

gd

GAME DEVELOPER MAGAZINE

NOVEMBER 2001





GAME PLAN

LETTER FROM THE EDITOR

Game Developer

600 Harrison Street, San Francisco, CA 94107 t: 415.947.6000 f: 415.947.6090
www.gdmag.com

Publisher

Jennifer Pahlka jpahlka@cmp.com

EDITORIAL

Editor-In-Chief

Jennifer Olsen jolsen@cmp.com

Managing Editor

Laura Huber lhuber@cmp.com

Feature Editor

Curt Feldman curt99@aol.com

Production Editor

Olga Zundel ozundel@cmp.com

Product Review Editor

Tor Berg tberg@cmp.com

Art Director

Audrey Welch awelch@cmp.com

Editor-At-Large

Chris Hecker checker@d6.com

Contributing Editors

Daniel Huebner dan@gamasutra.com

Jeff Lander jeff@darwin3d.com

Tito Pagan tpagan@wildtangent.com

Advisory Board

Hal Barwood LucasArts

Ellen Guon Beeman Beemania

Andy Gavin Naughty Dog

Joby Otero Luxoflux

Dave Pottinger Ensemble Studios

George Sanger Big Fat Inc.

Harvey Smith Ion Storm

Paul Steed WildTangent

ADVERTISING SALES

Director of Sales & Marketing

Greg Kerwin e: gkerwin@cmp.com t: 415.947.6218

National Sales Manager

Jennifer Orvik e: jorvik@cmp.com t: 415.947.6217

Senior Account Manager, Eastern Region & Europe

Afton Thatcher e: athatcher@cmp.com t: 415.947.6224

Account Manager, Northern California

Susan Kirby e: skirby@cmp.com t: 415.947.6226

Account Manager, Recruitment

Raelene Maiben e: maiben@cmp.com t: 415.947.6225

Account Manager, Western Region, Silicon Valley & Asia

Craig Perreault e: cperreault@cmp.com t: 415.947.6223

Sales Associate

Aaron Murawski e: amurawski@cmp.com t: 415.947.6227

ADVERTISING PRODUCTION

Vice President, Manufacturing

Bill Amstutz

Advertising Production Coordinator

Kevin Chandel

Reprints

Stella Valdez t: 916.983.6971

GAMA NETWORK MARKETING

Senior MarCom Manager

Jennifer McLean

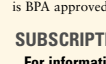
Marketing Coordinator

Scott Lyon

Audience Development Coordinator

Jessica Shultz

CIRCULATION



Game Developer is BPA approved

Group Circulation Director Catherine Flynn
Director of Audience Development Henry Fung
Circulation Manager Ron Escobar
Circulation Assistant Ian Hay
Newsstand Analyst Pam Santoro

SUBSCRIPTION SERVICES

For information, order questions, and address changes
t: 800.250.2429 or 847.647.5928 f: 847.647.5972
e: gamedeveloper@halldata.com

INTERNATIONAL LICENSING INFORMATION

Mario Salinas
t: 650.513.4234 f: 650.513.4482 e: msalinas@cmp.com

CMP MEDIA MANAGEMENT

President & CEO Gary Marshall
Executive Vice President & CFO John Day
President, Business Technology Group Adam K. Marder
President, Specialized Technologies Group Regina Starr Ridley
President, Technology Solutions Group Robert Faletta
President, Electronics Group Steve Weitzner
President, Healthcare Group Vicki Masseria
Senior Vice President, Human Resources & Communications Leah Landro
Senior Vice President, Global Sales & Marketing Bill Howard
Senior Vice President, Business Development Vittoria Borazio
Vice President & General Counsel Sandra Grayson
Vice President, Creative Technologies Philip Chapnick



WWW.GAMANETWORK.COM

Brave Small World

Mobile gaming: the new frontier. Or is it? What do game developers know about these emerging platforms, and what do hardware manufacturers and service carriers for mobile devices know about games, both those who make them and those who play them?

Entering the mobile gaming arena is becoming increasingly attractive to game developers at companies of all sizes and experience levels. The mobile market throws open the doors to millions of new potential game players who would otherwise never play games on a PC or buy a dedicated game console. Nokia sold 128 million mobile phones last year alone; it took Nintendo more than 10 years to sell 100 million Game Boys. Even if only a fraction of the hundreds of millions of mobile-device owners in the world are playing games on their devices, the size of this market and its potential for growth cannot be ignored.

Established game publishers and content providers have taken notice. THQ announced the establishment of its Wireless Division last May, and Sega recently partnered with Synovial to bring Sega Game Gear emulation to Pocket PCs; other big game companies are still flying low on the radar in this arena, but are poised to pounce.

If you're a gadget geek like me, you already own a PDA and a mobile phone, and have made a decision as to what level of service you demand from these devices. Your contact list, calendar, and Sonic the Hedgehog can coexist happily together on your iPaq. But what about that fellow sitting next to you on the subway? What about the woman who's waiting for a cab at the airport? Or the young adults sitting in a bar waiting for their friends? What, if anything, do they want to play?

Game developers have been clamoring to gain access to the true mass market for years now, and bringing games to mobile devices is perhaps more mass market than we've bargained for. Catering to trusted hardcore audiences you've depended on for years won't work, as these are the last people who will be satisfied with playing

games on such a restrictive platform. A fresh approach to game design is the only hope for groundbreaking success on mobile devices and winning support from the most casual of game players, à la the TETRIS revolution of the late 1980s. Someone has to help lead the way.

There have been to date at this early stage in the life of mobile gaming more promises made than promises kept. Everybody seems to have a differing opinion about which revenue models are sustainable, the real potential of next-generation networks and when they will arrive, and how games fit into the broader picture of this large, complicated, and ever-changing global market. Game developers must navigate a tangled web of device manufacturers, service carriers, game publishers, and technology providers to help them realize the importance of providing customers with content that surpasses their expectations for quality and keeps them coming back for more. For whoever succeeds, however, the trailblazing will have been worth it.

Changing of the Guard

Last month, we bade farewell to Mark DeLoura, who headed this magazine for the past year. You know the old saying: You can take the boy out of game development, but you can't take game development out of the boy. So Mark left the general's post here at *Game Developer* and has charged out on to the front lines, from where he'll continue to provide us intelligence as a wily informant.

As for me, it's been my privilege for the past three years at *Game Developer* to help remind our readers on a monthly basis that the world of game development extends far beyond your cubicle or office walls, your current project, your daily pressures, your boss's ego, your own methods and expertise. We hope you'll continue to turn to *Game Developer* each month for both a sizeable helping of technical resources and perhaps even a little moral support along the way.

INDUSTRY WATCH

daniel huebner | THE BUZZ ABOUT THE GAME BIZ



Taldren's new feature. **STAR TREK: STARFLEET COMMAND - ORION PIRATES**, published by Interplay.

Changes at Interplay and Midway.

Interplay's poor second-quarter results will be its last as independent company. The company posted a net loss of \$12.4 million, up from a loss of \$1.9 million in the second quarter one year ago. Revenues slipped to \$14.8 million, down 41 percent from last year's second quarter. Interplay released just four new titles in the second quarter.

Interplay's red ink comes on the heels of an announcement that Titus Interactive would seek to take control of the company. Titus increased its ownership stake in Interplay from 34 percent to 51.5 percent by exchanging 336,070 convertible bonds for over 6 million Interplay shares. Following the move, Interplay announced that it will change the composition of its board of directors and make additions to its senior management team. As a result of negotiations with Titus Interactive, three of Interplay's existing directors have resigned and three new directors nominated by Titus were elected to fill the vacancies. The new board consists of five individuals nominated by Titus and two directors previously nominated by Interplay management.

Midway's financial woes are also forcing changes in the boardroom. The company posted quarterly revenues of \$20.2 million for its fiscal fourth quarter, down from \$24.7 million last year. That figure includes a charge of \$8.9 million related to the company's departure from the coin-op business. The loss for the quarter came to \$30.5 million, compared with a loss of \$30.7 in the same period last year. For the fiscal year ended June 30, revenues hit \$168.2 million, down from a total of \$333.8 million in revenues last year. The loss for the fiscal year reached \$69.3 million. Midway officials attribute the poor results to a transition year, citing its move away from older platforms and arcade products.

