormfront Studios in San Rafael, Calif. He's been making sounds and music for computer games professionally since 1993. Drop him a line at aboyd@stormfront.com.

98, with 128MB RAM and Ultra Wide SCSI, and with Sound Blaster Live! and Digidesign Audiomedia III sound cards. I also had Cakewalk Pro Audio 8.1 installed on this machine, and Reality integrates nicely with it, but my primary use for Reality was as a MIDI module controlled through the Sound Blaster card's MIDI input. This was connected to a Macintosh running MOTU's Digital Performer 2.5, through a MOTU MIDI Timepiece AV, and using an Alesis QS-7 as a controller.

The installation was pretty smooth. I already had DirectX 6.1 installed, but Reality will, presumably, install the appropriate DirectSound drivers if necessary. Reality detected the hardware fine, asked about associating file types, offered options for installing extra Banksets and SeerMusic (Seer's web-based audio playback), and so on. Strangely, though, the program didn't work right until I rebooted, even though the setup neither requires nor suggests this. I tried two other machines with the same results. I suppose it's simply good practice to reboot after an installation, but there could have been a note to this effect somewhere. Other than this small issue, I found the program to be very stable in use, without any crashes throughout the testing.

Of course, it's all about sound, and Reality sounds pretty darn good. Steer clear of the demo songs, unless cheesy fusion or pseudo-ska really get you excited. Reality can sound much better than these demos would imply. Some of the included Banksets are quite usable, such as Drums and Bass, which has a great velocity-layered drum kit, and Woodwinds and Brass, which has fairly convincing instruments with great, playable modulation parameters (aftertouch is routed to "blat" on the trumpets, which is really fun). But more so than with sampled or even physically-modeled realistic sounds, Reality excels in non-realistic, "analog" and FM sounds and instruments. The stock banks Amoeba and Electron show these off nicely, with some fat basses and great, shimmering pads.

The analog output was more than acceptable for most purposes coming straight off the Sound Blaster card, and excellent from the Audiomedia (both are unbalanced, but hardly noisier than some of my professional equipment). I often took advantage of Reality's ability to capture its output directly to a .WAV file, and bypassed the sound cards entirely. The latency inherent in the Audiomedia's WaveOut drivers made playing the synth impractical (not completely impossible, but not worth the effort), but the Sound Blaster exhibited as little perceived latency as anything else in my MIDI rig. Reality's direct .WAV output dispatches a long-time complaint about software synths.


Reality's user interface won't win any beauty contests, but it gets the job done.

A great use of Reality is for pure sound design, such as non-instrumental sounds or sounds not meant for playing in a musical context. In the same way that I will often fire up my trusty old Prophet-600 to get some wacky source material for unusual effects, Reality proved useful as a petri dish for growing some really odd experiments in sound combinations. Parameter changes take effect in real time, which is great for tweaking, and there are so many available combinations of parameters and routings that surprises are always right around the corner. What if I take this source sound and stick it into all four oscillators using different tunings and envelopes on each of them, then combine them using different FM operator topologies? It's fascinating stuff.

Perhaps Reality's biggest weakness lies in its built-in effects. It would seem that, as a Windows audio application, it ought to be able to take advantage of the same DirectX audio plug-ins that have become standard, expected, even necessary on other similar applications (Sound Forge, Cakewalk, and Acid, for example). But apparently the architecture of these effects would cause unacceptable latency in Reality, and so they're not supported. While I understand these limitations, it would be nice if Reality at least offered a work-around — report to me the latency and let me program around it in my sequencer or something. But right now, every modern hardware synth runs rings around Reality in terms of available effects, and this is a bit frustrating. It does have built-in reverb and chorus effects, and they offer some amount of programmability. They even sound pretty good, for all that. They're useful, but ultimately disappointing because the rest of Reality's sound generation is so impressive.

For musicians and composers, Reality will be most useful to those who want that analog synthesizer sound without dealing with the irritating vagaries of such beasts. Reality never drifts out of tune or has a scratchy pot or broken key (features my beloved Prophet currently exhibits), for instance. Programmed correctly, Reality's sound can be remarkably fat and satisfying, with solid filters and a lot of motion and articulation available in a voice. Its polyphony, multitimbrality, and programmability go far beyond any analog synthesizer, too. And of course, Reality costs around a third of what you'd pay for that functionality alone in a module.

For sound designers, I think Reality is a great tool to have in the shed when it comes to making otherworldly, fantasy sounds. It's such a quick way to work, because its amazingly complex architecture is laid out so plainly

★★★★★	★★★★	★★★	★★	★
Excellent	Very Good	Average	Below Average	Poor

Seer Systems' Reality 1.5: ★★★★

Seer Systems Inc.	Pros:	Cons:
Portola Valley, Calif. (888) 232–7337 http://www.seersystems.com **Price:** $495 **System Requirements:** Windows 95/98, 133MHz processor (200MHz recommended), 32MB RAM	**1.** Astoundingly flexible, usable synthesizer engine **2.** Great sound quality. **3.** With the right sound card/driver combination, it's as playable as any MIDI instrument.	**1.** Unattractive interface has a "first pass" look to it. **2.** Built-in effects are not up to the level of hardware synths. **3.** Without DirectSound drivers, latency is too high for real-time playing.

before you. And since it uses conventions and terminology from synthesizer programming, it will probably already feel familiar and easily navigable to new users. Monster growls, laser guns, spaceship engines, transporter portals, and robot voices are so much fun to make with this thing, it's almost cheating. And since it can capture its output directly to a .WAV file, it integrates simply with audio editors and multi-track environments.

Reality's manual is clear, complete, and written in a friendly, accessible style. One of the challenges for the documentation to an application like Reality is that it needs to be helpful to users encompassing a wide range of experience. Some users are computer mavens who have never seen a synthesizer before, while others make their living programming synths. But even the latter group may have never worked with a host-based software synth, or with a physical-modeling synth, and few will have worked with a single synth that does all of this at once. Given this range, the manual does a great job providing all the necessary information and some helpful hints, too, without being condescending or overly dry.

The online help is, as is often the case with Windows products, excellent, relevant, well indexed, and much handier than you ever think it will be. Use it. It's your friend.

Conclusion

Reality is an awfully impressive product. It's one of those fun products with which you can get started quickly, but whose depths you'll probably never fully plumb. It's also one of those rare products that will fit as well within the context of a beginner's home studio as it will in the rig of a busy, experienced composer or sound designer. It isn't perfect — the overall look and interface could stand a little polishing, if for no other reason than the psychological effect this makeover would produce; the modulation routings could be a bit more intuitive; and the effects section needs real attention. But Reality lives up to its promise, and in addition to great sound quality and useful functionality, it's quick and stable. And for its price, I haven't seen anything that can touch it. ■

The Trials and Tribulations of Tribology

I have decided that friction is a drag. It's almost as easy to understand as gravity. We deal with it every day. Friction keeps me from sliding completely under my desk when I slouch in my chair. It keeps my car from spinning out of control as I turn corners with reckless abandon.

This experience with friction begins when as babies we attempt to scoot across the floor and find the carpet difficult and the linoleum floor relatively easy. We build upon our experience until as elementary-age children we are able to pick up our video console controller and expertly proclaim, "This game looks so fake — the cars are sliding all over the place. The physics in this game bites!"

That is the challenge game developers face. The physical world is so familiar to everyone in your potential audience, any departure from realism can be glaring. However, realistically simulating these simple physical properties is quite challenging. This month, I'm going to discuss simulation of friction in real-time 3D applications, otherwise known as the field of tribology.

F	Force applied to system
f	Force of friction
N	Force normal to surface
W	Force of weight of object due to gravity
μ_s	Coefficient of static friction
μ_k	Coefficient of kinetic friction
w	Width measurement
h	Height measurement
A	A point of reference
v	Velocity of particle (vector)
ε	Threshold of transition from static to kinetic friction

TABLE 1. *A summary of notations used in this article.*

Why Is It Such A Drag?

Let's take a look at what makes up the experience we term "friction." Grab your trusty copy of *Computer Graphics: Principles and Practice* and set it on the table. Give the book a push with a small horizontal force. Notice that if the force is small, the book will not move. As you increase the force, you will reach a point where the book will start moving. Once it's moving, you may notice that it takes a little less force to keep it moving.

How is it possible for a smooth book on a smooth table to create a force that resists your efforts to push it? Well, it turns out that even relatively smooth surfaces are actually pretty rough if you look closely enough. It's this roughness that opposes your efforts. But even more interesting is the fact that on a smaller scale, when objects rest against each other, atomic bonds tend to form between them. These bonds form a kind of glue that makes it necessary to apply extra force to get an object moving.

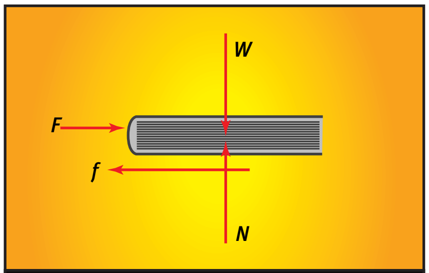


FIGURE 1. *A book in a state of static equilibrium.*

It's possible to measure the effect of this roughness. In fact, this is exactly what Charles Coulomb did in the late eighteenth century. He established a theory of dry friction (since called Coulomb friction) that predicts the maximum friction forces that are exerted on an object in contact with a dry surface before that object moves and becomes dynamic. The theory also predicts the friction forces that the surfaces exert when they are in motion relative to each other.

Don't Give Me No Static

When you are applying force on the book, the friction force opposes your efforts. Let's take a look at a diagram of this situation. Figure 1 shows a free body diagram of the book in static equilibrium, meaning that the book is not moving.

Since the book is in static equilibrium, we can determine a number of things via the principles of statics. The normal force, N, to the collision of the book with the surface is equal in magnitude to the weight of the book, W. Also, the friction force, f, must also be equal in magnitude to the force being applied on the book, F.

$$N = W$$
$$f = F$$
$$f \le \mu_s N \quad \text{(Coulomb Static Friction)}$$

When not fighting the friction that keeps his butt planted in Redondo Beach, Jeff creates custom 3D real-time graphics applications at Darwin 3D. What's the roughest surface you know? E-mail it to him at jeffl@darwin3d.com.

The Coulomb static friction model states that the magnitude of the friction force is less than or equal to the normal force, *N*, multiplied by a constant coefficient of static friction, μ_s. This coefficient describes the degree of smoothness between the two surfaces and generally depends on the material composition of the contacting objects. This value typically varies from 0 (which would be a perfectly smooth, frictionless surface) to 1 (for a very rough surface). Some examples of coefficients of static friction can be seen in Table 2.

There are some circumstances where μ_s can actually be greater than 1. Drag racing tires, for example, are designed to be sticky so that the friction force they exert is greater then the normal force exerted by the road.

When the force you are applying on the book causes the book to be on the verge of sliding, the friction force that opposes your efforts is at its maximum. At this point, slip is said to be impending. Through statics you can calculate the magnitude of the force necessary to cause this slip.

$$f = \mu_s N$$

(Coulomb static friction model)

$$f = F$$

(Objects are in static equilibrium)

$$F = \mu_s N$$

(The maximum *F* before a slip occurs)

Therefore, the maximum force that can be applied on the book before it begins to slip is $\mu_s N$. What is interesting, and complicated, about static friction is the fact that the friction force increases to equal the applied force until this threshold has been reached.

What Happens Then?

Once the applied force is greater than the slip threshold, the object starts moving. We now leave the world of statics and enter the world of dynamics. It's actually very similar to static friction. The magnitude of the friction force between two dry contacting surfaces that are sliding relative to each other is

$$f = \mu_k N$$

where μ_k is the coefficient of kinetic friction. This force resists the motion of the two bodies. Its direction is opposite the vector of relative velocity between the objects. In general, the value of μ_k is smaller than μ_s. However, this does not always have to be the case.

That covers the Coulomb dry friction model in both static and dynamic situations. By simply implementing these two methods, you can create a world represented by interesting physical properties.

Material	*Coefficient of Static Friction*
Metal on Metal	0.15 – 0.20
Wood on Wood	0.25 – 0.50
Metal on Wood	0.20 – 0.60
Rubber on Concrete	0.60 – 0.90
Metal on Stone	0.30 – 0.70

TABLE 2. *Some coefficients of static friction.*

How's This Good For Games?

An obvious application of the Coulomb dry friction model is for travel over surfaces. You may have a game that requires a character to travel over various types of terrain. By specifying different coefficients of friction for different types of terrain (asphalt, grass, ice, and so on), you can simulate movement over this terrain in a realistic, and even more importantly, a physically consistent manner.

Many games simulate friction simply by reducing the velocity by a percentage based on the surface type. This may seem at first to be the same thing as the dry friction model described above. However, it differs from it in many critical ways. By adjusting the velocity directly, you eliminate the side effects of applying the friction as a force. These side effects are what make objects in the physical simulation behave the way players expect them to behave. These small breakdowns in the simulation make it glaringly apparent that the world is fake. Perhaps an example would help here.

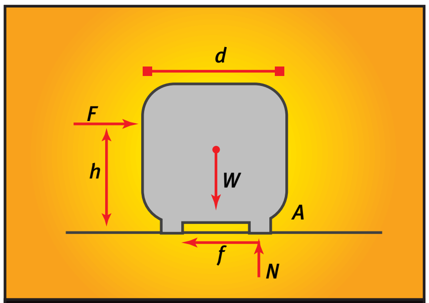


FIGURE 2. *Forces exerted on a box as it verges on tipping over.*

The Adventures of Sara Craft

Say I'm creating an adventure game starring a beautiful woman named Sara running around a dangerous, mystical temple in a stunning cocktail dress. To escape from the temple, Sara must manipulate a series of wooden boxes to activate various switches embedded in the floor. (Don't blame me, my producer came up with the concept.)

Sara pushes the boxes around by applying a horizontal force to the objects. If I do not consider friction at all, then once the boxes are sliding they will slide all around the room, bouncing off the walls forever. Clearly something needs to be done. So, I simply reduce the velocity of the object as it slides around. This can be made to look pretty good. However, there is still a problem.

If you have ever pushed a box really hard, particularly if your point of contact is near the top of the box, the box will sometimes tip over before it starts sliding. In fact, if you throw a box across the room, once it hits the floor it will tumble all over the place instead of simply sliding to a halt. People are used to these facts. They live with them every day. If your world does not address these behaviors, it will not feel right.

But why does the box tip over? Well, guess what, it is all about friction. Take a look at the box in Figure 2. Sara will be applying a force, *F*, to the box *h* units above the ground. What I'm looking for is a state for the system where the box is about to tip over at point *A*. I can apply the principles of statics to solve this problem. (If you are not familiar with statics, check out the For Futher Info section at the end of this column.) For an object to be in sta-

FIGURE 3. *You can control how much force Sara must exert on the box before it moves.*

tic equilibrium, the sum of all forces and the sum of all moments in the body must equal zero.

When the box is about to tip over, there is only a reaction to the ground at point *A*. The support on the other side has no reaction to the ground. Therefore, we can state the equilibrium equations. Let me start with the sum of forces.

$$\sum F_X = F - f = 0 \qquad F = f$$
$$\sum F_Y = W - N = 0 \qquad W = N$$

The sum of horizontal forces consists only of *F* and *f*, and they directly oppose each other. In the vertical direction, the weight *W* and normal force *N* are also equal and opposite. The sum of moments however, is a bit more complicated. You may remember from physics that the moment of a force about a point *P* is

$$M_P = DF$$

where *D* is the perpendicular distance from the point *P* to the line of action of the force *F*. Forces are sliding vectors, meaning that they act equally along their entire line of action. Let's look back at the drawing in Figure 2. When the object is about to tip over, it makes sense to look at the sum of moments about the point *A*. There are two moments being applied about point *A*. The force Sara is applying, *F*, and the force of the weight of the object, *W*.

$$M_{AF} = hF$$
$$M_{AW} = (d/2)W$$
$$\sum M_A = hF - (0.5d)W = 0$$

At the point of equilibrium where the box is about to slip,

$$f = \mu_s N = \mu_s W$$

So, I can substitute leaving

$$\sum M_A = h(\mu_s W) - (0.5d)W = 0 \qquad h = (0.5d)/\mu_s$$

If Sara applies the force at a point $(0.5d)/\mu_s$ units high or higher on the box, the box is going to tip over before it starts sliding. What's even more interesting is the fact that the equation above states that the value for h is not dependent on anything other than the dimensions of the box and the coefficient of static friction. The magnitude of the force F does not matter at all. It may seem that if Sara pushes harder, the box would be more likely to tip. Statics proves that this is not the case.

How Do I Use This Knowledge?

I am convinced. I want to have boxes that tip over if you push them too high. That seems like something cool to have in my game. But how do I go about accomplishing this task?

I have been building up the pieces I need. If you look back to my March and April 1999 columns ("Collision Response: Bouncy, Trouncy, Fun," and "Lone Game Developer Battles Physics Simulator"), I have a soft body dynamics package that models the forces and handles collision with surfaces. I will first handle the kinetic friction problem.

As I described above, the magnitude of the kinetic friction force is

$$f = \mu_k N$$

and the direction of the force is determined by looking at the current particle velocity. In my simulation, if the velocity of a point is greater than a certain threshold, ε, I determine that I need to use static friction for all contacting points. Listing 1 shows the code for calculating and adding in the force of friction.

The only change I really had to make to the structure of the program was to a storage space for the contact normal for contacting particles.

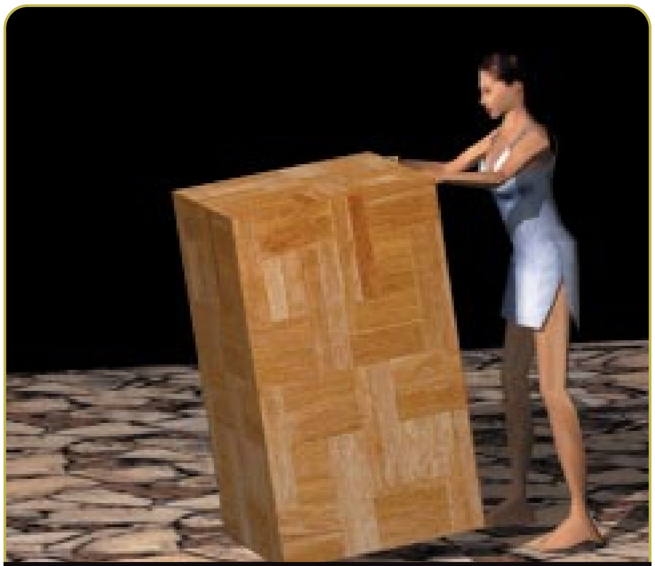


FIGURE 4. *Sara tips the box over instead of sliding it away.*

LISTING 1. *Code for calculating and adding in friction.*

```
// Calculate Magnitude of Fn
FdotN = DotProduct(&curParticle->contactN,&curParticle->f);

// Calculate Vt Velocity Tangent to Contact Normal
VdotN = DotProduct(&curParticle->contactN,&curParticle->v);
ScaleVector(&curParticle->contactN, VdotN, &Vn);
VectorDifference(&curParticle->v, &Vn, &Vt);

NormalizeVector(&Vt);      // Get the Direction of Vt
// Multiply By Normal force magnitude and Coef of Kinetic Friction
ScaleVector(&Vt, (FdotN * m_Ckf), &Vt);

// Add into the Force Accumulator
VectorSum(&curParticle->f,&Vt,&curParticle->f);
```

Static Friction

Handling static friction, however, is much more complicated. The problem is that static friction requires that I determine when each contacting particle makes the transition to sliding. From the calculations above, I know that the point of transition is when $F = \mu_s N$. Until that transition occurs, the static friction force needs to prevent sliding completely. That is, I need to make sure that the particle acceleration is kept at zero. Once the particle begins sliding, then the force opposes the acceleration and has a maximum of $\mu_s N$. All of these conditions lead to a situation that is too complex to be calculated in my simulation. David Baraff (see For Further Info) suggests a couple of approximations.

The more complicated method Baraff suggests is to approach static friction as a quadratic programming problem. However, this method is prone to failure in certain circumstances. The other suggestion, fortunately, is easy to implement.

First, establish a velocity threshold value ε which determines when to use static friction. This threshold is then used to scale the friction force as the velocity varies from 0 to this threshold. The formula for calculating the static friction force then becomes $F = (\mu_s N)(v/\varepsilon)$. This force is applied in the direction opposite the velocity of the particle. Listing 2 contains the code for handling the static friction forces.

LISTING 2. *Code for handling static friction forces.*

```
// Calculating Magnitude of Fn
FdotN = DotProduct(&curParticle->contactN,&curParticle->f);

// Calculating Vt Velocity Tangent to Contact Normal
VdotN = DotProduct(&curParticle->contactN,&curParticle->v);
ScaleVector(&curParticle->contactN, VdotN, &Vn);
VectorDifference(&curParticle->v, &Vn, &Vt);
Vmag = VectorSquaredLength(&Vt);
NormalizeVector(&Vt);            // Get the Direction of Vt
if (Vmag > STATIC_THRESHOLD)     // Handle Static Friction
{
   ScaleVector(&Vt, (FdotN * m_Ckf), &Vt);
// Multiply By Normal force magnitude and Coef of Kinetic Friction
   VectorSum(&curParticle->f,&Vt,&curParticle->f);
}
else   // Handle it as Kinetic Friction
{
   Vmag = Vmag / STATIC_THRESHOLD;
// Multiply By Normal force magnitude and Coef of Static Friction
// And Static approximation ratio
   ScaleVector(&Vt, (FdotN * m_Csf * Vmag), &Vt);
   VectorSum(&curParticle->f,&Vt,&curParticle->f);
}
```

A Word about Integration

In order for this static friction approximation to work, the particle must build up some velocity in order for the static force to kick in. If the value of ε is too large, it can cause the object to crawl around a little. By reducing this value, the crawling effect can be eliminated.

One unfortunate side effect of this approximation of static friction is that it can play hell with your integrator. When the particle is moving and subject to kinetic friction, things work well. However, when static friction kicks in, the direction of the static friction force swings wildly with small fluctuations in velocity. This plays havoc with the integration. If the value for ε is too small, the differential equations can become “stiff,” requiring more complex integration techniques (See “Lone Game Developer Battles Physics Simulator,” Graphic Content, April 1999).

Let’s Drag

Now I can get objects to tumble around realistically as well as slow to a halt based on the current coefficients of friction. You can download the source code and executable to the sample application from the *Game Developer* web site (http://www.gdmag.com). ■

FOR FURTHER INFO

• Baraff, David. “Coping with Friction for Non-Penetrating Rigid Body Simulation,” *Siggraph Proceedings:* July 1991, pp. 31–40.

• Beer and Johnston. *Vector Mechanics for Engineers: Statics, Sixth Ed*. New York: WCB/McGraw-Hill, 1997.

• Hecker, Chris. “Behind the Screen” columns. *Game Developer*, October 1996–June 1997. Also available on Chris’s web site at http://www.d6.com.

• Lötstedt, P. “Numerical Simulation of Time-Dependent Contact Friction Problems in Rigid Body Mechanics.” *SIAM Journal of Scientific Statistical Computing* Vol. 5, No. 2 (June 1984): pp. 370–393.

Talking Heads: Working with Expressions

Last month, we outlined the steps required to assemble a fully functional facial hierarchy. We discussed the reasons and motivations behind this time-intensive and potentially rewarding technique, and performed a rigorous examination of the physiological foundation of our hierarchical setup.

This month, we'll discuss the procedures for efficiently animating the facial hierarchy. We'll get our hands dirty setting up linked expressions and constraints, and by the end of the discussion we'll have an efficient, streamlined, and portable process for rapidly animating several different types of human heads.

Facial Animation Methodology

The main focus of the hierarchical method is generating a set of animation controls that will allow the animator to generate a wide variety of facial expression and speech animations rapidly. In order to be truly useful, this process must also be generic and portable, so that the animation controls created can be transplanted onto subsequent hierarchies with minimal effort. With the final result in mind, the process can be broken down into a few discrete steps, as the flowchart in Figure 1 outlines. (For a detailed discussion on generating the facial hierarchy, please see last month's column, "Talking Heads: Hierarchical Facial Animation in Real-Time 3D.")

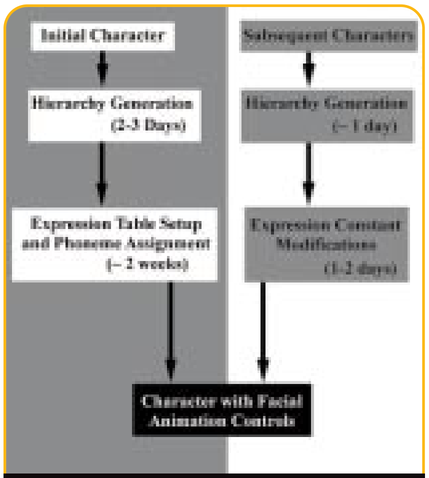

Initial Character
Hierarchy Generation (2-3 Days)
Expression Table Setup and Phoneme Assignment (~2 weeks)
Subsequent Characters
Hierarchy Generation (~1 day)
Expression Constant Modifications (1-2 days)
Character with Facial Animation Controls

FIGURE 1. *A flowchart representing how to generate a hierarchical set of animation controls.*

Clearly, the bulk of time spent on facial animation is in setting up the expression tables and assigning the phoneme keys. This is a rigorous and time-consuming process which can take up to two weeks for the first character, depending on the complexity of the mesh and the skill of the animator. However, if the technique is done correctly, subsequent character hierarchies can bypass this step completely, and with a few days invested into modifying the expression constants, the prep time for facial animation on an entirely new character can be reduced to only a few days.

Getting Started

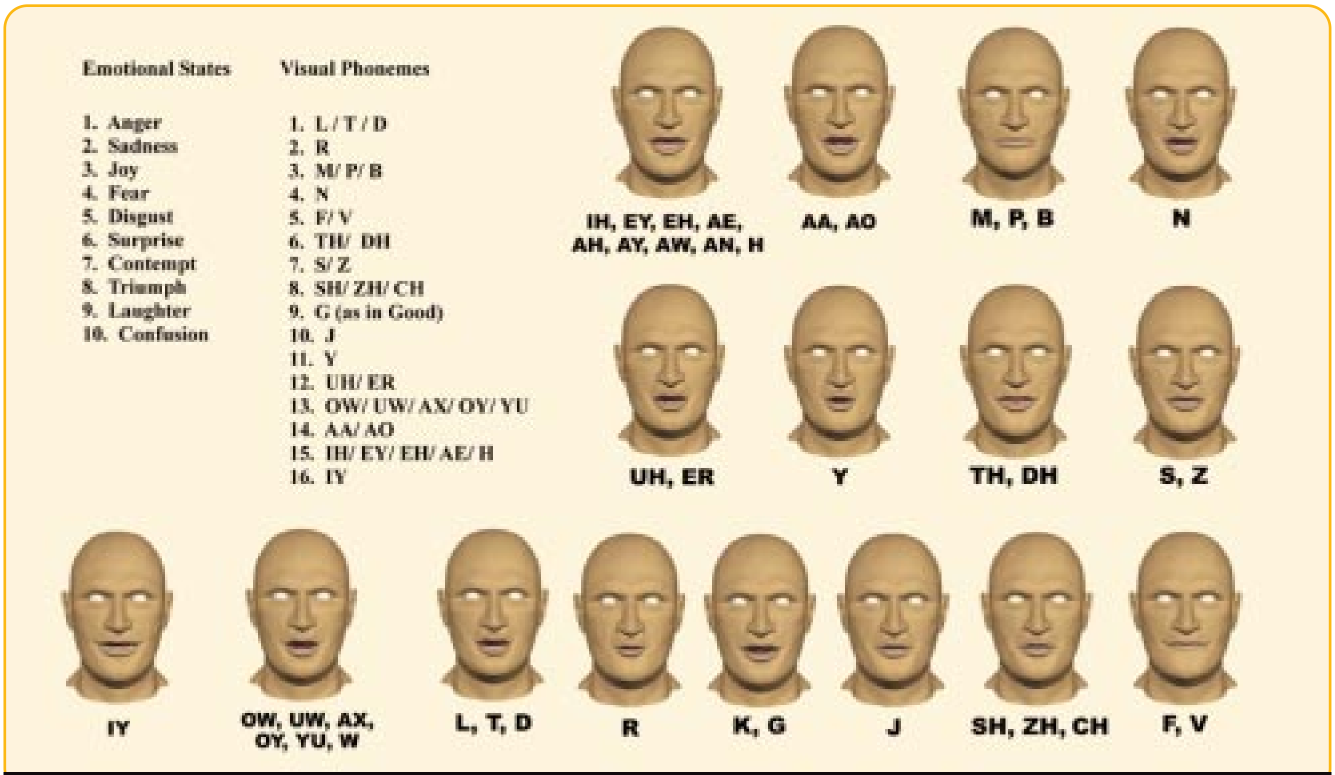
Before we start setting up our expression tables, it would be useful to reiterate the reasons behind our particular nodal setup. Ultimately, the facial animation we generate will be used to allow our characters to communicate in some way with the player. This communication can take multiple forms: non-verbal, as in the case of a conveyed emotion such as anger, fear or surprise; verbal, speech-driven communication; or a combination of emotion and speech. While the non-verbal, emotive expressions utilize the entire range of facial nodes, the speech-driven animations will deal only with those nodes around the mouth.

If we examine the hierarchy in Figure 2, we see that the nodal setup lends itself to just this type of breakdown. The animations used for emotion will involve linked expressions driving both the upper and lower facial node branches of the hierarchy, while the speech-driven animations will involve expressions driving only the lower facial node and its children. When we set up our expression tables, we will therefore generate two discrete sets of animation controls, one for each region of the face.

Setting up the Animation Controls

Once the hierarchy is in place, the next step in the process is to identify which emotions and phonemes will be animated for the character. While the game's genre and character background will influence this decision (if your characters are never going to cry, for example, you don't need to add the emotions of deep sadness to the animation controls), there are standard conventions which are useful. Figure 3 shows an example of these, although

Mel Guymon has worked in the games industry for several years, with past experience at Eidos and Zombie. Currently, he is working as the art lead on DRAKAN (http://www.surreal.com). Mel can be reached via e-mail at mel@surreal.com.



Emotional States	Visual Phonemes
1. Anger	1. L / T / D
2. Sadness	2. R
3. Joy	3. M/ P/ B
4. Fear	4. N
5. Disgust	5. F/ V
6. Surprise	6. TH/ DH
7. Contempt	7. S/ Z
8. Triumph	8. SH/ ZH/ CH
9. Laughter	9. G (as in Good)
10. Confusion	10. J
	11. Y
	12. UH/ ER
	13. OW/ UW/ AX/ OY/ YU
	14. AA/ AO
	15. IH/ EY/ EH/ AE/ H
	16. IY

FIGURE 3. *The 16 visual phonemes as identified by Fleming and Dobbs in* Animating Facial Features and Expressions.

depending on which convention you choose to follow, you may have anything from 5 to 15 emotional states and from 9 to 25 visual phonemes included in your control system. For further information on identifying visual phonemes, please see Jeff Lander's column, "Read My Lips: Facial Animation Techniques," (Graphic Content, June 1999) and Fleming and Dobbs' *Animating Facial Features and Expressions* (Charles River Media, 1998).

Once you have identified the actual target phonemes and emotions, you can then build the controls into the scene. Recall that earlier we discussed separating the control scheme between the upper and lower facial nodes. For the upper facial nodes, we will create a set of controls including only the emotional states identified in Figure 3. For the lower facial nodes, we'll create a similar set of controls made up of the emotional states plus the visual phonemes. Each control will consist of a slider free to move only in the vertical direction, with its vertical range of travel limited between zero and one. For our example this gives us a set of

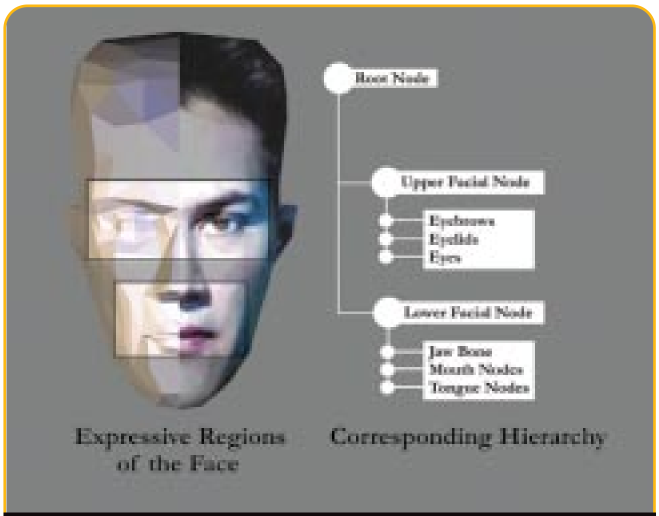
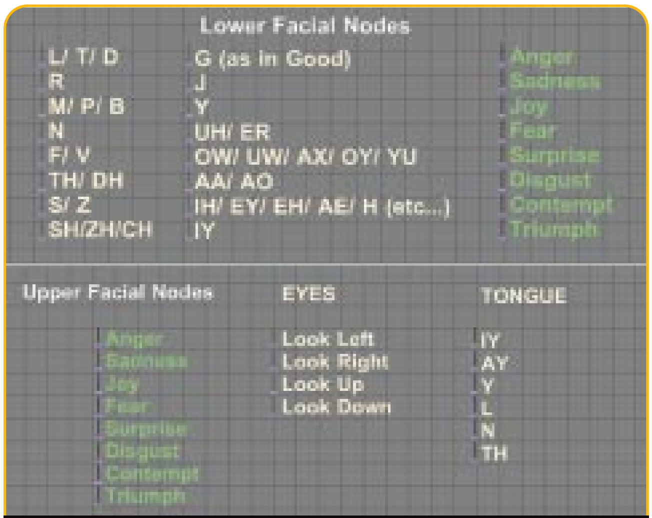


FIGURE 2. *A nodal setup hierarchy.*



Lower Facial Nodes

L/ T/ D
R
M/ P/ B
N
F/ V
TH/ DH
S/ Z
SH/ZH/CH

G (as in Good)
J
Y
UH/ ER
OW/ UW/ AX/ OY/ YU
AA/ AO
IH/ EY/ EH/ AE/ H (etc...)
IY

Anger
Sadness
Joy
Fear
Surprise
Disgust
Contempt
Triumph

Upper Facial Nodes

Anger
Sadness
Joy
Fear
Surprise
Disgust
Contempt
Triumph

EYES

Look Left
Look Right
Look Up
Look Down

TONGUE

IY
AY
Y
L
N
TH

FIGURE 4. *A typical controller setup.*

ten controllers for the upper facial nodes, and 26 controllers for the lower facial nodes (see Figure 4). Once we've set up our linked expressions, we can animate the face simply by sliding the controllers up and down.

Defining the Linked Expressions

Now we've come to the critical step in the process, defining the linked expressions. For many animators, this represents a seemingly insurmountable stumbling block , since working with equations and variables seems so foreign to working with keyframes and function curves. In reality, we will be using only a single equation and simple arithmetic rules to define our entire control system. The governing equation is given in Figure 5.