Following closing on the poor financial results, Midway announced the resignation of Byron Cook from the position of vice chairman of the board. Cook has been a senior executive with Midway since the company's acquisition of Tradewest in 1994. Midway did not name an immediate replacement.

Take-Two financials. Take-Two Interactive managed higher revenue but lower profits in the third quarter. Net sales for the quarter increased 18 percent over last year to \$84.5 million, benefiting from the success of Remedy's **MAX PAYNE**. Net income was \$409,000, including the retirement of \$1.5 million and a \$651,000 gain on sale of its Jack Of All Games distribution subsidiary. Take-Two reported net income of \$3.4 million in the same period a year ago.



Remedy's **MAX PAYNE**, published by Take-Two's Gathering of Developers.

Jack Sorensen to head up THQ's studios. THQ has tapped Jack Sorensen, who formerly spent six years at the helm of LucasArts, to join the company as executive vice president of worldwide studios. Sorensen will oversee each of THQ's six internal development studios, including Volition, Pacific Coast Power & Light Co., Cedar Ridge Construction, Genetic Anomalies, Heavy Iron Studios, and THQ's Game Boy Advance studio, Helixe. He will also manage THQ's product and studio acquisition activities.

Sierra shuts Dynamix. Sierra has closed TRIBES developer Dynamix as part of a corporate restructuring plan. The move eliminates 97 jobs at the Eugene, Ore., studio. Responsibility for the TRIBES franchise, as well as other Dynamix projects, will be transferred to Sierra's Bellevue, Wash., headquarters.



Sierra has shuttered TRIBES creator Dynamix as part of a corporate restructuring plan.

Changes are also being made in Sierra's corporate structure, with the elimination of 148 non-development positions. The moves are intended to better integrate Sierra into the larger Vivendi Universal Interactive structure. "While any reorganization involving staff reductions is difficult, there is no doubt that this is the right thing to do for the business," said Sierra president Michael Ryder.

Nvidia earnings jump 50 percent.

Nvidia saw its second-quarter earnings jump 50 percent from the same period last year. Based on that result, the company raised its outlook for the coming two years and also announced a 2-for-1 stock split. The company's earnings for the quarter reached \$33.6 million, compared with \$22.5 million for the same period last year. Revenue for the quarter reached \$260.3 million, up 53 percent from \$170.4 million last year. 🐝



UPCOMING EVENTS CALENDAR

MOBILE GAMES SECOND INTERNATIONAL CONVENTION

CONRAD INTERNATIONAL
Dublin, Ireland
November 5-7, 2001
Cost: variable
www.ef-telecoms.co.uk/mgames

DV EXPO

LOS ANGELES CONVENTION CENTER
Los Angeles, Calif.
December 3-7, 2001
Cost: variable
www.dvexpo.com



SN Systems' ProDG 2 for Playstation 2

by jamie fristrom



As I am accustomed to writing games for simple machines such as the PC and Dreamcast, I found the alien architecture of the Playstation 2 to be an inhospitable environment; so many different processors, any of which can fail. Fortunately, we have SN Systems' ProDG development system, which allows us to do source-level debugging on the Emotion Engine (EE), the I/O processor (IOP), or either of the vector units (VUs).

Because we do cross-platform development for both the Xbox and Playstation 2, we need an OS that can develop code for both of them, which means we're stuck with Windows. Thus, we had a limited choice of development systems for the Playstation 2, the most obvious choices being SN Systems or Metrowerks.

After being burned by Metrowerks' Codewarrior when writing code for the Dreamcast — we had to give up and switch to GCC because of the number of bugs the Metrowerks compiler would introduce into our code — we were willing to try anything else for our Playstation 2 development needs. Because our engine was already GCC, we used the development tools that came with our T10000 and used ProDG for debugging. Later we tried SN Systems' tool chain and discovered they had a faster link.

Once we switched, we never wanted to go back. Because GCC supports dual-processor compilation, a single target of our 200,000-line, STL-using engine compiles in around six minutes on our dual 1GHz Pentium II machines, although we had to create metafiles that contained lists of included .CPP files in order to prevent the huge headers from recompiling endlessly. And, because it's GCC, it has optimization options for inlining and unrolling loops, which noticeably sped up our code.

The ProDG debugger does everything a debugger needs to do: watch windows, local variable windows, call stacks, register dumps, and TTY output. It steps into template code without a hitch. Looking at variables declared anywhere in the callstack is no problem.

Because the Playstation 2 is a beast with multiple processors, the ProDG debugger provides source-level debugging for any of the processors you'll be writing code for: the IOP, both VUs, and the EE. It's easy to load up an IOP module, set breakpoints, and step through both main CPU and IOP code. Switching from one processor to another is as easy as pushing a button; you can have panes devoted to different processors open at the same time. You can split and tile windows and panes to your heart's content. The debugger is solid; it crashes less often for me than Visual C++ does.

ProDG supports hardware break-on-read and break-on-write, but the implementation is a little flaky. When we set a hardware breakpoint that didn't break, we could never be sure whether the program was accessing that memory or if the hardware breakpoint failed. However, this failing seems to be due to a limitation of the Playstation 2 — it will occasionally reset its hardware breakpoint registers — rather than a problem on the part of the compiler. Still, this quirkiness could be better documented.

ProDG users can run the target manager and debugger from the command line, which is something I couldn't live without because I like to schedule automated tests. This feature also helps us run a special build of our program to generate some of the content that we ship.

ProDG has some of the slick GUI features that a Visual C++ programmer

might be used to: you can hold the cursor over a variable name to reveal its value, and you can drag addresses between window panes. You can quickly go to any function in your program by entering the function name in the Go To Address box — if you don't remember the exact name of the function, ProDG offers an auto complete feature, which works in the source, disassembly, and memory panes. ProDG's memory pane is better than the one in Visual C++; it always aligns by words and can view your data as floats or even as fixed point.

ProDG provides integration with Visual C++, a feature that we don't actually use on our project but which is used by another team in our company. David Cook, the lead programmer on that project, said, "Overall, I think it's great. You can set up a project with minimal effort, and you can stay in Dev Studio practically all the time. You can set up dependencies between projects, so your libraries get rebuilt automatically as needed. You add new files to the build just by adding them to the project. You can launch the Target Manager or Debugger from Visual Studio, or launch VS edits from the debugger. You can pass breakpoints back and forth. The main drawback is loss of control. You don't have a makefile, so you are limited in what you can customize about the build. Passing in GCC command line arguments is a bit cryptic. And it's difficult to tell what build tools (compiler, linker, and so on) are being used."



ProDG tries to emulate Visual Studio's keys, with some differences (Ctrl-O to open a file in the source window, Ctrl-G goes to an address instead of a line number). It's frustrating that an important operation such as opening a file doesn't have a keyboard shortcut. Changes to your font settings don't affect the current window immediately; you can find yourself with different windows showing differently sized fonts.

ProDG's copy protection is tied to the network card; we've found the easiest way to hand a ProDG license from one programmer to another is to swap the network cards in their machines, which is annoying.

Furthermore, ProDG lacks conditional breakpoints and the ability to evaluate `sizeof()` expressions in the watch window. It can't evaluate functions, either. Because GDB, the Playstation 2 debugger that runs under Linux, can do this (and is free), it seems as though ProDG should have the ability to evaluate functions. Still, I'll admit that such a feature isn't strictly necessary in a debugger. And ProDG does offer a "Set PC to cursor" feature, so if you really want to evaluate a function, you can move the PC there yourself.

One thing that has occasionally made me wistfully wonder what we might be missing if we were using Metrowerks is its performance analysis tools. ProDG does have a profiler, but in the words of John Hall, a coder on my team, "It's barely worth messing with." The information the profiler gave us was that we were spending an awfully long time waiting for the GS to page-flip, which is something we knew already. ProDG's profiler isn't fine-grained enough to tell us why our slow functions are slow.