In order to make use of this equation, some of the values on the right hand side of the "=" sign need to be determined. First, record the initial local coordinates of each node in X, Y, and Z. These are the base values that will go into your equations as the initial position (X_{n_0}). Next, you need to pick one of the controllers from the list and position the nodes in the face to create the corresponding visual expression. For instance, in Figure 6 we can see that the node corresponding to the *orbicularis oris* muscle has been positioned for the facial animation "Disgust." The initial position (Y = 1.232) and "Disgust" position (Y=1.836) of the node have been recorded, and the net change, or

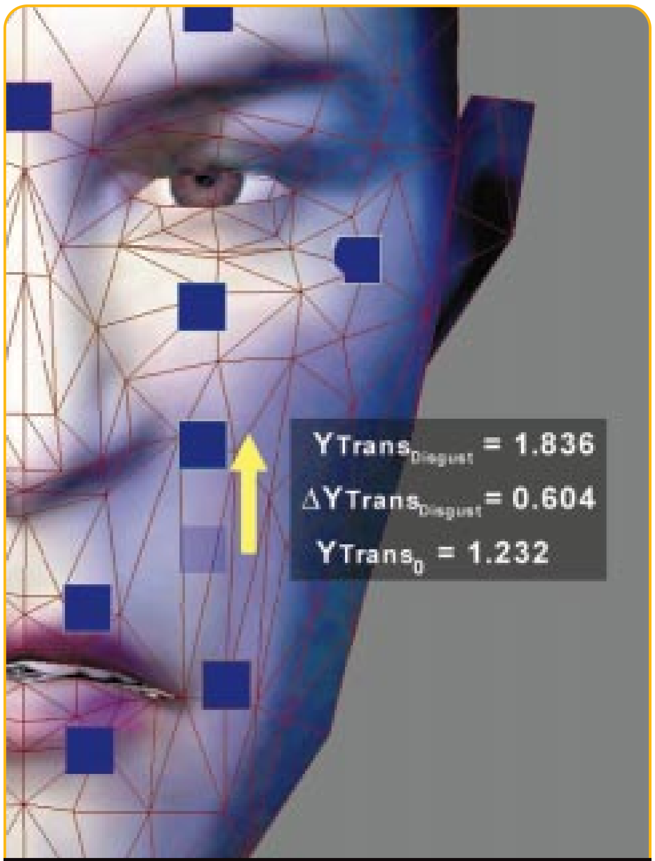


FIGURE 6. *The node for the* orbicularis oris *muscle, positioned for the facial animation "Disgust."*

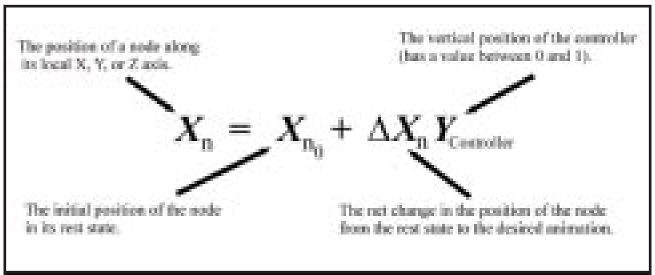

The position of a node along its local X, Y, or Z axis.
The vertical position of the controller (has a value between 0 and 1).
$X_n = X_{n_0} + \Delta X_n Y_{Controller}$
The initial position of the node in its rest state.
The net change in the position of the node from the rest state to the desired animation.

FIGURE 5. *The equation governing our control system.*

ΔY_{Trans} has been determined (1.836 – 1.232 = 0.604). The corresponding equation that will define this relationship is

$$Y_{Trans} = 1.232 + (0.604)Y_{Disgust}$$

so that the vertical position of the node at any given time is driven by the vertical position of the controller. Note that if you input a value of zero for the $Y_{Disgust}$ into this equation, you arrive at the result that the node stays in its initial rest position. This is a good first check to make sure your equation has been set up correctly: if the controller doesn't move, the node that's linked to it shouldn't move either. Conversely, if the controller is moved to its full extent, $Y_{Disgust}$ would be given a value of 1.0, and the node's Y_{Trans} value should equal 1.232 + 0.604 = 1.836, which is the final "Disgust" position initially determined. An intermediate position of the controller gives an intermediate result, so that the animation is totally scalable between 0 and 100 percent. And although in this example, the node in question moved only in the vertical plane, most of the facial nodes will move along more than one axis as they animate, so that the equations for all three axes will need to be determined for each node, as follows:

$$X_{Trans} = X_{Trans_0} + (\Delta X)Y_{Disgust}$$
$$Y_{Trans} = Y_{Trans_0} + (\Delta X)Y_{Disgust}$$
$$Z_{Trans} = Z_{Trans_0} + (\Delta X)Y_{Disgust}$$

The same basic set of equations is applicable to rotational nodes as well (such as the tongue, eyes, eyelids, and jaw). The positional variables of X, Y, and Z translation are simply replaced with X, Y, and Z rotational values. The net change, sampled in the same way (by subtracting the initial rotational value from the final rotational value), is entered in degrees.

Multiple Controllers on a Single Node

Setting up the controllers seems fairly straightforward so far, so why does it take so long (a few weeks at least), to set up all the controllers for a single facial hierarchy? In our example, there were 26 basic controllers which needed to be assigned. And, since most programs don't allow you to assign multiple expressions governing a single variable, the equations we will generate will be additive. That is to say, the expression governing a single node will have modifiers from every controller included within a single expression. So in our case, it's conceivable that for any given node, the expression for each of the three positional coordinates will have 26 factors in the equation, looking something like

$$X_{Trans} = X_{Trans_0} + DXY_{L/T/D} + DXY_{R} + DXY_{M/P/B} + DXY_{N}$$
$$+ DXY_{F/V} + DXY_{TH/DH} + \bullet\bullet\bullet + DXY_{Laughter} + DXY_{Confusion}$$

The final result of all this work is a nodal hierarchy linked through expressions to a simple controller system. To begin lip-synching and facial animation, simply animate the controllers and preview the result in real time. And since each node's expression includes modifiers from every controller, the initial set of basic controllers can be blended and used together to create thousands of unique facial expressions.

The Importance of Portability

As we discussed earlier, in order for the hierarchical method to be advantageous, it must be truly portable. That is, once the initial hierarchy has been created, it must be applicable to additional characters with a minimal amount of work, and that means keeping the expressions intact. If the subsequent characters' faces share the same general shape and size, then the nodal hierarchy may be directly applicable without any modification. However, if the characters differ significantly in weight, age, or gender, it's unlikely that the expressions for one will work flawlessly for the other. Figure 7 shows a comparison between two facial types which are different enough to warrant modifications in the underlying expressions.

The one advantage we have working for us is that all human and humanoid faces share the same basic characteristics. The bone construction and muscle characteristics common to one will be common to another. This means that although they may look different on the outside, they act the same on the inside, and that's what really matters to us, since the nodes we created mimic the form and function of the underlying facial muscles.

In Figure 7, the source model on the left has larger features, and due to gender differences, slightly different proportions. The head and jaw are more squared-off, the nose is broader, and the head more narrow. The target model on the right has a rounded face, proportionally larger eyes, a more delicate nose and chin, and so on. Regardless, as you can see by the placement of the respective nodes, the basic orientation and nodal structure, and the relative positions of the nodes to each other are almost identical between the two models, once you account for scaling differences.

Because of this feature, modifying the expression equations to accommodate the new facial structure is exceedingly simple. All that needs to be done is to change the initial rest position term X_{n_0} to the new initial position and the expressions become valid. Obviously, since the proportions in the new character are different from the original, the net change terms in each equation will not be exact. However, this is mitigated by the fact that the net change term operates on a variable which is under direct control of the animator: as the animator operates the controller, it will immediately become clear which net change terms need adjustment. But, the process is intuitive and much less time-consuming since the expressions are already in place.

Parting Words of Wisdom

This wraps up our discussion of hierarchical facial animation, and with your controllers set up and your hierarchies in place, you're ready to begin animating your talking heads. And while there is no getting away from the fact that facial animation can be extremely tedious, setting up a control system like this can reduce animator headaches and allow more interactive characters to populate our virtual environments. Here are some parting thoughts to keep in mind:

SET UP YOUR INITIAL POSITION CORRECTLY. In some programs, children in a hierarchy inherit the translation and rotation values of their parents. This can wreak havoc with your expression system if the offsets are not accounted for in determining the initial position.

USE EFFICIENT CONTROLLER SETS. Take some time to identify which controllers you need to have. Don't waste time generating a controller that you will never use. Remember, since the expression equations are additive, you can always go back and add terms at the end of the equation. It's better to have started out with too few controllers than too many.

TAKE YOUR TIME. As with everything, an ounce of prevention is worth a pound of cure, and this is especially true here. Because the process is so front-loaded, early mistakes can evolve into huge nightmares later on, and a few extra days or weeks spent setting up the building blocks of your animation system can save months of frustration. Remember that doing it right is more important than doing it fast. ■

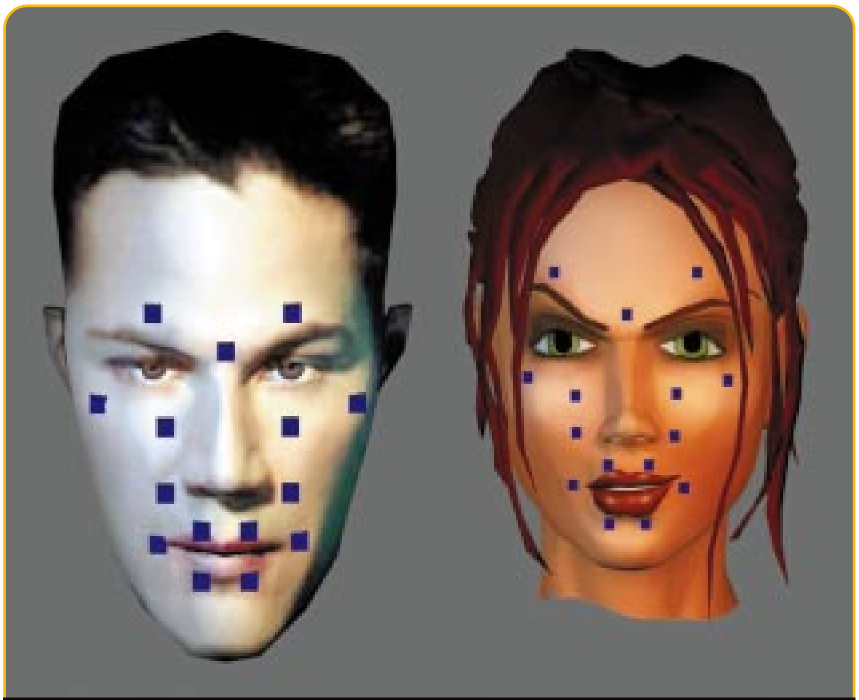


FIGURE 7. *A comparison of facial hierarchies for two different characters.*

Mac Games: The Hype and the Fizzle

I saw Steve Jobs's keynote at Macworld in San Francisco earlier this year, and it was quite a show. I must admit, I'm probably in the same boat as most people who can't quite comprehend the fascination with Apple that seems to consume the general public. However, I believe it has everything to do with Jobs's star power.

Apple's interim CEO is probably the best presenter, salesman, and bravado performer the computer industry has. When he stood on that stage in the Moscone Convention Center and said that Apple was serious about games, you were right there with him.

As if that weren't enough, Jobs then brought out John Carmack, the geek messiah of hardcore gaming, and he knew he had even the skeptics in the palm of his hand. Buttressed by the strength of Carmack's mere presence, Jobs managed to secure more ink and web space for Apple in the game world than at any time in the last six years that I can recall. P.T. Barnum would have been proud, but there has to be more to it than Jobs's showmanship.

The Apple Doesn't Fall Far from the Tree

It's easy enough to believe that Apple is the same company today that most of us in the industry knew in the 1980s. It has the same leader, it has a cool product to set it apart, and it's taking on the establishment. Unfortunately, the establishment this time around isn't IBM, a slow moving behemoth with other things on its mind besides the color of your computer. This time, the establishment is a company that got the Mac religion better than Jobs did, and turned it into a monopoly: Microsoft and its Windows operating system have won the battle for the desktop. Everything that isn't Windows seems an aberration, a positioning statement, or a niche market play. From one point of view, Apple is striving to get back to where it was in the early 1990s, a pioneering company with a slice of the personal computer market that stops Windows just short of absolute control of the desktop. But does that make the Mac a gaming platform? The market figures seem to indicate otherwise, but game players are a difficult lot to pigeonhole, so there must be a Mac clan out there worth targeting, right?

In truth, comparing the Mac game market to Windows is like having an elephant and a mouse play see-saw; you know it isn't going to be much of a contest. While Mac-only games are almost a business novelty these days, game publishers do succeed on the Mac by other means. Hybrid games that support both Windows and Mac platforms are strong market contenders. It's almost as if publishers are saying Mac and Windows are on a level playing field, but that's not true. In fact, most of Apple's recent evangelism of games has served to highlight the availability of the most popular Windows game titles on the Mac, where they have been largely absent in recent years. That's why the real coup for Apple has been the promise that QUAKE 3: ARENA would appear simultaneously in Mac and Windows versions. At best, the Mac gaming market can only hope for parity with the Windows platform insofar as content choice is concerned, which means the same A–list titles appearing on both platforms. This isn't progress for Apple, but it does take the Mac back close to where it was at the height of its success. In those heady days, Apple was actually the starting point for some of the game market's greatest success stories. You may remember MYST, which still plays to a worldwide audience despite a more advanced sequel and the ravages of time.

To appreciate the Mac market you have to dissect it in other ways. If you choose to look at the Mac market from the perspective of a niche market player, you might just find reasons to approach the platform more aggressively on your next development project. For instance, according to IDC Research, game software revenues for the Mac OS comprised 11.2 percent of worldwide entertainment software revenues in 1997, and it will show a compound annual growth (CAGR) projected to 2002 of 15.3 percent — not too shabby. Of course, 32-bit Windows games will see 31.4 percent CAGR in the same period, but that still leaves Mac OS games contributing 10 percent of worldwide gaming revenues projected for 2002. Presently, Windows and Mac OS games deliver between $2.5 and $2.9 billion in revenues, and IDC projects that figure will top $4.7 billion by 2002. That makes the Mac OS market for games a robust one; certainly nowhere near as big as the Windows market, but not one that should be ignored either. Mac OS games may best be viewed as an incremental sale, rather than a unique sale, and that

Omid Rahmat is the proprietor of Doodah Marketing, a digital media consulting firm. He also publishes research and market analysis notes on his web site at http://www.smokezine.com. He can be reached at omid@compuserve.com.

	No. of Titles	1998 Revenues	1998 Units
Mac-Only Games	237	$19.4 million	873,000
Hybrid Games	300	$136.4 million	6.227 million

This information is courtesy of the market research firm PC Data. The term "hybrid games" refers to titles that are designed to run on both Windows and Macintosh platforms. Unfortunately, there is no data to show whether a hybrid game is being bought for a Mac or PC platform. Number of titles is based purely on titles that have sold over $500.

TABLE 1. *Mac's slice of the game pie in 1998 revenues and units sold.*

seems to be the sticking point for the game industry, driven as much as it is by hits these days.

What Should You Do for the Mac Market?

Obviously, a Mac-only gaming strategy is not going to make anyone rich, judging by the total market opportunities. Neither is ignoring the Mac platform a good idea if you are serious about increasing your title's exposure to the biggest possible audience. There are a number of other factors at play in the Mac market that you can use to make your call. First of all, while you need success on the Windows platform to make or break your title, the Mac is an extension of your sales channels that can deliver healthy profits with little incremental costs. View it in the same way as you might view online sales versus retail store sales, for example. Each contributes a certain percentage to your overall game revenues, and you make your investment based on the expected return on investment in each channel.

Moreover, Apple has gone to great lengths to court game developers, and it has an installed base in excess of 22 million users. Therefore, by creating a hybrid title you're probably not going to add too much to your bottom-line costs, while ensuring that you reach the biggest non-Windows computer market there is. That's a sobering thought for all those developers hitting the Linux trail. As for actual bottom-line costs, I have heard figures as low as five percent of total development costs, to as high as 30 percent. It's all anecdotal. The ultimate cost of Windows and Mac hybrid support is up to your software engineering management. Generating a code base that can be easily modified from Windows to Mac is not a big issue these days, so the challenge is not technical, but rather more of a project management issue.

In addition to the return on investment, you should consider the other opportunities of being on the Mac platform. It's a self-contained universe that doesn't have the crowded masses of game development that inhabit the world of Windows. That's a natural cost savings because you don't have to fight so hard to be heard above the noise. The channels are clear cut, the shelf space is easy to target, and the distribution outlets are uniquely Mac-centric. Also worth noting is the fact that the Mac platform continues to be less support intensive than Windows. For one thing, you don't have as many permutations of hardware and components to contend with. That helps your profitability enormously, and again, if it isn't costing you much to develop a hybrid product, an added chance for profitability can't be bad.

The other thing that goes unnoticed sometimes is that Apple is going out of its way to support the game developer. Apple's developer site has lots of interesting information on everything from coding to marketing your game. Some of it may seem rather naïve to anyone who has been barraged by Microsoft Developer Network documents, but in general, it's clean, easy to get through, and a valiant show. I can't see too many new developers taking to the Mac platform over the Windows alternative, but I do think that Apple always offers the possibility of breeding another PRINCE OF PERSIA or MYST success story.

The only downside I can see to the Apple market is that Jobs is still Jobs, and Apple runs under his steam. Like all good salesmen, his attention span may be short once he's closed the deal, so once the Mac games market is fully rejuvenated, I just wonder what Apple will do for game players to up the stakes. In the world of Windows, there are plenty of people to push the hardware envelope, as we have witnessed in the 3D graphics arena, but on the Mac, that level of competition just isn't there. So long as Apple's customers hunger for games, there is a personal computer other than one running a Microsoft operating system. That may just be enough to keep the maverick game developer supporting the Mac. ■

Back

GAME AI:

Illustration by Robert Zammarchi

THE STATE OF THE INDUSTRY

B Y STEVEN WOODCOCK

It's been nearly a year since my first article outlining the then-current trends in the game development industry regarding game AI ("Game AI: The State of the Industry," October 1998). Since that time, another Christmas season's worth of releases has come and gone and another Game Developers Conference (GDC) has provided yet another opportunity for AI developers to exchange ideas. While polls taken at the 1999 GDC indicate that most developers (myself included) felt that the last year had seen incremental, rather than revolutionary advances in the field of game AI, it seemed that enough interesting new developments have taken place, which makes an update to my previous article seem natural.

I'm very pleased to say that good game AI is growing in importance within the industry, with both developers and marketeers seeing the value in building better and more capable computer opponents. The fears that multiplayer options on games would make good computer AIs obsolete appear to have blown over in the face of one very practical consideration — sometimes, you just don't have time to play with anybody else. The incredible pace of development in 3D graphics cards and game engines has made awesome graphics an expected feature, not an added one. Developers have found that one discriminator in a crowded marketplace is good computer AI.

Steve's background in AI comes from over a decade of SDI-related work building massive real-time distributed war games for the Air Force at the Joint National Test Facility. When he's not saving the world, he does AI development on a contract basis and goes target shooting when he gets the chance. Steve lives in Colorado Springs, Colo., with a very understanding wife and an indeterminate number of ferrets. He maintains a web page on game AI at http://www.gameai.com, and can be reached via e-mail at ferretman@gameai.com.

As with last year's article, much of the insights presented herein flow directly from the AI roundtable discussions at the 1999 GDC. This interaction with my fellow developers has proven invaluable in the past, and the 1999 AI roundtables proved to be every bit as useful in gaining insight into what other developers are doing, the problems they're facing, and where they're going. I'll touch on some of the topics and concerns broached by developers at the 1999 roundtables. I'll also discuss what AI techniques and developments seem to be gaining favor among developers, the academic world's take on the game AI field, and where some developers think game AI will be headed in the coming year or two.

Is The Resource Battle Over?

Last year there were signs that development teams were beginning to take game AI much more seriously than they had in the past. Developers were getting involved in the design of the AI earlier in the design cycle, and many projects were beginning to dedicate one or more programmers exclusively to AI development. Polls from the AI roundtables showed a substantial increase in the number of developers devoted exclusively to AI programming (see Figure 1).

It was very apparent at the 1999 GDC that this trend has continued at a healthy clip, with 60 percent of the attendees at my roundtables reporting that their projects included one or more dedicated AI programmers. This number is up from approximately 24 percent in 1997 and 46 percent in 1998 and shows a growing desire on the part of development houses to make AI a more important part of their game design. If the trend continues, we'll see dedicated AI developers become as routine as dedicated 3D engine or sound developers.

AI specialists continue to be a viable alternative for many companies that lack internal resources to dedicate developers exclusively to AI development. Several developers and producers present at the roundtables indicated that they had used independent contractors to roll the AI portions of their process with varying degrees of success. The primary complaints about using contract help were perhaps the universal ones — you never really know what you're getting, and maintaining good communication is, at best, a chore.

FIGURE 1. *Resources dedicated to AI development.*

	'97 CGDC	'98 CGDC	'99 GDC
Developers dedicated to AI	24%	46%	60%
% of the overall game CPU reserved for AI processing	5%	10%	10%

The most interesting comments, however, concerned CPU resources available to the AI developers (Figure 1). None of the developers answering the poll questions regarding CPU resources felt that they had too little CPU available. Everybody felt they could use more if they had it, but nobody said that they were having to fight tooth and nail for resources as they had in the past. This is an amazing turn of events, which is in stark contrast to previous years when AI developers complained often and bitterly of fighting the graphics engine guys for CPU cycles. The overall percentages of CPU cycles most developers felt they were getting didn't really change, but developers were feeling much less pinched than they had been in the past. When asked why this was the case, there were a variety of theories. Most developers felt that this was, quite simply, due to the fact that faster hardware is now standard on both PCs and consoles — 5 percent of a 400Mhz Pentium III is a heck of a lot more horsepower than 5 percent of a 200Mhz Pentium I. Others thought that the availability of faster 3D hardware, combined with greater expertise of the 3D engine manufacturers, had simply made 3D engines more efficient than they had been and thus freed up more CPU resources for other tasks. Whatever the reasons, everybody was happy about it, and they thought it would only get better as hardware got faster.

The one great problem mentioned by all was the impending-shipping-date-syndrome. Christmas hasn't moved from its place as an almost magical date for targeting new releases, and the increasing complexity of games in general hasn't made meeting deadlines any easier. While there are more programmers dedicated exclusively to the AI portion of game development now than there had been in the past, most developers felt that the task itself had become more difficult.

Part of the reason for this, of course, is the increasing importance of game AI itself — having made the case that good game AI is important in increasing the odds of a game's success, developers must now actually deliver better game AI. Quite simply, that takes time. When coupled with the fact that most AI testing can't really begin until substantial portions of the game's engine are up and running, you've got a situation wherein dedicated AI developers find themselves making compromises in the face of impending shipping dates.

Some developers also professed that part of the problem was the advances made in competing products. For example, after one real-time strategy (RTS) game introduced production queues, players started looking for all RTS games to do the same, and that means additional AI development for handling such things. There is also a desire on the part of most developers to avoid doing the "same old thing" in a new release.

Technologies in the Limelight

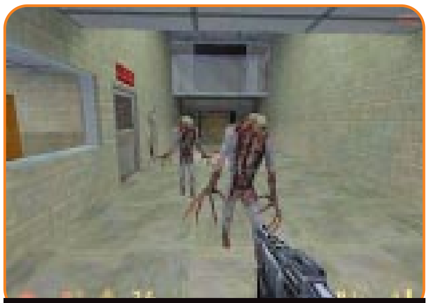
Exploring the AI technologies used by other developers in games has been a popular topic at past CGDC roundtables. Developers are increasingly turning to military and academic sources for new ideas and technologies (and those disciplines are turning their eyes on the game industry as well). Discussions with developers at the roundtables and at demo booths in the exposition hall yielded some interesting information about what technologies are in use today.

Rules-Based AI. Rules-based approaches to game AI, led by the Finite State Machine (FSM) and the Fuzzy State Machine (FuSM), continue to lead the pack as the most popular technologies among AI developers. The reasons for this remain the same:

- They're familiar to the developer, building on principles that are already well understood.
- They're easy to test against, making it simpler for developers to "customize" behavior in various ways, if necessary (something that happens in more games than one might think).
- They're still more familiar to most developers than the more exotic AI technologies, such as neural networks and genetic algorithms.

While every game shipped in the past year makes use of rules-based AIs to one degree or another, there were a couple that seemed to stand out in developer's minds as particularly interesting implementations. One of these was Epic Games' UNREAL, a first-person 3D shooter that provided some excellent examples of the complexity of behaviors available using FSMs and FuSMs. Taking the advances of Valve Software's HALF-LIFE one step further, the AI in UNREAL also makes heavy use of FSMs to control the behavior of the player's opponents to an often amazingly realistic degree. At higher levels, there is evidence of considerable intelligence on the part of monsters, which run away, hide when wounded, summon reinforcements, and can even lead the player into ambushes when possible. Herds of miscellaneous critters scuttle about the game levels using a fairly nice flocking algorithm, adding to the overall effect of the living world that the player has been thrust. All of this was done by the developers by layering FSMs, which were built on top of an extensible scripting system called UnrealScript (more on that below).

A game making heavy use of FSMs is


Valve Software's HALF-LIFE .

Epic Games' UNREAL .

Activision's CALL TO POWER. Billed as using "over-arching potentialities" to guide its strategic thinking, CALL TO POWER's AI actually makes heavy use of cascaded FuSMs throughout its design. The primary reason for this was straightforward enough, in that a number of different civilization personalities had to be accommodated in the design in order to reflect the differing governmental and militaristic bents of the various civilizations portrayed in the game. If the developers had used a strictly rules-based design to accomplish this, there would have been a considerable amount of special code to handle each civilization. Using FuSM technology allowed the developers to build on core AI engine in which its various decision-making thresholds could be modified by each civilization's unique personality and philosophical leanings.

This approach allows the game to accommodate a variety of different playing styles and technological research trees without bogging down the design in too many special cases. Every decision that a given civilization can make is based partly on the strategic situation, partly on that civilization's personality, and on the decisions it had made previously. Anytime something isn't terribly obvious, or not covered by a specific rule of some kind, the AI uses fuzzy logic (in the form of the FuSMs) to make a decision. This, in turn, results in an AI whose decisions are internally consistent and plausible, yet still leaves the chance for a surprise or two.

Extensible AIs. A number of recently-released games have featured Extensible AIs (ExAIs) in one form or another, building on a trend that began a couple of years ago with the release of QUAKE. The success of that game's QuakeC scripting language, which permitted players to build their own computer opponents, assistants, and companions (known as "bots") and trade them over the web, has inspired a number of other developers to build similar capabilities into their releases. Several developers at the 1999 roundtables mentioned that they were at least exploring the possibility of ExAIs in their projects.

To date, most ExAIs have cropped up in the first-person 3D shooter genre. Last year's UNREAL and HALF-LIFE provided players with interfaces through which they could devise their own rules for computer opponents. However, there were differences in implementation. UNREAL went with a general "directive-like" interface through which AI behavior is controlled using relatively simple commands, such as "Move forward until you see an enemy, then throw grenades." HALF-LIFE used a more traditional "programming-like" approach that somewhat resembles Perl or JavaScript. Both approaches have proven extremely popular with players and led to legions of users trading scripts and bots online for games based on both engines.

More recently, however, ExAI technology has been finding its way into other genres of games. Interplay's BALDUR'S GATE, a role-playing game (RPG) based in part on the Advanced Dungeons & Dragons paper RPG, uses scripts to control non-player characters (NPCs) within the game — including those that can be in the player's own party. These scripts allow players to specify the basic reactions of their NPCs to a variety of combat situations, permitting them to adjust behavior either to accommodate a player's particular style (making mages more cautious than they are by default, for example) or to create entirely new NPC classes. Several aficionados of the game have already seized on this last capability to develop a number of NPC classes not present in the original game, creating thieves, warrior-mage combinations, elven archers, and so on.

The AI scripts themselves are heavily rules-based in the HALF-LIFE vein, operating in a strictly linear fashion from top to bottom within the script. Thus, rules "later" in a given script might or might not ever "fire" depending on the circumstances of the game and whether or not the player overrides any particular NPC action (an option always available). Responses can be weighted to control their probability of occurrence, though there is no provision for being able to modify the internals of the game's AI engine itself. There are some pre-defined, basic strategies available for the player-cum-AI-designer to use, and, of course, the existing NPC scripts are readily available as examples of what can be done. Documentation shipping with the game is necessarily sparse (probably to help avoid too many support hassles), but a few web sites have sprung up on which tinkerers can exchange information.

Listing 1 shows a snippet of a script, which was kindly provided by BALDUR'S GATE enthusiast Sean Carley for my game AI web page. It's from a warrior AI he developed, and as you can see, the scripting system is very English-like in syntax.

However, adding ExAI capabilities to a game isn't at all easy, and most developers at the 1999 AI roundtables agreed with the opinion from previous years that the trend wasn't likely to become widespread. There are significant design considerations that have to be worked out if one desires to add the ability for players to modify a game AI to suit their tastes, not to mention the problem of after-sale support. Developers have to decide how they're going to provide these hooks (code interface? scripts?), how they're going to document them (in the manual? online? HTML on the CD? not at all?), and just how far they should go to bullet-proof

LISTING 1. *Sample BALDUR'S GATE AI script.*

```
IF
  // If my nearest enemy is not within 3
!Range(NearestEnemyOf(Myself),3)
  // and is within 8
Range(NearestEnemyOf(Myself),8)
THEN
  // 1/3 of the time
RESPONSE #40
  // Equip my best melee weapon
  EquipMostDamagingMelee()
  // and attack my nearest enemy, checking every 60
  // ticks to make sure he is still the nearest
  AttackReevalutate(NearestEnemyOf(Myself),60)
  // 2/3 of the time
RESPONSE #80
  // Equip a ranged weapon
     EquipRanged()
  // and attack my nearest enemy, checking every 30
  // ticks to make sure he is still the nearest
```

Interplay's BALDUR'S GATE.

the whole interface in the first place. (Whose fault is it if some player distributes an AI script that erases somebody's hard drive?)

These very issues were, in part, the reason why Activision somewhat de-emphasized its much-touted interface to the AI engine in CALL TO POWER. Originally, the development team had planned to provide full and total access to CALL TO POWER'S AI in such a fashion that players could have hypothetically replaced the game's AI with their own. The AI is completely encapsulated within a .DLL file, and it was planned to have players access it via header files that would have provided an interface to many of the internal functions (though the source itself was not going to have been released to the public). Users would have been completely on their own while using this interface — the support issues could have been nightmarish otherwise — and this approach would have allowed anybody who had the time and patience to replace CALL TO POWER'S AI completely with their own — a first in the industry.

Unfortunately for budding developers, the pressure of shipping on time and the design complications encountered while trying to implement this rather unique feature made that goal unrealistic. Activision was forced to drop that part of the plan (oddly though, you can still find a .MAP file listing the various function interfaces on the CD). Still, a number of extensible features made their way into the game, enough so that, although Activision isn't advertising the fact much, a number of players have begun making variations and trading them online. Players can modify unit attributes (all maintained in flat text files) and have access to the fuzzy logic rules sets used by the AI to set priorities for the strategic-level AI. This allows you to create new unit types and civilizations, in much the same fashion as UnrealScript permits new bots. In a similar vein, Microsoft's AGE OF EMPIRES provides much the same level of customization of units and civilizations, though emphasis is more on customization of the various personalities of each civilization type than on actual modification of their rules sets.

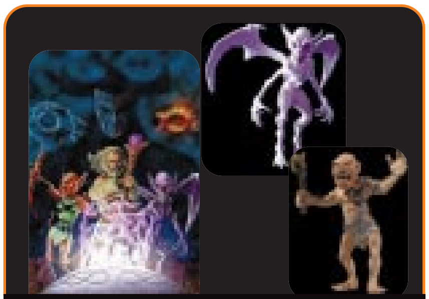
LEARNING AND STRATEGIC THINKING. Another trend that bubbled to the surface at the 1999 AI roundtables was experimentation with learning AIs in various games. While it was definitely a widely-held opinion that most games featuring learning AIs haven't really done a very good job of delivering, several developers had high hopes that they'd be able to incorporate some level of learning into their next round of releases.

Developers seem to be exploring a number of different approaches to simulate learning, most involve comparing the current strategic or tactical situation to similar past situations. Mythos Games, in their recently released MAGIC & MAYHEM, noted that they were doing localized assault planning by continually building a data file that describes how attacks had fared historically in previous scenarios. A proposed attack is compared to this database, and if it succeeded most of the time, it's actually carried out (the threshold is determined in part randomly and in part by the personality of the AI player). A "winnowing" algorithm discards "old" lessons so the learning file doesn't become too large. The developer reported that this approach resulted in an AI that gradually tailored itself to the player's style of play — a feature that is certainly something of a Holy Grail to AI developers.

Interestingly enough, some developers (roughly 20 percent of attendees) were experimenting with Artificial Neural Networks (ANNs) as a learning technology. ANNs have cropped up often in the AI roundtables as a potential solution to the learning-AI problem, but there are some interesting challenges in using the technology in games that have discouraged most developers to date. Historically, using ANNs within a game presents the developer with two particularly thorny problems: First, it can be very difficult to identify meaningful inputs and match them to outputs that make sense within the context of the game; and second, most ANNs learn through a technique called "supervised learning," which requires constant developer feedback. While it is possible to build ANNs that can learn unsupervised, there's no guarantee that they won't "go stupid" and become completely helpless players.

Most developers are trying to avoid these problems by training their ANNs exclusively during the development phase, then freezing them before the game actually ships. This allows them to let the AIs learn while playing against the development team and play testers without the risk that a shipping AI might wander off into some *Rain Man* universe of perception. The downside to this, of course, is that the game doesn't learn anything from the player, and so the whole effort boils down to an automated form of AI tuning (ultimately similar to using genetic algorithm to try to tune various game AI parameters). A developer of an upcoming sports game announced that he was working on a way to integrate unsupervised learning ANNs into his game, although he planned to include an option to reset the AI should the player feel it had become feeble-minded (or too strong a player, as the case may be).

Activision's CALL TO POWER.


Mythos Games' MAGIC & MAYHEM.

One big problem with learning AIs that caused much amused discussion at the roundtables was the fact that a learning AI is, by definition, unpredictable. This leads to huge problems when it comes time to do quality assurance testing on your game — how can anything be tested reliably if it behaves differently from game to game? How can a developer fix a bug if it's impossible to recreate the conditions that led to a certain behavior?

On a closely related vein, several developers noted that they were attempting to find AI technologies that would do a better job at strategic-level thinking and planning. To date, most strategy games do an adequate job at the tactical level — identifying cities or units to attack, taking advantage of unprotected assets, and so on — but do a lousy job at developing and implementing grand strategy. The problem, from a programmer's point of view, is basically one of optimization.

Most war games (ignoring for the moment most first-person shooters and RPGs, since they are primarily tactical in the extreme), whether real-time strategy or turn-based, do a much better job of optimizing small, tactical situations over larger, strategic ones. This leads to AIs that fight battles well but still manage to lose the war, often because they overlook solutions glaringly obvious to the human player. A large part of this situation is simply the result of the historical inclination of developers to build AIs at the unit level; for example, in a Civil War game, a cavalry unit might decide to attack an artillery unit without the presence of any other support. This in turn leads to an AI that often overlooks obvious attacks in favor of frittering away its forces. Adding in an ability for a unit to call for help balances things out somewhat, but that's still a far cry from strategic-level thinking.

Additionally, there's the problem that strategic-level planning may be very good for the war effort overall, but very bad for the individual unit. One example of this might be a brigade ordered to hold a vital mountain pass in the face of overwhelming enemy attack — the war might be won because the delaying action bought the time necessary to get reinforcements to the area, but the unit itself isn't likely to survive. An AI built to handle only unit-level thinking is going to have a hard time making this kind of trade-off. Chess game AIs are perhaps the one exception to this rule, but they're cheating, since most chess programs draw upon databases of thousands of games and simply pick the highest-scoring move available at that moment.