The important thing about ProDG is that it has rarely given us grief. We have never torn our hair out because a line of source compiled incorrectly. And it has caught some bugs that our previous compilers let slip.

As an added bonus, the employees of SN Systems are cool. When we needed to get a build of the code running at a Sony development conference and were running into operating system hassles, they offered to let us use their machines and even helped us debug our code. That attitude

inspires some developer loyalty. Also, SN Systems is committed to improving its system constantly. For example, when we first started using the ProDG debugger, we found it cumbersome; there was no way to search for text within a source file. That discrepancy has since been fixed, and the company has also optimized the symbol loading in order to minimize the turnaround between making a change and testing it.

We're glad we chose ProDG. We've purchased many licenses for the company, and we also use SN Systems' ProView product to run builds of our game on debug stations. I'm confident that we'll stick with ProDG for the duration of our project and use it for many products to come. ProDG hasn't given us any reason to switch.

Jamie Frstrom is a lead software engineer at Treyarch LLC, currently working on SPIDER-MAN: THE MOVIE for Playstation 2, Gamecube, and Xbox.

ProDG 2 for PS2 ★★★★★

STATS

SN SYSTEMS LTD.

4th Floor - Redcliff Quay, 120 Redcliff Street
Bristol, BS1 6HU
United Kingdom
+44 (0)117 929-9733
www.snsys.com

PRICE

\$5,000 per unit for the first 19 units (prices go down for more), including unlimited technical support.

SYSTEM REQUIREMENTS

486 or higher processor (Pentium II recommended) with Windows 95/98/NT/2000, 16MB of RAM (64MB recommended, 128MB for Windows NT/2000), 300MB hard disk space, CD-ROM drive, 8MB of video memory recommended.

PROS

1. Debugs EE, IOP, or VU code at the source level.
2. Offers GCC-based tool chain with fast link.
3. Generates solid code and the debugger rarely crashes.

CONS

1. Includes a mediocre profiler.
2. Generates flaky hardware breakpoints.
3. Does not support conditional breakpoints.

DIGITAL ELEMENT'S ANIMATEK WORLD BUILDER 3.0 PRO

by tom carroll

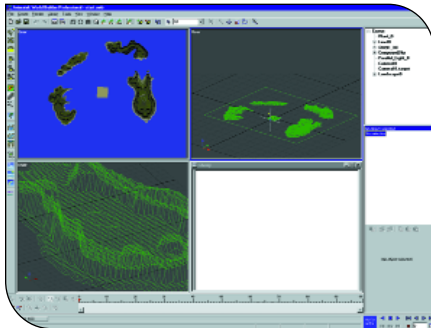
Regardless of what you're doing for your next videogame — knocking out storyboards, populating an island with rocks and trees, or constructing a low-polygon in-game environment — flexible world-building tools are very important. First published in 1995 by Animatek (Digital Element acquired the publishing rights earlier this year), World Builder 3.0 Pro is a comprehensive, high-end package that can be used for modeling, animating, and rendering realistic, fully functional 3D landscapes.

On the surface, Animatek World Builder 3.0 Pro may seem lightweight — the box and contents are light as a feather. Don't let appearances fool you, though. This software is quite powerful. The Professional package includes two CDs: the program CD itself, plus a bonus CD called Plants Thematic Library Disk 1. The library comes in handy right away; let's face it, nothing says landscape better than plants.

Having already used several terrain-building packages (most recently, the editor produced by Planet Moon Studios for the PC version of GIANTS: CITIZEN KABUTO), I was anxious to get started with World Builder. The software proved easy to install on my 850MHz Pentium III desktop machine with 256MB of RAM running Windows NT. It synched up with my copies of 3DS Max 3.1 and Lightwave 6.5, all the better to facilitate data exchange.

World Builder 3.0 Pro's interface is the very familiar quad-screen arrangement, with the lower right corner reserved for listings of textures, objects, scenes, terrains, trees, and such. The right-most portion contains the project manager, where you control the parameters of all the elements in your scene. The tool buttons on the bottom right are used for viewpoint control and are pretty much just like those in 3DS Max. Top and left-side menus complete the interface; I began to appreciate its efficient design as I worked with it.

The first step in developing terrain in World Builder 3.0 Pro is to draw skeleton lines within one of the viewports. Next, you skin the skeleton lines to create a sur-



Animatek World Builder 3.0 Pro employs the familiar quad-screen interface.

face. By adding more bones and increasing the fractal value on the skeleton lines, it's possible to control the overall look of the terrain. The last step is to add in numerous types of surface objects, including grass, trees, compound sky, and clouds.

You can also control animation of surface objects. With a camera operating in OpenGL mode, visual updates are quick enough to make on-the-fly editing quite manageable. Numerous render levels are available, including bounding box, skeleton, wireframe, OpenGL, draft, preview, and production render.

World Builder also includes a 3DS Max plug-in that lets you make changes to models in Max and transfer those changes to the World Builder development bed in real time. You can also share such assets as lights, cameras, and camera paths. I tested this feature and found it to be very useful and an incredible time saver.

Animating objects is refreshingly simple. First, click on the AutoKeyFrame button, then slide the timeframe slider while changing some parameters (changing the camera makes for an easy test). World Builder tweens smoothly from keyframe to keyframe. And nearly every parameter can be keyframed in World Builder 3.0 Pro, enabling rivers to flow and clouds to scud across the sky, all with a great deal of realism.

Because rendering times are longer for complex scenes, World Builder 3.0 is compliant with network and multi-processor rendering. Although I did need to reboot my system a couple of times as system resources hit the ceiling, overall memory management seemed fine. This kind of activity is de rigueur for 3D animators,

though you'll almost certainly want to have 512MB or even 1GB of RAM if you're seriously considering swapping back and forth between World Builder 3.0 and 3DS Max.

World Builder 3.0 Pro is available for Windows 98/NT 4.0/2000 for \$999 (the Standard version is \$399). On September 1, Digital Element began shipping World Builder 3.0 Pro with a plug-in to allow full integration of Curious Labs' Poser scene files, saving further time and trouble.

★★★★★ | Animatek World Builder 3.0

Digital Element | www.digi-element.com

Tom Carroll is a 2D/3D artist who would be quite happy if he could somehow fit 25 (or more) hours in each day. Reach him at jetzep@pacbell.net.

PIXOLOGIC'S ZBRUSH 1.23

by spencer lindsay

Employing aspects of both 2D and 3D applications, Pixologic's ZBrush is one of the fastest modeling, rendering, and texturing programs I have come across. I was able to crank out fully finished 2D artwork in a fraction of the time it would have taken with my usual tool set of Maya, 3DS Max, and Photoshop. However, ZBrush also features an Everest-like learning curve.

Although the majority of the work that is posted on Pixologic's web site is rendered 2D images, the 3D objects created in ZBrush can be exported with textures and coordinates to either .OBJ or .DXF formats (or Pixologic's proprietary .ZTL format). ZBrush also imports .DXF and .OBJ formats, and both worked well in my preliminary tests. A few developers are using this function to modify their existing meshes and textures.

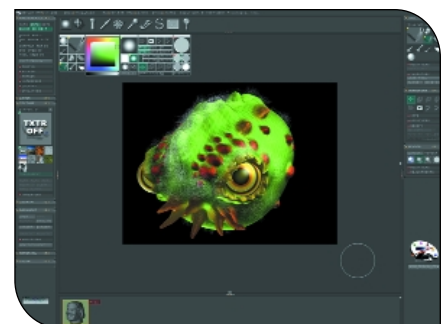
The most difficult aspect of ZBrush is becoming accustomed to the fact that the objects that you create in 3D are temporary. Once you create an object and position it, it's locked onto the canvas in that orientation (scale/rotation/translation) and cannot be moved again. It's as if you've created text in Photoshop and then flattened the text to your image. You can't change fonts once you've flattened the text.

After you've locked the 3D object in its position in ZBrush, you can still apply materials and textures and place other 3D objects on its surface, but the original model itself is no longer a true 3D object. Think of it as modeling something in clay, cutting it in half, and then mounting it on a canvas. The 3D topology still exists, but only from the plane of the canvas towards you. Stay with me, it gets weirder.