Many developers present felt that the time had come to redress this imbalance and were looking to a number of AI-related technologies for help. Some were building on the same techniques used for learning algorithms by using databases of previously-successful moves to develop plans for similar future moves. Others were looking at tools such as Influence Maps (see sidebar "Influence Maps in a Nutshell," p.40) to provide ways for their AIs to "see" the grand strategic picture. A few were hoping simply to solve the problem the same way most chess programs do, which is to build large databases of opening strategic moves based on feedback from play testers and the development team.

Interestingly enough, a vocal minority of developers felt the move towards developing better strategic AIs was primarily a waste of time, particularly in games in which players can't easily see the other side's forces. The theory they put forth was that if the player can't see what the computer is doing, why waste time on elaborate strategic AIs in the first place? A few well-placed but thoroughly plausible unit placements (via judicious cheating on the part of the AI) would go a long way towards providing the player with an enjoyable gaming experience. Many of this group felt that the mere appearance of a tank deep behind enemy lines would be ascribed a meaning by the player if the attack came at a particularly vulnerable time. They based this opinion on the reams of e-mail they had received from players that raved about the intelligence of the AIs in their games, when the AI was, in fact, cheating outrageously just to keep up.

Influence Maps in a Nutshell

Influence Maps (IMs) are an interesting AI technique with its roots in the field of thermodynamics, of all things. The technique is known by a variety of other names, such as "attractor-repulsor" and "force fields".

The basic IM algorithm is refreshingly simple for something in the AI field. Imagine an array which corresponds in size to the size of a strategic-level map. For instance, a strategic map of the U.S. might have resolution down to the state level — in that case, the array might consist of an array of five by ten values (one value for each state). Set all values of the array to zero. Adjust the value of each array element upward by one for each friendly unit in that sector of the map, or downward by one for each enemy unit in that sector. Then begin looking at each location of the array and adjusting the value found there by its neighbors. Typically values are increased by one for each adjacent friendly unit and decreased by one for each adjacent enemy unit.

Do this across the entire map and you now have a "picture" of sorts, that your AI can use tell how much control the two players have over the board. The sign of the value indicates which side has some control. Values near zero indicate areas where control is contested—the front. Large values (positive or negative) indicate strong control over an area.

There can be any number of variations on this basic algorithm depending on the needs of the game, of course, but the principle is the same regardless. This technique can be invaluable in providing all kinds of strategic disposition information to an AI, information which is often difficult to characterize otherwise.

— Steven Woodcock

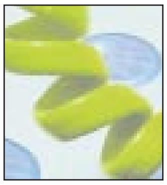
PATHFINDING. Pathfinding is a perennial favorite topic at the roundtables, but most developers this year were far more interested in finding ways to solve unusual pathfinding situations than in learning "how to." The A* algorithm (for more details, see Bryan Stout's excellent article, "Smart Moves: Intelligent Pathfinding," *Game Developer*, October/November 1996) has become the de facto solution to this problem for one very simple reason: It works, and it works well. A* has the added benefit of scaling well into newer games that feature 3D terrain, and it requires few tweaks and modifications.

The 3D pathfinding issues presented by the latest generations of first-person 3D shooters were generally felt to be nowhere near the problem most developers were afraid it might be. The early implementations of A* for 2D games had been adapted easily by most developers for the 3D environment, with most developers coming up with variations of the same solution of overlaying a system of nodes within the 3D environment against which paths were found. Some games generate the nodal network when the game map is loaded, while others simply load a pre-defined network as a part of the map data itself. At least one upcoming first-person-shooter style game, THE WAR IN HEAVEN from Eternal Warriors, features an AI that uses a pre-defined node map for its basic pathfinding, but goes one better by dynamically generating new nodes for finer control based on the tactical situation.

Developers at the roundtables were very interested in exploring ways to handle special case pathfinding problems. Identifying and dealing with highly-restrictive terrain (such as bridges or mountain passes) was a hot topic, since these terrain types can lead to traffic jams that make an AI look extremely stupid to the player. Most developers simply marked these terrain features by hand in some fashion in order to make them easy for the AI to identify — although this greatly complicated things when the AI had to deal with randomly generated maps. Many developers said that they solved the problem in part by assigning a special AI agent to play traffic cop, thus side-stepping the issue of bogging down individual unit AIs with the details of crossing a bridge politely.

Another problem of keen interest to developers was how to handle the issue of changing terrain gracefully. One of the failings of the A* algorithm is that it assumes the terrain over which it has calculated a path doesn't change — this is a bad side effect should the bridge you were planning to cross get blown up by an artillery round. To solve this problem, some developers were using D*, a dynamic A* variant tuned to handle changeable terrain, but none were happy with it due to the CPU hit required to recalculate paths. Others simply ignored the change until the unit in question reached the point where it couldn't move, but this approach leads to behavior that most players find objectionable. A few confessed that they didn't bother trying to fix it — if a unit got stuck, they just jumped it a few squares to get it going again.

Eternal Warriors' THE WAR IN HEAVEN.



Technologies on the Wane

One interesting side discussion that cropped up at the roundtables dealt with AI technologies that developers had played with, but then discarded. Some of these will be familiar, since a year ago there was quite a bit of excitement over the possibilities offered by some of them.

Generally speaking, Artificial Life (A-Life) doesn't seem to have gained much use outside of the realm of RPGs and CREATURES-style games. A-Life is a natural for RPGs in particular, since it gives developers a way to flesh out a game world using NPCs to do all the dozens of dull and mundane jobs that no player wants to do, but which are vital to the gaming experience. A good A-Life AI can make whole hordes of monsters and NPCs behave realistically with very little CPU overhead, which gives the player the feeling of being a part of a living, breathing world.

Last year, a number of developers were exploring different areas using A-Life technologies in everything from first-person shooters to RTS games, but when push came to shove, many ditched those plans in the face of the inherent difficulty of predicting exactly what a given unit would do in a given situation. Developers found, for example, that it really annoyed their producers when they created a 3D shooter level in which a guard was only "usually" at the bottom of the stairs to raise an alarm. Others found themselves wrestling with games in which a unit would ignore the commands given to it by the player — a realistic situation, perhaps, but hardly one the player is happy to be paying for.

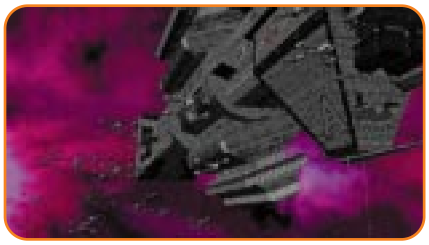
However, some subsets of A-Life technology have found their way into various games. Several of the recent first-person shooters have used flocking algorithms to one degree or another to handle the movement of herds of monsters, birds, fish, and so on. Some RTS games were also making use of flocking variants for group unit movement, and at least one upcoming space combat game (BABYLON 5 from Sierra Studios) plans to make use of flocking algorithms to control the movement of enemy fighter wings and fleets of enemy capital ships.

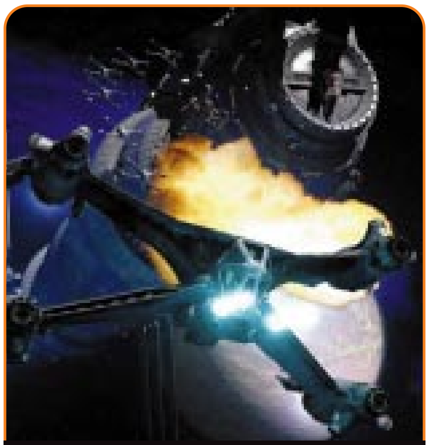
Genetic algorithms (GAs) also haven't found much use in games in the past year. Again, outside of the CREATURES genre (which that game nearly owns entirely unto itself), most attempts by developers to use this technology have fallen flat. The main reason most developers cited was the usual one — too much CPU was being taken up for adaptation and learning that happened at too slow a pace to be useful. After spending several months experimenting with GAs, developers found themselves abandoning the technology in favor of more traditional FSMs and FuSMs. Not only are these more traditional techniques easier to predict and tune, but they demand considerably fewer resources of the CPU.

A few developers did report success in efforts to adapt GAs as tools to aid in tuning their AIs, and they found them easy to adapt to this task. AI tuning is always something of a problem for developers, because by the time a game is near enough to completion to make tuning a meaningful activity, there can be hundreds of parameters that can affect the AI's style of play. Testing every combination is an impossible task, more so given the often tight deadlines looming towards the end of the development cycle. Using GAs to tune an AI lets the developer automate this process, making hundreds of runs of a game using various parameters for the computer opponents. The best variations can be saved out as the basis for the default AIs shipping with the game.

Academia and the Game Industry

One interesting development at the 1999 GDC AI roundtables was the attendance of several members of the research, or academic, AI profession. The primary reason for this was probably the close scheduling of the 1999 American Association for Artificial Intelligence (AAAI) Spring Symposium and the GDC (see the sidebar "AAAI Spring Symposium," p. 42 for more information on the developments at the AAAI conference). This presented an interesting opportunity for many of the theorists in the field to meet some of the engineers.





Flocking in Sierra Studios' BABYLON 5.

Feedback from our academic brethren was fascinating, to say the least. Two guests in one of my roundtables, one a physics major dabbling in AI, and the other a formal AI professor, were adamant that the game industry appears to be light years ahead of academia in producing practical, working AI solutions to some very tough problems. This view was echoed by several others in Dr. John Laird's final-day lecture titled "Developing an Artificial Intelligence Behavior Engine." They greatly admired the game industry's rapid pace of development, noting that more formalized AI studies can often take years to formulate theories of behavior, examine possible solutions, and develop prototypes for testing. Of necessity, the game industry moves much faster (an order of magnitude was how one professor characterized it). The lack of rigorous methodology frustrated our guests somewhat because it makes many of the game industry's solutions unacceptable as support for formal AI studies. Despite this, the academic world was still very interested in studying the solutions game developers have engineered.

AAAI Spring Symposium

Many AI game programmers probably returned home from the Game Developers Conference (GDC) unaware that the next week held another interesting gathering at nearby Stanford University. The American Association of Artificial Intelligence (AAAI) holds both Spring and Fall Symposia, and this year the Spring Symposium (March 22-24) included a session focused on AI in commercial computer games.

Overall, it was an enjoyable experience. The Symposium was small enough that all participants met together in one lecture hall for each session, and attendees from both academia and industry got fairly well acquainted with each other in those two-and-a-half days. There were both lectures and demos, but most of the sessions were panel discussions. The early sessions summarized game AI's past, looking at its successes and failures; sessions in the middle looked at current work. In demos and sessions on NPC design and NPC control, we saw work exploring techniques such as AI control architectures, hierarchical AI, explanation-based representations, pathfinding, natural language interfaces (speech), smart environments, and artificial life. (You can order symposium proceedings at the URL below.) Robotics received a fair amount of focus, which is worth noting by game developers for a couple of reasons. First, game companies may wish to branch into robotic toys (for example, Lego is designing programmable vehicles that kids can tinker with, and its entries at RoboCup soccer tournaments have performed respectably). Second, software techniques and architectures used for mobile robots are often applicable to computer game AI—even low-level movement calculations are useful as game physics simulation gets more realistic.

As interesting as these presentations were, I was even more excited by the discussions about possible future developments in the field of game AI. For example, one discussion session covered AI engines and toolkits, which is a topic of growing interest. In a survey made by one panelist, results revealed the main reason game developers wanted an AI toolkit was to make a better product, rather than reducing production cost or time. Many potential obstacles to toolkit use were given, but the desire to understand the tools was the most common response. Other obstacles brought out in discussion included a suspicion of outside code, a need to know that the technology works, a lack of knowledge of AI fundamentals among developers, potentially large licensing fees, and common demands such as fast speed, low memory, flexibility, availability of source, ease of use, documentation, and support. Desired techniques for toolkits included pathfinding, rule-based expert systems (perhaps with fuzzy logic), finite state machines, inverse kinematics, resource allocation solvers, and perhaps natural language handling.

Two sessions focused on new directions for game AI, and potential killer applications for AI; these discussions were necessarily more speculative. Possible new areas for AI in entertainment included speech and camera input into almost any program or toy (such as a Tamagotchi or Furby, but more creative); genuine give-and-take conversation; intelligent physical interaction in museums or theme parks; artificial life (as CREATURES and PETZ are beginning to explore); real interactive stories; and more personality presence in artificial agents. Other suggestions for killer applications included a "god game" apprentice that could recognize plans and intentions; reliably smart AI for subordinates in strategy games or teammates in action games; variable-skill QUAKE bots; intelligent story development (causality propagation); and "Furby done right." There was some debate on what landmarks could show that AI has arrived (comments included "when AI is mentioned first in game hype," or "when AI is occasionally the lead cover story in magazines"). It was generally agreed that games and toys will be the vehicle to help familiarize and encourage acceptance of AI by the general public.

Perhaps the favorite topic of discussion was how game companies and academic AI researchers can work more closely together. In the opening session, John Laird of the University of Michigan outlined the mutual benefit: AI makes games more fun (a better challenge, more believability, better interaction), and AI helps sell more games; games help AI research by giving great demos, igniting student interest, and providing robust environments to work in and interesting research problems to solve. Further, he said the games community wants academia to provide more information on AI technology, fast, simple (and good) techniques, and more good AI programmers; academia in turn would like case histories of AI development in games, lists of important problems, interfaces to hook AI into real computer games, and funds to support research.

The symposium ended with a discussion of ways to build better bridges between the game companies and academia. Ideas include summer internships for AI students with game companies, reverse internships to send programmers to school for a course or two, a peer-reviewed journal on game AI topics, cheap student rates to the GDC, and college degree programs in interactive/electronic entertainment. It was decided that there would be a similar symposium next year. I enjoyed this year's symposium so much, I hope to attend next year. See you there. *—Bryan Stout*

FOR FURTHER INFO

For a more detailed report on the symposium, check out a longer version of this sidebar at http://www.gamasutra. com
Check out the 1999 AAAI Spring Symposium proceedings at http://www.aaai.org/Press/Reports/reports.html#spring
For information on next year's symposium on AI in interactive entertainment: http://www.cs.nwu.edu/~wolff/AIIE-2000.html
1999 Symposium on AI and computer games: http://www.cs.nwu.edu/~wolff/aicg99/index.html

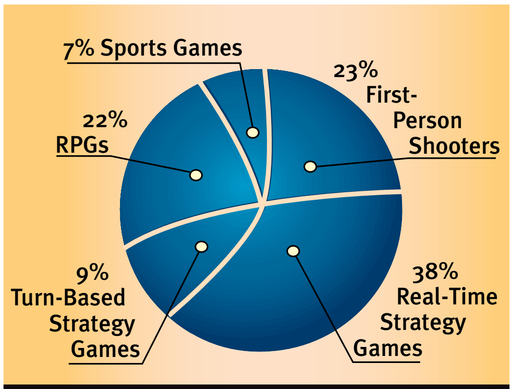


FIGURE 2. *Where do you think the next innovation in game AI will come from?*

Several of the game developers present (including myself) were both flattered and astonished by this interest, since many of us have long looked upon the work being done in the academic realm as "real" AI. Both groups agreed that there were lots of things each could learn from the other, something which I hope this article may help facilitate.

What's Next?

As always at the AI roundtables, I asked my fellow developers for their opinion on a number of questions regarding the future of the industry. Where did developers think game AI was going in the next year or so? Will AI continue to be an important part of game design, or will multiplayers render good game AIs moot? Where did developers feel the next big advance in game AI would come from?

Opinions on these questions were mixed, as one might expect. Any AI developer worth his salt, after all, is pretty darn sure that his or her next game will be the one to contribute something of particular value to the field. Most continued to feel that there would be a slow move away from rigid, rules-based AIs towards more flexible, fuzzy AIs that made use of a variety of technologies in combination with one another. Additionally, as noted above, most developers seemed to think that there would continue to be a move towards opening up the AI to ever-greater levels of user interaction, mostly through a scripting interface of some kind. Everybody was hoping that somebody would manage to put out a game that actually provided programming-level hooks into the AI engine, though nobody at the roundtables volunteered.

Nearly every developer present felt strongly that good game AI would only increase in importance as a part of the finished product, whether multiplayer options were present or not. The reasons for this belief were much the same as they were last year — good game AI will become more of a discriminator as 3D technology levels out, and advances in that area become less spectacular. Learning AIs that can adapt to a given player's style are considered to have big potential, and many developers are concentrating their efforts in that area.

When it came to where developers felt the next big advance in game AI would come from, opinions varied widely across all genres. This was echoed by a poll recently posted on my game AI page, the results of which are shown in Figure 2.

No particular conclusions can be drawn from the above, except perhaps that developers as a group seem to feel that turn-based strategy games and sports games just don't offer much opportunity for advancing the field. I can only speculate as to the reasons behind this, but I would hazard a guess that developers feel there won't be many more turn-based games released in the future, while sports games have a number of restrictions that make AI innovations a bit more difficult (if you get anything wrong, 100,000 angry fans will write the company to let you know).

There is no question that the game AI field continues to be one of the most dynamic and innovative areas of game development. CPU and memory constraints are (slowly) being lifted, freeing developers to experiment with much more interesting and aggressive AI techniques. We're figuring out what works and what doesn't, slowly building suites of tools to speed things along, and just generally getting better at the job. Better and more entertaining games will be the inevitable result. ■

FOR FURTHER INFO

Books: There are precious few books that discuss AI from a gaming perspective. Most are more academic-oriented texts that go into theory more than practice. My favorite comprehensive reference is still *Artificial Intelligence: A Modern Approach* by Stuart J. Russell and Peter Norvig (Prentice Hall, 1995).

In progress:
Author Bryan Stout is working on a book dedicated to game AI due out in early 2000. It's tentatively titled *Adding Intelligence to Computer Games* (Morgan Kaufmann).

Newsgroups: Several Usenet newsgroups focus on artificial intelligence in general and game AI in particular. A few of the better ones in terms of noise-to-content ratio are comp.ai.games, comp.ai, and rec.games.programmer.

Web Sites:
http://www.gamasutra.com
The sister site to *Game Developer* magazine maintains an online roundtable of game AI that grew out of the GDC roundtables. Highly recommended.
http://www.gameai.com
The author's page, dedicated to all things game AI related, provides links to other AI resources and archives of various Usenet threads.
http://hmt.com/cwr/boids.html
Craig Reynolds, known as the "father of flocking," has the best page on the web to start research into the theory and technology behind flocking and similar A-Life technologies.
http://www.geocities.com/ResearchTriangle/Facility/3773
James Swift has written a neat little utility that allows exploration of various 3D navigation algorithms.
http://ai.iit.nrc.ca/ai_point.html
The Artificial Intelligence Resources page maintained by the NRC/CNRC Institute for Information Technology is an excellent starting point for AI research.
http://www-cs-students.stanford.edu/~amitp/gameprog.html
Amit Patel's game programming page, crammed with information on pathfinding algorithms and pointers to other AI resources is a good basic starting point for anything AI-related.

Back

What is a modern computer game made of? It fuses a technical base with a vision for the player's experience. All of the disciplines involved (design, art, audio, levels, code, and so on) work together to achieve this synthesis.

F mal Ab t act De ig T l

By Doug Church

In most disciplines, industry evolution is obvious: The machines we play on are far more powerful, screens have better resolution and more colors, paint and modeling tools are more sophisticated, audio processing is faster, and sound cards are more capable. Technical issues not even in our vocabulary ten years ago are solved and research continues with essentially infinite headroom. The technical base on which games stand (game code and content creation tools) is evolving.

Across all genres and companies, we build on our own and others' past ideas to expand technical limits, learn new techniques and explore possibilities. Ignoring an anomaly or two, no single company or team would be where it is now had it been forced to work in a vacuum.

Design, on the other hand, is the least understood aspect of computer game creation. It actualizes the vision, putting art, code, levels, and sound together into what players experience, minute to minute. Clever code, beautiful art, and stunning levels don't help if they're never encountered. Design tasks determine player goals and pacing. The design is the game; without it you would have a CD full of data, but no experience.

Sadly, design is also the aspect that has had the most trouble evolving. Not enough is done to build on past discoveries, share concepts behind successes, and apply lessons learned in one domain or genre to another. Within genres (and certainly within specific design teams), particular lines have evolved significantly. But design evolution still lags far behind the evolution of overall game technology.

How Do We Talk About Games?

The primary inhibitor of design evolution is the lack of a common design vocabulary. Most professional disciplines have a fairly evolved language for discussion. Athletes know the language of their sport and of general physical

Doug Church values beta testers heavily. After a beta test of this article, he learned half the testers found the first two pages slow reading. If you're in that half, skip to the third page, read to the end, then read the intro. Hey, it's an interactive, multipath article.

conditioning, engineers know the technical jargon of their field, doctors know Latin names for body parts and how to scribble illegible prescriptions. In contrast, game designers can discuss "fun" or "not fun," but often the analysis stops there. Whether or not a game is fun is a good place to start understanding, but as designers, our job demands we go deeper.

We should be able to play a side-scrolling shooter on a Game Boy, figure out one cool aspect of it, and apply that idea to the 3D simulation we're building. Or take a game we'd love if it weren't for one annoying part, understand why that part is annoying, and make sure we don't make a similar mistake in our own games. If we reach this understanding, evolution of design across all genres will accelerate. But understanding requires that designers be able to communicate precisely and effectively with one another. In short, we need a shared language of game design.

A Language Without Borders

Our industry produces a wide variety of titles across a range of platforms for equally varied audiences. Any language we develop has to acknowledge this breadth and get at the common elements beneath seemingly disparate genres and products. We need to be able to put our lessons, innovations, and mistakes into a form we can all look at, remember, and benefit from.

A design vocabulary would allow us to do just that, as we could talk about the underlying components of a game. Instead of just saying, "That was fun," or "I don't know, that wasn't much fun," we could dissect a game into its components, and attempt to understand how these parts balance and fit together. A precise vocabulary would improve our understanding of and facility with game creation.

This is something we already do naturally with many technical innovations, since they are often much easier to isolate within a product or transfer to another project. A texture mapper or motion capture system is easily encapsulated. When everyone at the office gathers around some newly released game, major technical "evaluation" is done in the first five minutes: "Wow, nice texture mapping," or "Those figures rock" or "Still don't have a sub-pixel accurate mapper? What is their problem?" or "Man, we have to steal that special effect." But when the crowd disperses, few observations have been made as to what sorts of design leaps were in evidence and, more importantly, what worked and what didn't.

Design is hard to point at directly on a screen. Because of this, its evolutionary path is often stagnant. Within a given genre, design evolution often occurs through refinement. This year's real-time strategy (RTS) games clearly built on last year's RTS games. And that will continue, because design vocabulary today is essentially specific to individual games or genres. You can talk about balancing each race's unit costs, or unit count versus power trade-offs. But we would be hard pressed to show many examples of how innovations in RTS games have helped role-playing games (RPGs) get better. In fact, we might have a hard time describing what could be shared.

These concerns lead to the conclusion that a shared design vocabulary could be very useful. The notion of "Formal Abstract Design Tools" (or FADT, as they'll be referred to from here on) is an attempt to create a framework for such a vocabulary and a way of going about the process of building it.

Examining the phrase, we have: "formal," implying precise definition and the ability to explain it to someone else; "abstract," to emphasize the focus on underlying ideas, not specific genre constructs; "design," as in, well, we're designers; and "tools," since they'll form the common vocabulary we want to create.

"Design" and "tools" are both largely self-explanatory. However, some examples may help clarify "formal" and "abstract." For instance, claiming that "cool stuff" qualifies as a FADT violates the need for formality, since "cool" is not a precise word one can explain concretely — various people are likely to interpret it very differently. On the other hand, "player reward" is well defined and explainable, and thus works. Similarly, a "+2 Giant Slaying Sword" in an RPG is not abstract, but rather an element of one particular game. It doesn't qualify as a FADT because it isn't abstract. The general notion that a magic sword is based on — a mechanic for delivering more powerful equipment to the player — is, however, a good example of a FADT, so the idea of a "player power-up curve" might meet the definition above.

Let's Create a Design Vocabulary — What Could Possibly Go Wrong?

Before we start investigating tools in more detail and actually look at examples, some cautionary words. Abstract tools are not bricks to build a game out of. You don't build a house out of tools; you build it *with* tools. Games are the same way. Having a good "player power-up curve" won't make a game good. FADT are not magic ingredients you add and season to taste. You do not go into a product proposal meeting saying "this game is all about player power-up curves." As a designer, you still have to figure out what is fun, what your game is about, and what vision and goals you bring to it.

But a design vocabulary is our tool kit to pick apart games and take the parts which resonate with us to realize our own game vision, or refine how our own games work. Once you have thought out your design, you can investigate whether a given tool is used by your game already. If it is, are you using it well, or is it extraneous? If it isn't used, should it be, or is the tool not relevant for your game? Not every construction project needs a circular power saw (sadly), and every game doesn't need every tool. Using the right tools will help get the shape you want, the strength you need, and the right style.

Similarly, tools don't always work well together — sometimes they conflict. The goal isn't to always use every tool in every game. You can use an individual tool in different ways, and a given tool might just sit in a toolbox waiting to see if it is needed. You, the designer, wield the tools to make what you want — don't let them run the show.

Tools Would Be Useful — Where Do We Find Them?

So we need a design vocabulary, a set of tools underlying game design practice. There is no correct or official method to identify them. One easy way to start looking is to take a good game and describe concretely some of the things that work well. Then, from concrete examples of real game elements, we can attempt to abstract and formalize a few key aspects and maybe find ourselves a few tools.

There isn't enough space here to exhaustively analyze each tool or game — the goal here is to give an overview of the ideas behind and uses of FADT, not a complete view of everything. With that in mind, we'll start with a quick tour of some games, tools, and ideas. Since we are looking for examples of good game design, we'll start by examining MARIO 64. Once we have explored some concrete aspects of the game itself, we'll step back and start looking for things to abstract and formalize that we can apply to other genres and titles.

MARIO 64 Game Play

MARIO 64 blends (apparent) open-ended exploration with continual and clear direction along most paths. Players always have lots to do but are given a lot of choice about which parts of the world they work on and which extra stars they go for. The game also avoids a lot of the super-linear, what's-on-the-next-screen feel of side-scrolling games and gives players a sense of control. In MARIO, players spend most of their time deciding what they want to do next, not trying to get unstuck, or finding something to do.

A major decision made in the design was to have multiple goals in each of the worlds. The first time players arrive in a world, they mostly explore the paths and directions available. Often the first star (typically the easiest to get in each world) has been set up to encourage players to see most of the area. So even while getting that first star, players often see things they know they will need to use in a later trip. They notice inaccessible red coins, hat boxes, strange contraptions, and so on, while they work on the early goals in a world. When they return to that world for later goals, players already know their way around and have in their heads some idea about how their goals might be achieved, since they have already visited the world and seen many of its elements.

MARIO's worlds are also fairly consistent and predictable (if at times a bit odd). Players are given a small, simple set of controls, which work at all times. Simple though the controls are, they are very expressive, allowing rich interaction through simple movement and a small selection of jumping moves. The controls always work (in that you can always perform each action) and players know what to expect from them (for example, a triple jump goes a certain distance, a hip drop may defeat opponents). Power-ups are introduced slowly, and are used consistently throughout (for example, metal Mario can always walk under water).

These simple, consistent controls, coupled with the very predictable physics (accurate for a MARIO world), allow players to make good guesses about what will happen should they try something. Monsters and environments increase in complexity, but new and special elements are introduced slowly and usually build on an existing interaction principle. This makes game situations very discernable — it's easy for the players to plan for action. If players see a high ledge, a monster across the way, or a chest under water, they can start thinking about how they want to approach it.

This allows players to engage in a pretty sophisticated planning process. They have been presented (usually implicitly) with knowledge of how the world works, how they can move and interact with it, and what obstacle they must overcome. Then, often subconsciously, they evolve a plan for getting to where they want to go. While playing, players make thousands of these little plans, some of which work and some of which don't. The key is that when the plan doesn't succeed, players understand why. The world is so consistent that it's immediately obvious why a plan didn't work. This chasm requires a triple jump, not a standing jump; maybe there was more ice than the player thought; maybe the monster moves just a bit too fast. But players get to make a plan, try it out, and see the results as the game reacts. And since that reaction made sense, they can, if needed, make another plan using the information learned during the first attempt.

This involves players in the game, since they have some control over what they want to do and how they want to do it. Players rarely feel cheated, or like they wanted to try something the game didn't support. By offering a very limited set of actions, but supporting them completely, the world is made real for players. No one who plays MARIO complains that they want to hollow out a cave and make a fire and cook fish, but cannot. The world is very simple and consistent. If something exists in the world, you can use it.

Great! But I'm Not Writing MARIO 64. I Mean, It's Already Been Written.

So MARIO has some cool game design decisions. In the context of MARIO itself, we have examined briefly how they work together, what impact they have on the players' experience and how these design decisions, in general, push the player toward deeper involvement in the game world. But if you're developing a car-racing game, you can't just add a hip-drop and hope it will work as well as it does in MARIO. So, it's time to start abstracting out some tools and defining them well enough to apply them to other games.

Looking back at the MARIO example, what tools can we derive from these specific observations? First, we see there are many ways in which players are encouraged to form their own goals and act on them. The key is that players know what to expect from the world and thus are made to feel in control of the situation. Goals and control can be provided and created at multiple scales, from quick, low-level goals such as "get over the bridge in front of you" to long-term, higher-level goals such as "get all the red coins in the world." Often players work on several goals, at different levels, and on different time scales.

This process of accumulating goals, understanding the world, making a plan and then acting on it, is a powerful means to get the player invested and involved. We'll call this "intention," as it is, in essence, allowing and encouraging players to do things intentionally. Intention can operate at each level, from a quick plan to cross a river to a multi-step plan to solve a huge mystery. This is our first FADT.

INTENTION: *Making an implementable plan of one's own creation in response to the current situation in the game world and one's understanding of the game play options.*

The simplicity and solidity of MARIO's world makes players feel more connected to, or responsible for, their actions. In particular, when players attempt to do something and it goes wrong, they are likely to realize why it went wrong. This leads to another tool, "perceivable consequence." The key is that not only did the game react to the player; its reaction was also apparent. When I make the jump, it either works or it doesn't. MARIO uses this tool extensively at a low level (crossing a river, avoiding a rolling boulder, and so on). Any action you undertake results in direct, visible feedback.

PERCEIVABLE CONSEQUENCE: *A clear reaction from the game world to the action of the player.*

We have examined the ideas behind some parts of MARIO and abstracted out two potential design tools. Note also how MARIO uses these tools in conjunction; as players create and undertake a plan, they then see the results of the plan, and know (or can intuit) why these results occurred. The elements discussed are certainly not the only cool parts of MARIO, nor the only tools that MARIO uses, but hopefully this discussion gives an idea of how the process works. Later, we'll return to examine how multiple tools work with each other. But first, let's see if intention and perceivable consequence can be applied to some other games.

Same Tools, Different Games

Perceived consequence is a tool often used in RPGs, usually with plot or character development. A plot event will happen, in which the game (through characters or narration) essentially comes out and says, "Because of X, Y has happened." This is clearly a fairly pure form of perceived consequence.

Often, however, RPGs are less direct about consequence. For example, the player may decide to stay the night at an inn, and the next morning he may be ambushed. Now, it may be that the designers built this in the code or design of the game. ("We don't want people staying in town too much, so if they start staying at the inn too often, let's ambush them.") However, that causality is not perceivable by the player. While it may be an actual consequence, to the player it appears random.

The story unfolds in FINAL FANTASY VIII.

There are also cases where the consequence is perceivable, but something still seems wrong. Perhaps there's a fork in the road, where players must choose a direction. As a player travels down the chosen path, an encounter with bandits occurs, and the bandit leader proclaims, "You have entered the valley of my people; face my wrath." This is clearly a consequence, but not of a decision players thought they were making. Players bemoan situations where they are forced into a consequence by the designers, where they are going along playing a game and suddenly are told, "You had no way of knowing, but doing thing X results in horrible thing Z."

Here we can look at how MARIO uses the perceivable consequence tool in order to gain some insight into how to make it work for us without frustrating players. In MARIO, consequences are usually the direct result of a player decision. Rarely do players following a path through the game suddenly find themselves in a situation where the game basically says, "Ha ha, you had no way of knowing, but you should have gone left," or "Dead end! Now you get crushed." Instead, they see they can try a dangerous jump or a long roundabout path or maybe a fight. And if it goes wrong, they understand why.

So it should come as no surprise that, in RPGs, often the best uses of consequence come when they are attached to intentional actions. Being given a real choice to do the evil wizard's bidding or resist and face the consequences has both intention and consequence. And when these tools work together, players are left feeling in control and responsible for whatever happens. However, being told "Now you *must* do the evil wizard's bidding" by the designer, and then being told, "As you did the evil wizard's bidding, the following horrible consequences have occurred," is far less involving for the player. So while both examples literally have perceived consequence, they don't cause the same reactions in the player.

Same Game, Different Tool

Of course, there are reasons why RPGs often force players into a given situation, even at the cost of removing some of the player's feeling of control. The usual reason is to give the designer greater control of the narrative flow of the game. It is clear that "story" is another abstract tool, used in various ways across all game styles in our industry. But it's important to remember that, although books tell stories, when we say "story" is an abstract tool in game design, we don't necessarily mean expository, prewritten text. In our field, "story" really refers to any narrative thread that is continued throughout the game.

The most obvious uses of story in computer and video games can be found in adventure-game plot lines. In this game category, the story has been written in advance by designers, and players have it revealed to them through interactions with characters, objects, and the world. While we often try to set up things to give players a sense of control, all players end up with the same plot.

But story comes into play in NBA LIVE, too. There, the story is what happens in the game. Maybe it ends up in overtime for a last-second three-pointer by a star player who hasn't been hitting his shots; maybe it is a total blowout from the beginning and at the end the user gets to put in the benchwarmers for their moment of glory. In either case, the player's actions during play created the story. Clearly, the story in basketball is less involved than that of most RPGs, but on the other hand it is a story that is the player's — not the designer's — to control. And as franchise and season modes are added to sports games and team rivalries and multi-game struggles begin, story takes on a larger role in such games.

STORY: *The narrative thread, whether designer-driven or player-driven, that binds events together and drives the player forward toward completion of the game.*

Using Multiple Tools: Cooperation, Conflict, Confusion

Adventure games often have little intention or perceivable consequence. Players know they will have to go everywhere, pick up everything, talk to everyone, use each thing on each other thing and basically figure out what the designer intended. At the lowest level, there is intention along the lines of, "I bet this object is the one I need," and just enough consequence that players can say, "That worked — the plot is advancing." But there is little overall creation of goals and expression of desires by players. While the player is doing things, it's usually obvious that only a few possibilities (the ones the designers pre-built) work, and that all players must do one of these or fail.

But as we've also seen, this loss of some consequence and most intention comes with a major gain in story. By taking control away from the player in some spaces, the designer is much freer to craft a world full of tuned-up moments in which the designer scripts exactly what will happen. This allows moments that are very powerful for players (moments that often feel as involving as player-directed actions, if not more so). So here is a space where tools conflict, where intention and story are at odds — the more we as designers want to cause particular situations, the less control we can afford to give players.

Once again, tools must be chosen to fit the task. Being aware of what game you want to develop allows you to pick the tools you want and suggests how to use them. You cannot simply start adding more of each tool and expect the game to work.

Concrete Cases of Multiple Tool Use

An interesting variant of the intention versus story conflict is found in traditional SquareSoft console RPGs (for example, the FINAL FANTASY series and CHRONOTRIGGER). These games essentially give each tool its own domain in the game. The plot is usually linear, with maybe a few inconsequential branches. However, character and combat statistics are free-form, complex systems, which have a variety of items, statistics, and combo effects that are under player control. Players must learn about these systems and then manage the items and party members to create and evolve their party.