The user interface is like nothing I've ever seen before, and it can be pretty confusing for those of us accustomed to the File/Edit/View toolbar menus in the Mac OS and Windows GUIs. There are plenty of places to get lost. For example, in order to texture-map an object with planar coordinates, you must first dig through menus to convert it into a polymesh object, apply the map, and then dig down through several more pull-down menus to find the mapping type.

One of the more useful things about this program is its ability to add 3D components such as lights and materials in the scene and affect the look of the geometry ("pixols") on the canvas. If you're not satisfied with the lizard skin you applied to the original model, simply choose another material, texture, or color and paint it on. Adding and changing lights is intuitive and easy and affects the scene in a predictable way.

One excellent use for this application would be to create and texture a character's head. Starting with a primitive sphere object, you push, pull, scale, and warp the surface with tools that make it feel very much like sculpting with clay. Once you've got your geometry built, you save it into the tools menu and then begin applying texture and material. The texturing is accomplished



With ZBrush and a warped imagination, creating models such as this mutant heart is amazingly fast and functional.

via a complex series of tasks involving saving the mesh's position, painting on it, copying the screen pixels to the texture map, reloading the mesh, repositioning, saving the position information, and then repeating for all angles. This process is quite confusing at first, but once you get the hang of it, it's amazingly fast and functional.

I found the macro-like Zscript tutorials to be very useful once I learned how to slow down the playback. One feature that might be helpful in future releases would be pause and speed controls, so that one could skip to the appropriate sequence in the tutorial. Zscripts are somewhat like MEL or MaxScript in the way that you can, as an artist, record and play back operations that would otherwise be tedious and time consuming.

The support for this product at the Pixologic's ZBrush Central web site (www.pixolator.com) was invaluable. I regularly received three to five replies to my newbie questions within a few minutes. This forum is almost like a chat window; it's that fast. Without this support, the process of learning this tool would have been too much to bear.

If you've got your character or scene sketched out and you know what you want to create, ZBrush will have you cranking out world-class images, models, and textures at a fraction of the time it takes with your traditional tools. Just put aside a few weekends to learn it.

★★★★★ | ZBrush 1.23 | Pixologic
www.pixologic.com

Spencer Lindsay has been an art monkey in 3D and games for over 12 years and is currently looking for a job. Hire him. Reach him at lindsay@etribestudio.com.

GAME DESIGN: THEORY & PRACTICE BY RICHARD ROUSE

reviewed by *damian schubert*

Hey, designers: You know all those snot-nosed high-school students, QA guys, and distant cousins that ask you how to break into the game industry? Richard Rouse has thankfully provided an answer with his solid *Game Design: Theory & Practice*.

The book's chapters can be divided into three categories, each worth discussing in its own right. Possibly the most enjoyable of these contains the six interviews that Rouse conducts with bona fide legends of the gaming industry, including Jordan Mechner, Chris Crawford, Will Wright, Steve Meretsky, Ed Logg, and Sid Meier. All of the interviews are long and in-depth, with detailed descriptions of the early days of the industry and surprisingly candid opinions on where we are now.

These interviews tend to contain more war stories than practical advice, and most are with those from the golden age of gaming, which means that they offer little information on how to make games for today's audiences, with today's larger teams and budgets. Nonetheless, the interviews are highly entertaining, and Rouse's more experienced readers will be wishing that the book had more of them.

In the second part of the book, Rouse breaks down five popular games: MYTH, CENTIPEDE, LOOM, THE SIMS, and TETRIS. These examinations vary in quality, with the chapter on CENTIPEDE being perhaps the single most eye-opening in the book, in terms of design insight. More useful is simply the innate importance that Rouse manages to place upon the examination of completed games as a design tool.

All the remaining chapters of the book discuss, in a very broad sense, the process of coming up with an initial design for a game, from conception to the completion of the design document. This final section also examines certain aspects of game design that require a high degree of designer insight, such as artificial intelligence, level design, and design of the game's tools.

These chapters are a mixed bag. Rouse's chapter on AI provides an extremely strong overview of the real design-side goals of artificial intelligence, all without going into indecipherable technical jargon. Most young designers would benefit greatly from reading this chapter. Even stronger is Rouse's discussion of prototyping and organic design — hundreds of thousands of industry dollars would be saved if more teams approached the initial design of their game in the manner that Rouse suggests. In particular, Rouse counsels his readers to avoid too much detailed design work before the game has reached a certain

threshold of technological advancement.

Similarly, his discussion of play-testing is extremely affirming, and most designers will want to give a copy of this chapter to their bosses. Most notably, Rouse advises against using producers and marketing people as official play-testers, and then gives extremely obvious, logical reasons why this is the case. I remember the words "Preach it, brother!" leaving my lips as I read that passage.

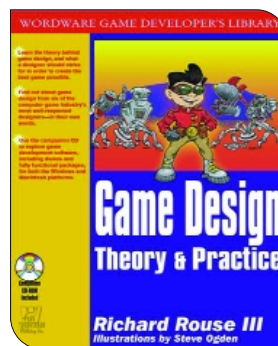
I found my self in sharp disagreement with some of Rouse's assertions. His chapter on storytelling, in particular, often seemed to be in strong opposition with my own opinions on the subject. A matter of ideology? Perhaps. But while I prefer nonlinear games, his blanket dismissal of linear storytelling as a failed and outmoded concept seems to ignore both the fact that many games that have succeeded with linear storytelling, and also that nonlinear storytelling has enormous design challenges of its own. Similarly, his assertion that games with strong, central, player-controlled characters are doomed to inferiority will fall on deaf ears to anyone who has ever played and enjoyed DUKE NUKEM 3D.

Several subjects are surprisingly absent from the book. Rouse fails to include any real, in-depth discussion on player interface design or multiplayer and online play. Even more concerning, Rouse provides no examination or discussion of more modern, collaborative design techniques (such as Use Cases and UML).

Overall, however, the book is very solid. Most experienced designers will learn little from the book, instead finding that it merely encapsulates that which experience has already taught them. Designers who are new to the field, however, will find the book to be a good primer on the art and business of being a professional game designer.

★★★★★ | *Game Design: Theory & Practice*
Wordware Publishing | www.wordware.com

Damian Schubert was the lead designer of MERIDIAN 59 and the late ULTIMA ONLINE 2 project. He currently acts as the creative director of Ninjaneering. He can be reached at damion@ninjaneering.com.



Yoot Saito: SEAMAN Unbound

Yoot Saito's SEAMAN was a genre-breaker when it was released on Dreamcast. Now the title is bound for Playstation 2 — and in Japan, for cell phone users. We talked to Yoot Saito about SEAMAN, his thoughts on wireless gaming, and what it feels like to be the owner of a brand-new puppy — and we don't mean the kind that lives in a chipset.

Game Developer. Congratulations on the news that SEAMAN is coming to Playstation 2.

Yoot Saito. Yes. Thank you.

GD. Will you and the Vivarium team be working on the programming?

YS. We are working with ASCII on the project. Vivarium, my company, will be focusing on the AI, voice recognition, and logic. The ASCII team is developing the graphics part and the Playstation 2 part. The Playstation 2 environment is not easy, but ASCII has already been involved in Playstation 2 development. I think it's a very good collaboration.

GD. Where does your fascination with AI and voice recognition come from?

YS. SEAMAN didn't come from a fascination with voice recognition or AI. I had a passion to realize my original idea — creating a pet that talks back to its owner.

GD. But this isn't your average pet.

YS. My new dog, Ma-ru, is very cute. People are likely to think that if he could speak he would say, "Hi, Yoot, I'm Ma-ru and I'm very happy today." But one day I thought it might say, "What's up, man? I'm not very happy today. Don't talk to me now." The process and the approach we needed was through voice recognition and AI.

GD. One thing is for sure, Seaman ain't cute.

YS. Let me explain. Seaman is not speaking rudely on purpose; rather, he speaks colloquially. He's just casual, speaking like regular people do on the street. Those people are not always rude, just more direct than most other speakers.