During exploration of the game world, the plot reveals itself to the player. The designer creates cool moments which are shown to players, in the game, but are not player-driven. Despite little intention in terms of the plot, players are given some control of the pacing as they explore. While exploring, however, players find objects and characters. These discoveries impact the combat aspects of the game. Combat in the game is entirely under the players' control, as they decide what each character does, which abilities and items to use, and handle other details. Thus, players explore the story while combat contains intention and consequence.

SquareSoft games are, essentially, storybooks. But to turn the page, you have to win in combat. And to win in combat, you have to use the characters and items that come up in the story. So the consequences of the story, while completely preset and identical for all players, are presented (usually) right after a very intentional combat sequence. The plot forces you to go and fight your former ally, but you are in complete control of the fight itself.

Rather than trying to use all three tools at once, the designers use intention and consequence in the combat system, and story and consequence in the actual unfolding of the story. So, the designers get to use all the tools they want and tie the usage together in the game. However, they make sure that tools can be strongly utilized when called on. They don't try to put them in places where it would be hard to make them work effectively.

With a bit of a stretch, one can say that sports and fighting games actually mix all three of the tools into one. The story in a game of NHL 99 is the scoring, the missed checks or the penalty shot. While this story is somewhat basic, it's completely owned by the player. Each player makes his or her own decision to go for the win by pulling the goalie, or not. And, most importantly, the decision and resulting action either works or does not, driving the game to a player-driven conclusion. Unlike adventure games, there is no trying to guess what the designer had in mind, no saving and loading the game 20 times until you click on the right object. You go in, you play the game, and it ends.

Similarly, in a fighting game, every controller action is completely consistent and visually represented by the character on-screen. In TEKKEN, when Eddy Gordo does a cartwheel kick, you know what you're going to get. As the player learns moves, this consistency allows planning — intention — and the reliability of the world's reactions makes for perceived consequence. If I watch someone play, I can see how and why they're better than I am, but all players begin the game on equal footing. The learning curve is in figuring out the controls and actions (in that it's player-learning alone that determines skill and ability in the game). The fact that actions have complete intention and consequence allows this.

In sports games, you direct players, select an action, and watch something happen in response to that action, which gives you feedback about what you tried to do. The player does direct the action — a cross-check missed, a slap shot deflected, a pass gone wrong — but one level removed. While watching the action on screen, one sees everything that happens, but can't be sure exactly why it happened. This is because the basis of most sports games is a statistical layer, and thus the same actions with the controller can lead to different results. When you combine the different player ratings with the die-rolling going on behind the scenes, the probabilities make sense, but may not be apparent to the player. The intention is still there, but the perceived consequence is much less immediate. This removal of direct control (and the entire issue of directing action) through a statistical layer, which the player can intuit but not directly see, is often present in RPG combat. Thus, in TEKKEN, I can't say, "Man, bad luck, if only I'd rolled better," or "Yeah, now that I'm a tenth-level ninja, I can do that move," but in NBA LIVE or an RPG, I often do.

Tool-Based Analysis

A fighter has a simple story ("I had just a sliver of health left, but I feinted a kick and then did my triple punch combo — barely finished him off"), but it's the player's story. There is no, "Man, I can't believe I missed that shot," or "Why did I go and do that?" or "How come my check didn't work?" A simple story, backed up by complete intention in a game that provides clear consequences, makes a very powerful experience for the player. So, both fighting games and, with some obfuscation of consequence, sports games attempt to fuse intention and consequence and from that allow the players' actions tell a story. The complete control provided by a fighter may make the game more real to the player, but the larger scale of a sports game may provide more sense of story. Or, it may be that the direct control of the fighter makes for a more personal story, and the large scale of a sports game makes for a more epic story. In either case, neither the fighter approach nor the sports simulation approach to story and intention is right or wrong. Each elicits a different set of reactions from the player. As a designer, you must understand the ramifications of tool usage if you're going to create the experience you intend.

Ahhh, So What?

Tools as a vocabulary for analysis present a way to focus on what player experience the designer wishes to create. In this high-level introduction to FADT, I have focused on intention and perceived consequence, with less attention to story. (And what story is mentioned is slanted toward the player-driven.) This is not because these are the only tools or even the best tools. However, as we start to analyze our designs and the player experience provided by the tools we use, it's vital we try to understand what our medium is good at.

Games are not books; games are not movies. In those media, the tools used (camera placement, cuts, zooms, music cues, switching narrators, and so on) are used to manipulate viewers or readers, to make them feel or react exactly the way the director or author wants them to. I believe the challenge and promise of computer game design is that our most important tools are the ones that involve and empower players to make their own decisions. That is something that allows each player to explore him or herself, which is something our medium is uniquely equipped to do.

So I look to tools to help me understand that aspect of game design and to maximize the player's feeling of involvement and self. But that's because that's the kind of game I want to make. Each designer must choose the game he or she wants to create and use the tools available to craft that experience.

Hopefully, I have presented enough examples of the tools and tool-based analysis process to provide a useful overview. Of course, I only mentioned a few tools, but, as stated previously, this article was not intended to be exhaustive or complete. It's a justification for us to begin to put together a vocabulary. For this to become genuinely useful, we must engage in discussion and analysis to get a set of tools we like and then refine those tools until they are well understood. With that, we can start to do more careful analysis of the stuff we like and don't like in current games and work to improve future ones. And we can talk to each other more about design innovations, not just technical ones.

We will have to invest a lot of time if we're to generate a full list of tools we've used (or should use) in our work. There are resource economies, learning, player power-up curves, punishment/reward and many others to consider. And each tool could have an article written just about it — how it has been used over time, what games use it particularly well or poorly, and different aspects of it. Similarly, it would be great to take a game such as MARIO or WARCRAFT and really deconstruct it, perform as complete an analysis as possible to see if that would be useful. This article is simply a primer to scratch the surface and give examples of this sort of process.

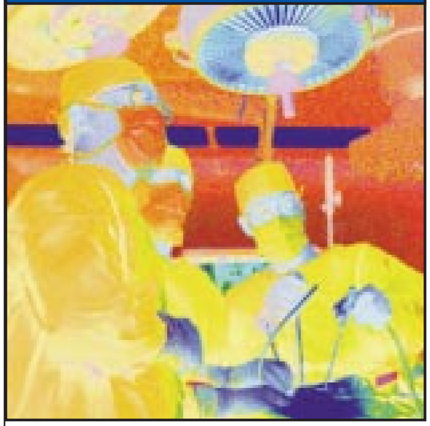
I make no assumption that tools are necessarily useful. Many people may find them overly pedantic. And there's clearly a danger of people starting to use words such as "intention" and "consequence" in the same way that terms and phrases such as "nonlinear," "endless variety," or "hundreds of hours of game play" are used meaninglessly. Not surprisingly, that's not the intent.

FADT offers a potential framework for moving the design discussion forward — no more, no less. Although it's no magic bullet, the hope is for this framework to be broadly useful and allow collaborative analysis and refinement of the game design practice, leading to better designs, more interesting products, and satisfied players. If they're not the right framework, we should figure out why and determine what is the right framework. And then we'll work to evolve and develop it together. ■

The outcome of consequence in FINAL FANTASY VIII.

Back

POSTMORTEM



L ca Lea i g STAR WARS DROIDWORKS

By Jon Blossom
and Collette Michaud

n the fall of 1998, LucasLearning emerged from its shell with the offering of its first educational software product, STAR WARS DROIDWORKS. The game combines first-person shooter game technology with solid educational content to create something different: a thoughtful game that's actually fun and that helps kids learn within the game medium.

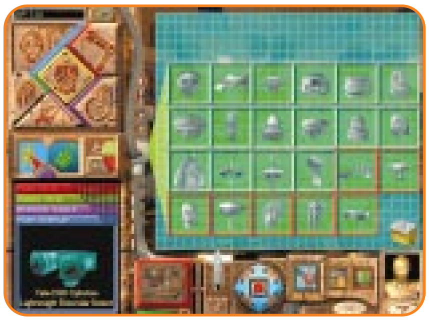
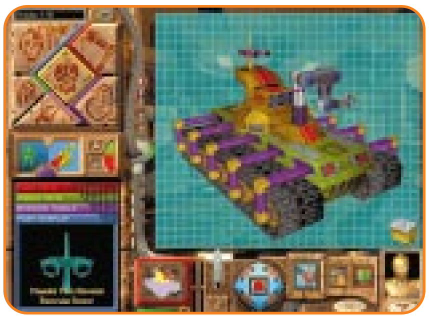
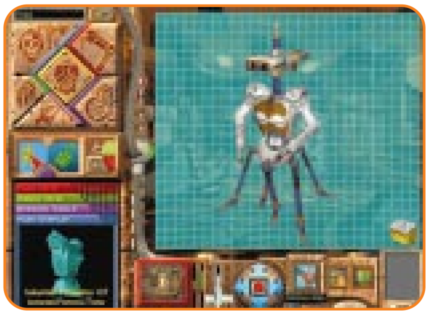
In DROIDWORKS, players take on the role of a rebel spy disguised as a Jawa droid engineer, and are assigned the mission of learning the art of droid building. Players use their skills to solve several physical puzzles, collecting clues that lead them along the path to a secret factory where Jabba the Hutt has been making evil assassin droids for the Empire. To defeat Jabba, players must engineer droids that roll, jump, walk, and run.

In many cases, players need to build droids with special abilities — to move heavy objects, see in the dark, or perform some special task — and players have to explore and apply basic physical principles in order to infiltrate the factory and reprogram the assassin droids.

DROIDWORKS met with rave reviews from family magazines, online educational sites, teachers, kids, parents, and even from many hard-core gaming sites. We did see a number of game players scratching their heads, possibly thinking, "What's the point?" But for the most part, people seemed to love the game for its entertainment and educational value. Eventually, we won several awards including the first-ever award for children's interactive software from the British Academy of Film and Television Arts, the NewMedia Invision gold medal in the children's category, the NewMedia Invision award in the entertainment category (against "pure" entertainment titles including AGE OF EMPIRES, UNREAL, and THE X-FILES), and the 1998 Codie award in the young adult category.

As a company founded to create a new kind of educational software, we knew we couldn't cre-

Project leader Collette Michaud and lead programmer Jon Blossom were two of the first employees at LucasLearning. They designed and built STAR WARS DROIDWORKS with the help of an incredible team. Collette can be reached at collette@lucaslearning.com, Jon at blossom@aya.yale.edu.

ate just another action game. In addition to the unique challenges introduced by an educational bent, DROIDWORKS faced the same technical, artistic, and design hurdles faced by strictly entertainment computer games. We combined gaming styles as diverse as the 3D puzzle games used in TOMB RAIDER, engineering and outfitting games such as TERRA NOVA, adventure games such as MONKEY ISLAND, and role-playing games (RPGs) such as DIABLO, and we took ambitious steps forward to tackle them all at once.

What Went Right

Imagine the scene: A bunch of *Star Wars* fans with overactive social consciences, who have been culled from the computer game industry and the world of education and curriculum development, are dropped into a room and told by George Lucas to "make software that is as educational as it is entertaining, make it better than anything else on the market, and don't spend a lot of money." They're given a blank slate and released from the starting gate, ready to create products that grab kids and parents with the *Star Wars* name, hold them with great game play and production value, and satisfy them with intelligent, thought-provoking content. What could possibly be more right?

1. SIMPLICITY, FREEDOM, AND VISION. Lucas's three-point mandate to the design team was brain-dead simple. His philosophy — and ours — was equally simple: When kids play, they're experimenting, and by experimenting, they're learning. Lucas gave us the freedom to innovate, just as our products would give our users the freedom to experiment and learn. He directed us towards software that would give players the freedom to build, explore, and learn from the experience, one that would mimic Erector sets and Legos. He asked us to allow kids to make mistakes, learn at their own pace, and direct their own learning in a fun and open environment, and then he let us go.

Project leader Collette Michaud started dreaming up product ideas. She had often thought about a *Star Wars* game in which you build different types of droids, and she eventually simplified the concept into one sentence: Give players the opportunity to build any kind of *Star Wars* droid and see it animate. To any computer-savvy *Star Wars* fan, that idea immediately conjured a complete vision, and as soon as anyone heard it, they said, "Of course." Lucas had been pondering a similar idea and he quickly approved it as the product to launch his new company.

Distilling an idea always renders it more potent, and all of the ideas behind DROIDWORKS had been distilled to their very core before we started building. This made it easy to communicate the idea quickly, get people excited, and align them with a central vision that drew on all the good memories they had of their own childhood toys. We could see from the very beginning how cool the game would be, and because the ideas were so concentrated, they never lost potency for the life of the product.

2. KID ADVISORY GROUPS. Usually, computer game designers have the luxury of designing games for ourselves. We're the target audience and we're the judges, so if our game makes us say "wow," it's a good game. For many of us making the transition from adult games to kids' games, we had to realize we were no longer the target audience — we couldn't just build something we thought was cool, we had to build something kids would think was cool. So we enlisted the help of a Kid Advisory Group (KAG), and building that into our production process proved very successful.

We assembled a group of about a dozen kids in the product's target age range to help critique and design the product during the stage when their input could still change the game's evolution. Involving kids early in the process helped us make important design decisions, gauge the level of educational appropriateness, and make sure the game remained fun as we moved through the development process. Seeing the kids react to our ideas and grab onto the game energized and enlightened us month after month, and working with the KAGs proved to be one of the most rewarding and useful aspects of the development process. Their responses, which were always blatantly honest, could throw us into tears of laughter or despair, and their visits always left us with something new to think about.

DROIDWORKS

LucasLearning
San Rafael, Calif.
(415) 444-8800
http://www.lucaslearning.com
Release date: October 1998
Intended platform: Windows 95/98, Macintosh 7.5.5
Project length: 19 months
Team size: 15, including three programmers, four artists, and three level designers
Critical development hardware: Pentium 200MHz 60MB RAM
Critical development software: Form•Z, Softimage, 3D Studio Max, and Adobe Photoshop

The KAGs were more than focus groups. Our kid advisors came back several times and grew emotionally attached to the project as they saw some of their own ideas filtering through the game. They became part of the team, helping the rest of us step back from our adult lives and watch the project through the eyes of our audience. Regular consultation with KAGs from the early stages of an idea has become standard practice at LucasLearning, and each project gets new groups about every six months, retiring one group in order to assemble another with a fresh perspective. The experience we had working with kids on DROIDWORKS made clear to us the importance of timely, appropriate, and constant feedback.

3. SUBJECT MATTER EXPERTS. We also consulted on design issues from the opposite end of the classroom by enlisting Subject Matter Experts, fondly called SMEs. The SMEs are adults with interest, expertise, and experience in the field of study at the core of our games, who were invited to help brainstorm, provide feedback, and check in periodically during the course of the design process. As with the kids' groups, SME groups quickly became a standard part of the LucasLearning product development process.

As game designers, artists, producers, and programmers, we spend most of our days working with computers, piecing together code and artwork, working out budgets and schedules, and keeping the project on track. To create a game, you don't need much else — you can make most of it up, or possibly base it on historical research. However, to create an educational product, you have to check every detail to make sure what you're teaching is correct and well-presented, and, like a teacher, you have to know your audience. We couldn't possibly do this full-time, so we looked to the SMEs for their expertise.

For DROIDWORKS, our original team of SMEs consisted of a variety of science and math teachers, curriculum experts, science museum coordinators, and engineers. We spent a day with them at Skywalker Ranch, tossing around ideas and playing with the DROIDWORKS concept. We continued to talk with them during development, but focused on two participants — one a middle school science teacher from the Sacramento area, and the other a retired science teacher now working at the Exploratorium museum in San Francisco. We continued to meet with them every other month until the end of the project to discuss the status of the game and the educational content we had incorporated into each of the missions. These two consultants helped us immensely in designing the product by pointing out flaws in our physics and showing us how to present things in kid-friendly ways.

4. LUCASARTS, JEDI KNIGHT, AND THE LUCASFILM FAMILY. LucasLearning benefited immensely from its family tree. A small, brand-new educational software company would have had a very difficult time starting up without a powerful license, and our ability to use the *Star Wars* universe was never in doubt. Furthermore, we enjoyed a special relationship as the sister company of LucasArts, and they made many of their resources available to help us jump-start our own efforts. We contracted with their sound, voice, and music departments, used their testers, and took advantage of their cutting-edge game technology (including their proprietary Smush video compressor), and, most significantly, the JEDI KNIGHT 3D engine.

When we first started adding details to the DROIDWORKS concept, interactive 3D graphics seemed an obvious fit, but we weren't sure we could do it. Time and budget constraints meant we didn't have very many resources to devote to core technology, and with all we had planned, starting from scratch would mean cutting several key features we had hoped to include. When we realized we could use the JEDI KNIGHT engine, which was then nearing its beta phase, the doors opened up for us. Not only would we have a real-time 3D engine, we would also have a custom-built level design tool (Leia) and an animation previewer/editor (ModelX), and we'd be able to find artists who had used both — in fact, two of our three level designers had worked on JEDI KNIGHT. Better yet, we'd be able to grab the code immediately after it had gone through years of development and debugging.

Significant work had to be done to adapt the JEDI KNIGHT code, though. Pieces had to be ripped out, twisted around, and somehow welded into a new framework to turn a first-person shooter engine into a creativity tool. Then, a software funnel would have to be fitted to translate all the information from the workshop area of the game into a form that the world simulation engine could handle. The final structure felt like a digital Frankenstein's monster, but the first time we saw a custom droid standing in an empty room in place of Kyle Katarn, the hero of JEDI KNIGHT, we breathed a huge sigh of relief and thought, "Maybe this can work."

DROIDWORKS would have been a very different project without the JEDI KNIGHT engine. We could have tried to build a new simulation engine, using some high-end 3D API such as OpenGL for rendering, but it could easily have ended up taking much longer. We could have created a new level design tool, or hacked something into an existing 3D modeling package, but we

would have had to solve a whole host of other technical issues. We could have written our own scripting language and interpreter, which could be prone to bugs. We could have thrown our hands up in the air and made DROIDWORKS in 2D... and what a different game it would have been.

5. THE VIOLENCE ISSUE. Almost as soon as we settled on creating the game in an interactive 3D environment using technology from a first-person shooter game, our Director of Content, Jane Boston, spearheaded a debate that raged up until the very end of the project. DROIDWORKS exists in the *Star Wars* universe, a universe of explosions, guns, death, and violence. How much of that should we include in DROIDWORKS? What level of violence would be acceptable? How could we create a story about saving the world from evil assassin droids, in a game with the word "wars" in the title, without resorting to violence? And how do we even define what "violence" is?

Eventually, we decided to avoid violence at all costs, to bend the design in any way necessary to avoid scenarios that would otherwise end in violence. There would be no gratuitous explosions, no laser guns, no death (just total loss of power or shields), and the assassin droids would have to get by with only their "shock" value — literally, since they disable you by touching you to drain your power. The gamers among us groaned, but everyone got behind the idea, and, remarkably, it turned out to be one of the better decisions we made.

Implementing the non-violence mandate often proved next to impossible as we drew trick after trick from our game-design tool chests, and then threw away idea after idea when we realized how much violence had become part of our everyday game vocabulary. As game designers, we're used to looking for a fun solution that's easy, so we resorted to standard game vocabulary, such as bad guys hidden around corners or doors guarded by adversaries so difficult that you avoid them until you've completed enough puzzles to gain the power to defeat them. Many times we banged our heads for days trying to find an interesting solution, wishing we could just hand over a laser gun or blow up a bad guy. Imposing the constraint on violence forced us to get creative and pushed us into new territory.

We're very proud to have created missions that require players to think rather than react, and in the end, some of the kid advisors who had complained about the lack of violence and guns thanked us, saying they enjoyed the game more. They described having the time to really use their brains and learn something while having fun, rather than shutting off their brains in order to react to violent situations. Needless to say, parents thanked us, too.

What Went Wrong

Actually, few things went unexpectedly wrong in the development of DROIDWORKS. The team moved together so well that often what seemed to be insurmountable roadblocks from a distance proved to be minor bumps when we actually approached them. Not that we didn't have our share of mind-numbing challenges. We had aimed so high that we couldn't possibly include everything we had hoped for in the beginning, and many features we wanted to include had to be cut. (Where's the print button? And, why can't you plug an arm into the head socket?)

1. JEDI KNIGHT. Ironically, choosing the JEDI KNIGHT engine rather than rolling our own 3D world simulation system proved to be one of the worst problems we had to face, even though it was also one of the best decisions we made. Looking back on the project, we go back and forth trying to decide whether it was the right thing to do. Could we have saved the time and energy we put into working around its limitations by creating a new engine from scratch that would have addressed our needs directly? We'll never know.

In choosing the JEDI KNIGHT engine, we hoped to benefit as much from its physics simulator as from its rendering engine. We wanted our game to teach simple physics in an open-ended environment, and we hoped concepts of velocity, acceleration and gravity would just fall out of the simulator. How wrong could we have been? The level designers closely controlled the world of JEDI KNIGHT, and the physics engine had been tuned to make maximum fun out of running characters and flying projectiles. We had hoped to teach interactive real-world physics, not cartoon game physics in which the main character could run 80 miles an hour, nothing bounced, and nothing could be pushed. We expected a fairly robust dynamics simulation but wound up with sphere-based collision detection in a system that often couldn't respond properly to the collision of two moving objects. The lack of rotational forces made it difficult to build levers and fulcrums.

By the time we discovered how difficult it was to create complex combinations of physical puzzles within the constraints of the existing system, we had no choice but to move forward with the plan. We had to redesign the missions constantly to accommodate the physical quirks of the engine; ultimately, we scripted many key interactions where we hoped the physics engine would "just work." Using heavy scripting by our level designers, and a few custom modifications to the engine, we managed to create missions that worked well within the engine, but unfortunately they fell somewhat short of our original design intentions.

Designing for the JEDI KNIGHT engine posed other problems as well. In a game such as JEDI KNIGHT, almost all significant player interactions with the world have been scripted by the level designers and the animators, in a situation where they know everything about the character ahead of time. The character has been completely modeled and animated ahead of time, and his or her strengths and weaknesses have been determined well in advance. In fact, many physical constants were hard-coded into the engine itself. We felt DROIDWORKS had to give players choices during droid construction that had tangible physical consequences when they took their droid into the game world. "Make the droid matter," we chanted. In JEDI KNIGHT or TOMB RAIDER, level designers know exactly how tall the character is, how far she can jump, how much she can lift, and which things she can pick up. In DROIDWORKS, we couldn't even promise the level designers that the character tackling their worlds would have legs.

Another hurdle we didn't see coming was the JEDI KNIGHT art path. The artists who worked on JEDI KNIGHT had used 3D Studio Max to model and animate their characters, while our artists wanted to use Softimage. The GRIM FANDANGO team had also chosen to use the JEDI KNIGHT engine for rendering, animation, and collision detection, and they had proven an art path from Softimage in concept. What the heck, we thought, it's just data. After outfitting our artists with Softimage and starting the production process, we discovered hidden bugs in the data path that caused glitches in our animations and scaling problems in our models — problems that often required a complete rebuild to fix. We frequently found ourselves manually editing text files containing pitch/yaw/roll values to fix problems introduced during the translation from Softimage.

2. COMPLEXITY: "MAKE THE DROID MATTER." The number of droid combinations possible in DROIDWORKS gives the game its power. How many droids can you build from 87 droid parts, anyway? At one point during development, we could build over 65 million fully functioning droids, but design decisions forced us to scale back to 25 million by the time we shipped. How do you deal with that kind of complexity?

To begin with, we knew we could never test the game completely. It would take one person 290 days working around-the-clock, building one droid a second, just to build them all — let alone test them. A team of ten testers building a droid every minute in a non-stop eight-hour day could do it in 14.25 years if they agreed to work seven-day weeks. As is often the case with simulation games, we knew we would ship the game without complete testing coverage. Our testers accepted that challenge and did a great job covering a huge variety of droid types.

Our level designers also saw the nightmare of complexity. They could never be entirely sure what kind of droid would be walking into their rooms, and they had to leave the levels as open as possible. In many cases, they created ingenious physical filtering mechanisms that would guarantee only certain types of droids went beyond certain points in the level: A steep hill would weed out biped droids in favor of droids with tractor treads, a chasm would weed out wheeled droids who couldn't jump, a narrow canyon would weed out wide droids, and a short door would weed out tall droids. The designers used the terrain leading up to key puzzles to reduce the complexity of the puzzle itself, giving kids a chance to uncover the mission requirements within the context of the game rather than being told, "You can't bring that kind of droid here."

3. POSITIONING. In part because we used the JEDI KNIGHT engine, and in part because we designed it for fun, DROIDWORKS looks more like an entertainment title than a traditional educational product. In our minds, the real beauty of DROIDWORKS lies exactly in that constant tension between entertainment and education. If we hoped to attract the attention of eight- to twelve-year-olds, we felt we had to give them something that could compete for their attention against games such as QUAKE or JEDI KNIGHT. We believed we could combine traditional game-play elements with new and interesting kinds of physical puzzles, creating a game players can think through rather than shoot through, and we pushed hard to steer the company in that direction.

From the beginning, our KAGs gave the game rave reviews, but many adults greeted it with confusion. How could something that felt so game-like be educational enough the be called a learning title? How would we explain it to consumers, who expect to find products either on the game shelf or the education shelf? At some point, reality hit us full in the face, and we had to decide how to position the product and the company. Which shelf should be our home?

We had very little data, other than hunches. Who made the purchasing decisions in this age range? Parents typically look on the education shelves, while teenagers and pre-teens typically look on the game shelves. Teens wouldn't want to play a game they found next to Barney or Barbie, would they? Additionally, we were piggy-backing on the LucasArts sales force in order to get into all the big chain stores. All their distribution channels led straight to the entertainment shelf. In the end, that's where DROIDWORKS landed, too.

Landing in the entertainment category proved problematic for DROIDWORKS. On those crowded shelves, DROIDWORKS disappears among action-oriented shoot-em-up games. Its bright blue box stands out from the dark tones of the adult games and catches the eye, and it feels strangely out of place. Furthermore, retailers have shorter attention spans for products on the entertainment shelves than they do for educational products: If a game doesn't disappear in the first two weeks, the stores send it back. Despite its great reviews and the press coverage we garnered in the month or two prior to its release, DROIDWORKS had a difficult first month of sales, and we started thinking our particular education/entertainment blend might just be too confusing for consumers. Luckily, stores gave it a break — thanks to the *Star Wars* name and the muscle of LucasArts distribution — and by the end of the month, sales had improved and things were looking up.

4. TIMING. DROIDWORKS also had its share of timing problems. We originally planned a fairly luxurious development cycle that had us landing on shelves in time for the 1998 holiday season. About halfway through production, after looking at the numbers, our marketing department decided we should aim for Labor Day instead, which is the official kick-off of the back-to-school buying season. Everyone agreed, so we took a deep breath, reworked major pieces of the design, and made some heavy cuts. The team rallied behind the new date, and we managed to hit our sign-off date.

As the game took final shape, the company's marketing efforts began to kick into full gear. Until that time, LucasLearning had no public presence. Our mission and message had never been projected to the outside world, so we weren't just marketing a product, we were marketing our entire company. We had lived in the cocoon of secrecy that typically surrounds Lucas's endeavors, and the time had come to let the world know what we were up to. That process turned out to be harder — and more time-consuming — than we expected.

While the development team cranked away making the product, the marketing team cranked away designing the box, advertisements, and collateral materials. The entire company watched in frustration as materials were proposed, created, and shot down in their final moments. Labor Day came and went, and no box had been approved. Magazine advertising deadlines came and went, and we watched in horror as the holiday season approached. Finally, an approved package came down, and we landed on shelves in October, with our first magazine ads poised to hit newsstands in December, barely in time for Christmas.

5. GROWING PAINS. Although we had several aces in our hands, including guaranteed private funding and one of the most popular film licenses in the world, LucasLearning was still a startup. We had to hire staff, find space, clarify our vision, define our culture, and adopt processes for doing everything. Even though we had a lot of industry veterans on board, we often ran on hunches, not having time to pause the project in order to figure out the "right" way to do something.

Building a company while building a groundbreaking product is a difficult task. We tripped a few times, and we cobbled together many of our processes from bits and pieces of past experience. It turned out all right, and many of these processes worked well enough to become common practice in the company. Others, however, fell apart when the light of day shone on them after the project, and there are many cases today where we look at what the Droids team did and say, "This is exactly the wrong way to do this."

We all learned from our mistakes: Centralize your asset database, even if you're working with outside contractors (we ended up using Access, FileMaker, and Excel to track various kinds of assets, depending on the format used by our suppliers). Also, don't try to track art production too closely, as collecting, processing, and keeping all that information up to date takes more time than it saves. Keep whatever assets require internationalization in a single database from the beginning, not scattered around in several obscure files that also contain non-internationalized assets. These are just a few of the things we did while we devoted our brains to other urgent tasks.

Success

However it sells or doesn't sell, we consider DROIDWORKS a huge success. We succeeded in creating a product that uses *Star Wars* in an engaging, non-violent way to teach basic physical concepts in an open-ended interactive environment that competes in production value and fun value with its entertainment-oriented peers. We succeeded in creating a product with broad appeal: Players have written us fan mail, the press has praised our effort, and teachers have developed lesson plans around the product. We succeeded in pulling together a great team of talented people, producing a quality game on time and on budget, and launching a brand new company, all at once. ■

Back

SOAPBOX

by Bernd Kreimeier

Killing Games: Violence vs. Censorship

From a German's perspective, the current U.S. discussion about violence in videogames is best described as déjà vu. U.S. game developers have often considered German legislation to curtail the effects of violent videogames on the public inadequate and inconsistent — now that the U.S. Congress is contemplating similar regulations, the industry proves unable to take a stand for itself.

I describe the German law in some detail in an article on Gamasutra.com, and it was my 1998 investigation into this topic that got me invited to climb onto this soapbox. German history holds a number of lessons on violence and censorship alike, and German legislation echoes that history. It strives to prevent exposure of minors to content that might be "ethically disorienting." In addition, criminal law prohibits "glorification of violence," be it in print, on the big screen, or on the little screens of TVs and PCs. These laws have to coexist with Germany's constitution, which ostensibly denies all censorship, and legalistic tap dancing still surrounds this issue today. Ironically, these laws, which were conceived with vivid memories of the Nazis' use of entertainment for propaganda purposes, also remind us of the inevitable contradiction between censorship and freedom of expression.

To make the problem worse, laws are executed by fallible individuals, and the power of censorship is easily abused. (A German public prosecutor's indictment of Art Spiegelman's *Maus* as Nazi propaganda is a recent case in a string of embarrassing examples.) In addition, the means to prevent minors from exposure to "harmful" material (such as the Index, which imposes restrictions on sales and advertising) are rapidly losing effectiveness in the Internet age.

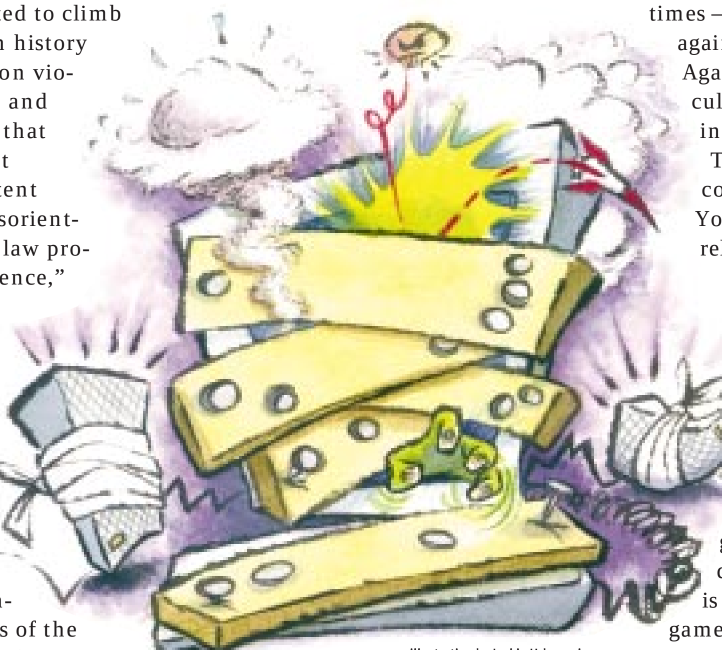


illustration by Jackie Urbanovic

In the U.S., the advent of the Internet gave birth to the Communications Decency Act and its similarly flawed offspring. In Germany, attempts have been made to apply said criminal law to all kinds of location-based entertainment: video arcades in general, laserdromes, paint ball, and online multiplayer games in particular. Unlike the Index, criminal law applies to adults and minors alike. Mere possession of MORTAL KOMBAT or WOLFENSTEIN 3D, the only two games that have been confiscated under this legislation (as opposed to hundreds merely put on the Index), is illegal.

Games never had much ground to stand on in Germany. *Homo ludens*, the "player of games" is considered a liability in a society which favors games about economy, and criticizes literature as an escape from reality. Being a writer of fantasy and science fiction, I have been accused of "escapism" numerous times — a truly German objection against entertainment of any kind. Against this backdrop, it's difficult to make a case for an incriminated game based on its merit. This, however, is the ultimate consequence of rating content: You have to prove your work's relevance and value.

In a way it seems appropriate that the unquestioned pursuit of visual realism has brought the current turmoil upon the industry — media goons thrive on images. Photorealism has no inherent value, neither with respect to art in general, nor for game design in particular — quite the contrary, visual detail is often paid for by sacrificing game play. Yet, the much debated statement "a game is not a simulation" does not seem to be an issue when it comes to anatomical correctness.

Personal preference, however, should never lure one into accepting false accusations and double standards. The influence of game violence on children of all ages has yet to be understood, as the few serious researchers out there would readily admit, placing the ongoing blamestorming on shaky ground. None-

continued on page 63.

Bernd Kreimeier is a writer, physicist, and coder, mystified by the intricate delusions we call reality. Share your personal waking dream at bk@gamers.org.

continued from page 64.

theless, a videogame bill recently proposed in Pennsylvania by State Senator Jack Wagner (D – Pittsburgh) takes aim specifically at games' biggest asset — their interactivity. All of a sudden, active participation in cartoonish violence passes as shooter training, with multiplayer games presumably being the worst.

Some claim that the alleged hypnotic effects of all entertainment are amplified by interactivity. That, if nothing else, should meet vocal opposition from all game developers.

History offers many lessons about the penalty of keeping silent at the wrong time. Every so often, there comes a moment when you have to stand up for what you stand for. Being vocal about the technology and standards you require, while keeping quiet in the face of unfounded public criticism of the art you believe in, is a double standard in itself. In Germany, we've seen it all — publishers abandoning controversial writers, products quietly withdrawn from the market, prejudice leveraged against competition, abuse of the law for personal crusades, and courts judging the quality and relevance of a work of art by weighing it against its alleged offensiveness.

Freedom is often stripped from the individual with the empty promise of public protection. No rating system will ever suffice to appease the driving force behind censorship: the primal urge to protect us, minors and adults alike, against the evil inside. It's not the goal, it's the means that have to be questioned, and we are the ones to do it. Don't expect anybody else to speak on our behalf.

The witch-hunts are not over yet. Ignore the henchmen at your peril. ■

ADVERTISER INDEX

NAME	PAGE	NAME	PAGE
Atari Games Corporation	60	Locomotion Studios	56,57
Black Ops Entertainment Inc.	60	Metrowerks Inc.	8
Boss Game Studios	61	Motion Analysis	C3
Conitec Datensysteme GmbH	63	Multigen	11
Digital Anvil	60	Nichimen	21
Digimation	47	Numerical Design	2
Discreet Monsters	16	Rad Game Tools Inc.	C4
Duck Corporation	13	Resounding Technology	62
Future Light	22,23	Savannah College of Art and Design	62
Hewlett-Packard	15	Savannah College of Art and Design	33
Immersion Corporation	6	Stainless Steel Studios	59
Intel	5	Upstate Games	62
Kinetix	C2,1	Yamaha	18