GD. Do you sense a lack, or even fear, of creativity in the game space in Japan?

YS. Last night I went to see *Final Fantasy*. The computer graphics were great. It was a CG demo reel. But the script, stories, and background stories were very . . . well, let's just say they didn't compare to the graphics. Currently, game development is getting bigger and bigger. The technical part and the story part, or planning part, are two different components. Fifteen years ago, say, programmers did the music, script, and everything else. But now, game development is getting more like the movie industry. We have so many programmers, graphic designers, and hardware-oriented people, designing and the scriptwriting talents are not as well developed.

GD. What sort of encouragement did you get when you were working on SEAMAN?

YS. When you are starting a new project, you don't want to take big risks, so you are likely to depend on the existing genres. When you create a brand-new genre, you need quite a lot of encourage-

ment to take a risk. Most of the publishers here are public companies, and they can't take such huge risks. They want to follow in the footsteps of the existing genres and stick with FINAL FANTASY or some other existing franchise.

GD. How did you secure the backing for SEAMAN?

YS. We [went to banks and] explained a nonexistent game, and they said they could not loan us the money because the market was very unclear — this sort of game had never existed, and there was no data to consult. It was heartbreaking news, but it was also encouraging. If the bankers had liked it, it would have meant it had already existed.

GD. And then came Sega.

YS. Yes. Based on the contract with Sega, we were able to get loans.

GD. SEAMAN is also going wireless. How will SEAMAN perform in the wireless space?

YS. SEAMAN helps people deliver messages that you can't easily tell. Like confessions.

GD. It sounds kind of . . . dark.

YS. It's not always like that. If you are not encouraged to say "I love you," or "Would you marry me?" or "I think it's time to get divorced" [Seaman can help]. Seaman understands what you want to say, and on behalf of you he speaks about that and gets the answer and brings it back. It helps with communication. Actually, I don't know if it's helpful or disturbing.

GD. How does it work?

YS. With the SEAMAN program, he asks some questions and you tell Seaman how you want your message to be told. Seaman understands what you want to say, goes to the other side, and explains the situation in his own way.

GD. Sounds like Seaman does a little interpretation along the way.

YS. That's right.

GD. How far along in the process are you?

YS. We've done the basic R&D on the PC and we are porting that onto a server so that all the cellular users can use it. Only the cellular program couldn't have voice recognition, so now it's text-based.

GD. Do you view the cell phone as a new gaming console?

YS. You wouldn't believe how many telephone calls I've received, people asking, "Are you interested in mobile games, wireless games?"

GD. Do you think it is an authentic gaming platform?

YS. My point of view is that the wireless platform is a tool, something like a PC. When new software is being developed, people get excited. After the first thousand games, after one or two years, after the honeymoon is over, you'll like games less and focus more business and more utilitarian uses. Game development on that platform will be a part-time job. 🐾



SEAMAN creator Yoot Saito



Falling from the Trees

**Limitless
creativity,
once
impossible,
is now just
a matter of
experience,
time, and
money.**

At this time of year, the days are becoming colder and darker (even in Southern California, where I live). Children are trying to take advantage of the few hours of light by playing in the trees, even while the last few leaves are clinging on. It makes me wonder when we are going to get to the point when an interactive character can actually climb a tree. Sure, BANJO-KAZOOIE had that bear that could shimmy up a tree, and I think AMERICAN MCGEE'S ALICE included a section in which players jumped from branch to branch. But in that game, each branch looked more like a flat concrete ledge with a bark texture on it. I am talking about getting up there, really grabbing a branch, and swinging around from limb to limb. The zoologists call it brachiating, but for games it's just broken.

If our primitive game characters are ever going to swing from the trees, they're going to need to do some seriously quick evolution. They need to evolve much more sophisticated skeletal systems and methods for animation. We still build characters largely for standing on the ground upright (in an evolutionarily ironic twist). However, the "backbone" in most animated characters is a reasonable enough skeleton.

Generally, the artist that designs the character also creates the skeletal hierarchy for that character. Art departments at many companies have standards that dictate how a character should be created. For example, some like to start with the hips. It's also fairly common for people to build skeletons with the skeletal root on the ground, between the character's feet. This approach makes collision detection a bit easier. When this position hits the ground, you know the character's feet are fairly close. Of course, that doesn't mean

that the feet will actually end up on the ground. You see this all the time in games. A 3D character will jump off a ledge onto a flight of stairs. The collision point between the feet is used to determine when the character has hit the ground. But the feet don't know about the ground, so it is not only possible but also pretty likely that one or both of the feet will have passed through a step.

The animation system in most games is kinematic, meaning that it is driven directly by animation data without any physical interaction with the world. Generally, not much is ever done to correct the problems of kinematic animation. Most game developers feel lucky if they can get all of their animation data exported from their animation program and actually working in the game. Keeping the feet from penetrating the ground is often a luxury that is the first thing scratched off the to-do list when the bugs start rolling in.

There have been several games that use a standard, simple system to detect when the character has hit the ground, but then project rays out from the base of the legs to detect where the feet would actually hit the ground. They then can use inverse kinematic methods to place the feet correctly. An even more sophisticated system would calculate the center of mass and make sure that the character is in balance, as I discussed in this column last month ("A Fine Balancing Act," October 2001).

Making It Happen

As you recall from last month, I described a system by which it is possible to calculate the center of mass of an articulated 3D character and then move the feet of the character so that the center of mass is supported by the feet. This



JEFF LANDER | Jeff has decided the whole walking upright thing was a mistake, and he is heading back to trees. Luckily his laptop has a long extension cord and is scratch resistant. Swing on over to his branch at jeffl@darwin3d.com.

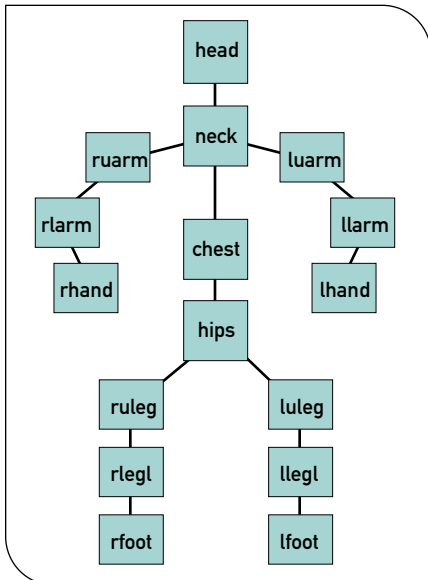


FIGURE 1. The basic character setup.

approach is fine if you have a situation in which you don't mind the feet actually moving in order to support the character. Many times, however, we might prefer to have the character just shift its weight rather than move the feet for balance.

This approach requires that I move the object from the point of view of a fixed support point, meaning that the root of the animation hierarchy must be one of the feet. Last month, I discussed how to implement this method simply by selecting a dominant foot as the root of the animation system. However, this implementation is clearly not a very easy way to animate a character. You would need the dominant foot to position the character, and then every time you moved that foot, the entire character would change position.

Let's take a look at this graphically. I have an articulated character that I want to represent. For simplicity, I'm going to pick a biped. As you can see in Figure 1, the character is composed of objects in a hierarchy that branch out from the hips and chest.

Because all of the body parts radiate from the central point, it's convenient to animate the character by moving the center points first, with all the branches following along. So if I organize the character into a parent-child hierarchy based on the hip as my root point, I get the branching tree structure that you see in Figure 2.

This type of animation setup is commonly used in game production. As I said before, often a root bone exists above the hips in the hierarchy, but generally, the hierarchy in Figure 2 is what you will see for animated characters.

From this structure, you can see why I'm not able to dynamically rebalance my character by simply moving the hip. If the position or orientation of the hip moves, the entire rest of the tree will shift around as well. Thus, the feet will move and need to be repositioned using inverse kinematic constraints to keep them in the same place on the ground.

Now, if I select a dominant foot to be the root of the hierarchy, I get a different tree structure that looks like I grabbed the character by a foot and let it hang there, as you can see in Figure 3.

You can see from this structure that if I were to manipulate the hips, most of the character would still move but the dominant foot that is anchored to the ground would stay in place. The secondary foot would still need to be placed with inverse kinematics, but at least the main support leg wouldn't move.

I could just build my character hierarchy in this form and have my animator simply animate the character this way. But this approach has a few problems. The first is a practical production problem — the animator is probably going to want to strangle me. This approach is not the easiest way to move a character, and most artists really like having the flexibility to set up the animation system the way they want it.

Another problem is that while picking a foot to be dominant will probably work well in some cases, other cases dictate that

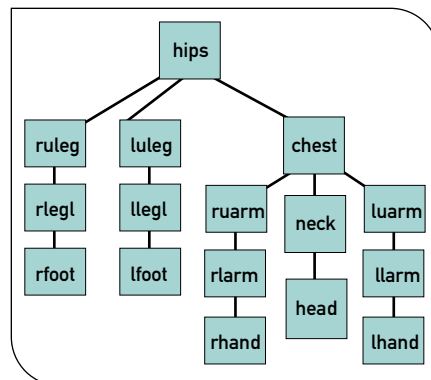


FIGURE 2. The hip as the root.

the selection of dominant foot won't work well at all. For example, you may have a walk cycle where the dominant foot is in the air when the character falls, or a kicking animation where the character leads with that dominant foot. The ideal solution would be to allow the selection of a dominant foot to happen automatically, and then the character hierarchy can be reordered so you can start the balancing process.

Shaking the Tree

So, I need to come up with a way to dynamically change the animation hierarchy so that I can select the root of the system as needed. This kind of task is a pretty common one. I remember that one of my first computer classes in college had similar problems. Given a hierarchical tree structure, rebalance the tree by selecting a new root bone — basic programming 101 stuff. Rebuilding the hierarchy is the easy part; all you need to do is select the new root. Then you walk through, up the hierarchy, making each parent bone a child instead. All the children at each bone remain children. Then, when you reach the current root, you are done.

This approach will get the hierarchy the way I want it. However, because the animation is dependent on inheriting rotations and positions from the character

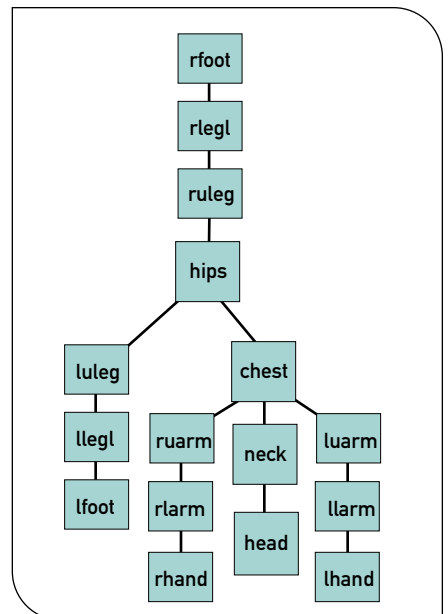


FIGURE 3. The foot as the root.

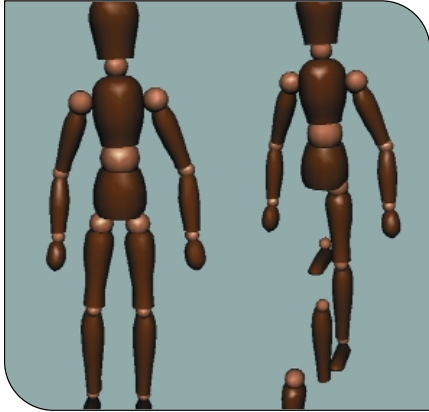


FIGURE 4. The character before (left) and after reparenting.

hierarchy, just rebuilding the tree is not enough. Once everything is reparented, the entire character is completely messed up, as you can see in Figure 4.

So what is the correct solution? I need to recalculate the translations and orientations so that the original position is maintained. I can begin with the root bone. When I select a new root bone, such as the right foot, I know the final position and orientation of the foot by where it sits in world space in the initial tree structure. I create the world transformation matrix for the foot by going through the initial tree and accumulate the rotations and translations into a final world-space matrix. This matrix is assigned to the foot and becomes the new foot matrix.

Then I proceed up the tree, looking at the parent of each bone. So I start with the right lower leg (RLLEG). I need to determine the position for this bone. In order to place RLLEG correctly, I actually have to use the initial values for the right foot (RFOOT). Originally, the right foot was translated down from the root position of RLLEG (the knee) and there was no initial rotation.

To reverse the position of RLLEG, I just need to assign it the inverse of the translation and rotation of the new parent, RFOOT. In this case, I just negate the initial translation. If the bone also had a rotation, I would just invert that rotation as well. In my case, I am using quaternions for the bone orientation, but it would be just as easy to use Euler angles or a matrix and invert them for this step. Degree-of-

freedom restrictions are also reversed in the same way (more on that later).

If any children are encountered along the way, they just remain children. Once I have reached the original root, I am done. I have just established the algorithm for reversing a hierarchy.

1. The new root is assigned the world space orientation and translation of the original position.
2. Each parent of the new bone becomes a child of that bone and inherits the inverse of the original translation and rotation of its new parent.
3. Children of the new bone remain children, untouched.
4. When the original root is reached, I am done.

With this algorithm, I am able to dynamically rebuild the animation hierarchy without affecting the pose. I just select the root that I want as the anchor position and run the routine. I can then use my dynamic balancing system from last month, reset the hierarchy, and blend the balanced pose right in with my regular animation.

In fact, having this algorithm available in my run-time animation system allows for a great deal of flexibility. I am no longer constrained to solving inverse kinematic chains that the artist sets up ahead of time. I can dynamically create and then solve an IK chain that, for example, goes from one of the feet to the opposite hand. A lot of possibilities exist.

It's Always Something

Of course, as with any algorithm, there are some issues. The iterative inverse kinematics systems that I use for coming up with these dynamic poses require some help. I need to define the degree-of-freedom restrictions for each link. This keeps the system from ending up in a position that is not actually possible for the character to achieve in real life.

When the hierarchy of the skeleton is reversed, the degree-of-freedom restrictions need to be reversed as well. This step can lead to some ambiguities. It's possible to have a sequence of bones for which I haven't defined a set of degree-of-freedom restrictions. This problem is particularly apparent where multiple bones come together to a common parent.

Otherwise, the dynamic hierarchy creation is an easy-to-implement and very useful tool. It will certainly become a standard part of character animation systems as more dynamic animation becomes common in interactive applications.

Parting Shot

With a bit of a heavy heart, I have decided to take a break from this column. It has been an amazing experience that I have thoroughly enjoyed since I wrote my first piece for *Game Developer* all the way back in January 1998. Over the years, we have covered a great deal of ground together. I hope you have enjoyed following the exploits of an independent game developer as he struggled to stay on his board and post a good score riding the game technology wave.

I cannot emphasize enough what an amazing time it is to be working in this exciting and dynamic field. We finally have the power to create worlds with limitless creativity. Things that were once impossible are now just a matter of experience, time, and money. Research in cutting-edge game technology has become a hot topic not only at the Game Developers Conference, but also at the big leagues of graphics research, Siggraph, as well as at schools and research facilities around the world.

Through the columns in *Game Developer* as well as articles on Gamasutra.com, game development issues have reached a worldwide audience. I have been stunned by the amount of e-mail I receive from around the globe. People interested in games are working on projects of every level in each corner of the planet. At last check, I have received e-mail from more than 100 countries, from places as varied as Iran, Tonga, Siberia, and Iceland. All the international gamers have humbled me with their amazing knowledge of English, as well as technology. I haven't been able to write back to everyone yet, but I hope to soon.

So, get out there and create. There is a whole lot of work to get done. Just make sure that when you come out with that great new graphics algorithm, you share it with the gaming community. Write an article, give a talk, rant away on Usenet. Great things happen when we all learn from each other. 🙌

RIGGING BEYOND BIPEDS



When you want believable character animation in your game, proper character setup, or “rigging,” is essential, second only to good animation talent. The general consensus among seasoned animators is that a good virtual rig for your virtual puppet is the foundation for good animation. If you currently work as a character animator in today’s frugal game industry, then you are likely to be responsible for rigging your own CG character as well. The challenge of rigging a quadruped or something anatomically more complex than a biped can be difficult using basic animation tools. With a bit of creative thinking, you can go beyond the basic humanoid rig. Having been given the responsibility for finding reliable solutions to such challenges, I’d like to share with you my findings and several proven approaches to some pleasant experiences.

Let me start by defining the term “rigging.” Rigging is the process of outfitting a character mesh with an armature or skeleton hierarchy that enables you to deform the character mesh and thus pose it for animation. The animated skeleton is the underlying structure that drives the movement of geometry, thus creating a virtual character. A successful character setup for a game hero or evil creature takes into account all of the worst-case motion sequences that the real-time character will encounter throughout the game. Your rigged actor should be ideally suited to handle all animation sequences that will be visible from the in-game camera’s point of view.

Unlike our counterparts in the film business, game industry character animators are not as specialized and usually wear several hats during any given project. Shortcuts and plug-ins will often help expedite and streamline an art path while addressing some common issues experienced by production animators. The benefit of personally creating a custom rig for each character before animating it is that

you can go back at any time and address any problems you may have with your original structure for optimal animation performances. Quite often, very subtle yet undesirable behaviors with a rig don’t show up until the animator is well into creating sets of animations. One example is the correct positioning of each rotational joint. You will want an art path for creating CG characters that is flexible enough to allow you to go back and make such changes without losing existing keyframed animation that took some time to create.

Features of Desire

In a production environment, the right tool can take the work out of work. A common and obvious tool of choice for many game companies developing real-time character animation is Discreet’s Character Studio (CS). It has been a staple tool for me on many projects for several years. CS is a plug-in for 3DS Max that provides professional tools for building and animating 3D characters quickly and easily. CS has a broad range of tools that aid in the creation and animation of virtually any type of two-legged character. With one of its two components, conveniently called Biped (the other component is Physique), you can create a basic biped skeleton hierarchy instantly and then alter its structure using Biped’s parametric figure controls. You can add tails and ponytails for animating ears, jaws, or insect antennae. It’s also possible to add modifiers to Biped objects for creating precise bone shapes. In addition, the software uses many built-in 3DS Max features to provide a foundation for character skin modeling, as well as general manipulation

tools for object transformations and keyframe editing.

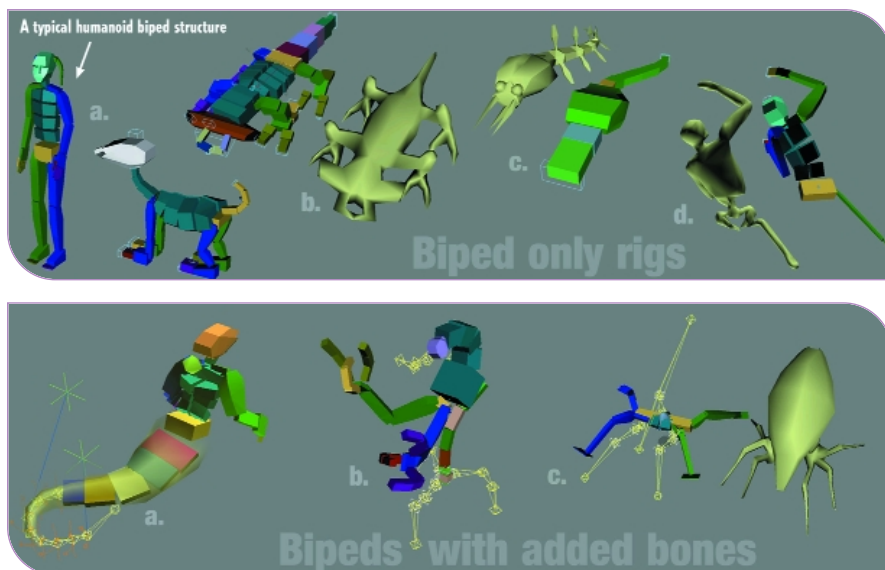
As a time-saving animation aid within CS, you can apply footsteps (Biped’s footstep-driven keyframe animation feature) or free-form animation from one Biped character to any other Biped character, regardless of height, proportion, or even structural differences between them. A simple yet useful feature is Autogrid, which lets you build Biped skeletons with their feet planted on top of other objects. You can also use Autogrid to place footsteps on uneven terrain. Another powerful feature is the Motion Flow Editor, which makes it easy to create long, complex animations by breaking them down into smaller motion clips that you can link together. You do this by stringing together multiple Biped animations (based on .BIP files) using a motion flow graph that resembles a simple flowchart. CS also supports the use of ever-popular motion capture data.

Who You Calling a Biped?

As game environments become more ambitious, the number and variety of characters that inhabit them increases. Many fantasy, science fiction, or even true-to-life game experiences today call for creatures that are not humanoid or bipedal in structure at all. This requirement presents a challenge for any animator that is more comfortable with human animation or software packages or plug-ins (such as CS) that are better geared towards bipedal character motion. Again, with a bit of creative application, animators can use a modified CS Biped to get their rig work done without changing or adding to the existing skeleton hierarchy (Figure 1).



TITO PAGÁN | *Tito is a seasoned 3D artist/animator working at WildTangent and teaching at DigiPen in Seattle. His e-mail address is tpagan@w-link.net, and his web site is www.titopagan.com.*



FIGURES 1A–1D (top). A Character Studio Biped structure will bend and contort to satisfy your needs. FIGURES 2A–2C (bottom). You can augment your skeleton rig with extra bones to accommodate a more complex character design.

Every time I think about what I can do with a CS Biped, I arrive at the same conclusions. With a bit of experimentation, I can create just about any rig I need. When necessary, I also link regular Max bones to a Biped. This extends the use of a Biped structure beyond the default batch of the more than 75 predefined bones it normally provides. But why limit yourself to just this set? For example, I often use my extra bones at the extremities of my skeleton structure and animate them using forward kinematics (Figure 2). This approach makes it easier for me to continue using a Biped, as I am accustomed to doing, and reap the benefits of using all the features that come with this plug-in. By doing so, I can still do things such as save the majority of my Biped motion and reapply it to the same character or any other similar character physiology to save myself production time.

Made for Real Time

As the former lead animator on Gas Powered Games' soon-to-be-released RPG *DUNGEON SIEGE*, I was responsible for rigging and animating many unusual creatures. This PC title is Gas Powered Games' first in development and has all the makings of a robust good-versus-evil real-time RPG. We used CS as well as many of the built-in features in 3DS Max to get the job

done. The development team also called upon the character concept work of traditional artist Joe Kresoja (Figure 3). Besides the typical clichéd bipedal humans for this game genre, Joe dreamed up many weird and unusual beings that really challenged several artists and animators.

These fantastic fictional creatures, which range in status and scale from minions to bosses, were given a wide range of abilities by the game design. Many of their performances required much more than a typical walk cycle. Joe, I, and a few others would model, texture, and rig each of our assigned characters and ultimately animate them using free-form animation. Free-form animation is an option that CS gives you to animate character poses with or without the aid of footsteps. In free-form mode, you set all the keys manually. We had a blast making characters fly, crawl, hop, swim, float, slither, run, die, attack, fidget, taunt, and react to pain. We took turns acting out what our characters would do physically to accomplish their individual performances.

For such projects, I typically animate a game character using 12 frames per second instead of 15, 24, or 30. This produces less keyframe data, thus creating a much smaller file. For real-time game characters, the keyframed rotation and/or position data for each rotational joint of this skeleton is read directly by the game's animation sys-

tem. This underlying rig deforms the mesh that is attached, or weighted, to it. Using as few bones as possible also contributes to smaller file sizes. This optimization is preferred by most real-time game developers, especially those developing web-based games that still require smaller assets for a faster download. Having fewer keyframes in general also makes it much easier to manage and manipulate your keys throughout the motion creation process.

When using a CS Biped, however, you are confined to a minimum set number of skeleton objects you can use in your structure's parameter setting. For example, Figure 2c actually has a Biped Spine, Neck, and Head bone in the middle of the character that I'm not using. Because I can't make them go away by setting their parameter values to 0, I hide them all or scale each one down to make it very small so that it is visually out of my way. I freeze such objects to avoid accidentally selecting them while working with their adjacent bones. I usually avoid weighting any of my vertices to these bones.

I find that with fewer frames to animate an arm wave or a tail swing smoothly, I am challenged with ensuring that I animate each joint rotation with a more linear interpolation between frames. To assist myself in the process, I often display the trajectory for each joint, found in the Display Properties section of the Display panel (Figure 4). Biped offers a similar feature for its bone objects (see the white arrow in Figure 5) but only displays it when that bone is selected. For a smooth and more natural motion, avoid having spikes or too much noise in your trajectory lines. Sometimes your best option is to remove unimportant keys that don't contribute much to the motion path shape or timing of the motion.



FIGURE 3. Joe Kresoja's character concept.

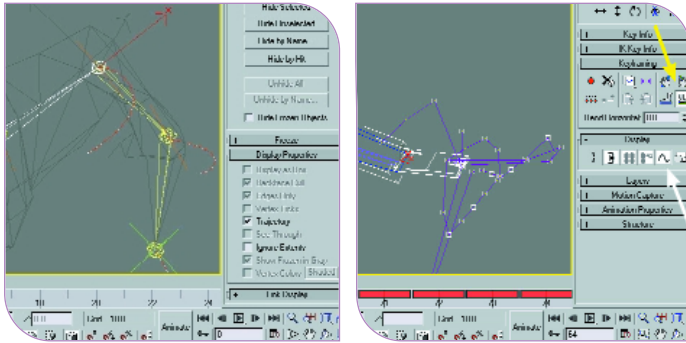


FIGURE 4 (left). Display trajectory. FIGURE 5 (right). Animation aids.

If you would like to see some of these finished character models up close and fully animated, I have posted examples on my web site. I invite you to view them in real time, as they were intended to be seen in a game. I use free-form animation with a combination of forward and inverse kinematics. A Biped supports both.

The hand and foot bones of a Biped are similar to IK handles or end effectors. Whether you need them for your rig or not, having them exposed at all times makes it easier for the animator to grab them and animate the entire limb at once. As an animation aid, these hands and feet can also be locked down in space so that they hold their position through any number of frames (see yellow arrow in Figure 5). You can later “bake” that position in by placing new keys in the frames where you’ve anchored them via the Anchor tools.

The latest version of Character Studio, version 3, has implemented additional features that really improve this process, especially the selectable IK pivots for the feet and hands. It makes free-form animations with IK blends for the feet much more flexible, and also makes animating quadrupeds using a biped very easy and precise. Previous versions of CS had problems with unwanted sliding of feet and feet moving through the surface they were supposed to be walking on. CS3’s improved IK key control makes it easy to fix feet firmly in place.

Are We There Yet?

Setting up a custom rig for a new model requires much trial and error. Getting it to work well isn’t a trivial issue. Don’t be afraid to try something and then abandon it moments later if you find it

isn’t working properly. This means that you must test your new rig as much as you can at every step. I can’t stress this enough. Get into the habit of attaching your mesh to your rig quickly and as often as you can. If you are creating

this character for a real-time game, make certain that your team can provide you with the internal tools, exporter plug-ins, game or animation engine, and so on, to view these new assets in the intended context. Besides the immediate gratification you will enjoy while viewing them, you and your team will appreciate knowing that your art path and character specifications are optimal.

Figure 6 shows a character, modeled and textured by Rick Winter, for which I recently built a rig. Because it is a fairly complex model, I felt I had some options about how I could structure this model using a combination of two Biped or a combination of a modified Biped with a few bones.

I had two things to consider before I could start. I knew that this model was intended for WildTangent’s new web-based game EVILUTION and would be viewed in real time from within a web browser. First, I had to make my rig with as few bones as possible. Then I would need to test its effectiveness by laying down some keys and exporting it using WildTangent’s 3DS Max exporter.

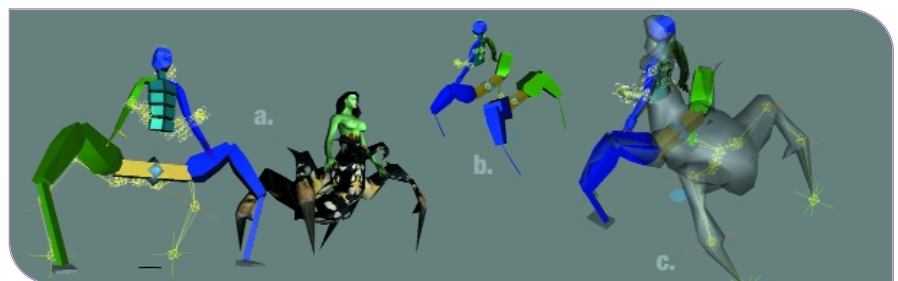
The second thing I had to be certain of was that I could create an atypical rig that

would use only one skin modifier (Physique) to attach the character mesh to the skeleton (the exporter currently doesn’t support using two or more). After a few tests, I abandoned the rig in Figure 6b that uses two Biped linked together with a link controller.

I also knew that this character would be seen on a cliff overhead, so I could get away without having the feet locked down perfectly. I settled for the rig in Figure 6c. Here I have a bone hierarchy attached to the Pelvis of my Biped with two end effectors at the other end. These help to “hold down” the back legs while I bob the character up and down.

The alternative to good planning can be painful in this line of work. Nothing is more frustrating than being given a list of characters to begin cranking out without first knowing certain technical issues, such as whether you are well over budget in polygon count, how many frames you can use per animation sequence, whether your character will eventually hold a weapon or interact with other characters, and so on. These may seem like simple problems, but they do add up quickly in the loss of man-hours that become weekend makeup time.

The revisions involved in this kind of work can put a damper on the development experience. It’s always fun for the first few months until overcoming these technical issues becomes challenging. This is another important reason to use the tools that can help make up for lost time, or at least ease the pain of having to revisit a large cast of character animations over and over again. Such tools can make the process as iterative as possible and easier to troubleshoot. 🐛



FIGURES 6A–6C. There is always more than one approach you can take. Make the time to find the right one.

It's a Wireless World



At a time when the game industry stands at the threshold of a long-awaited hardware transition that will carry us forward into next few years, why take this opportunity to address the still relatively nebulous world of mobile gaming?

Most game developers have come to rely on the trusty tide of Moore's law to drive PC hardware advancement, and on console manufacturers to give their hardware a respectable life cycle in the market in order to recoup as much of their hardware investments as possible. Compared to the traditional game market, there are as yet relatively few known quantities in the rapidly developing mobile game space.

Still, the market has matured just enough at this point that it is possible to shed a reasonably accurate light on the state of the industry from the perspective of those operating on the front lines of mobile game development. These people

have heard all the analysts' and big mobile companies' lofty statistics and assurances of astronomical global growth, the magnitude of which hearken back to the early promises made about the revenue potential of e-commerce or online gaming or widespread broadband adaptation — and we know where those stand today. But, as developers in this space have discovered, mobile gaming presents some unique opportunities unlike any we've experienced before.

Whether you're currently developing games for PCs or consoles, or have already sallied forth into the mobile game development wilderness, the following pages present a circumspect view of the current state of mobile game development around the world, from current and next-generation technologies, to the reinvention of game design, to working and nonworking business models, to the promises and lessons that the mobile gaming markets in Europe and Asia present. The growth of mobile gaming will ultimately affect the rest of the traditional game market, whether you're ready for it to or not.

WIRELESS GAMING

- 30** Wireless Game Development: Coding Without a Net
- 34** Wireless Game Content: How to Think Big In A Small World
- 36** The Business of Mobile Games
- 38** The Wireless Scene in Europe
- 40** Japan Gets Hooked on Java



Wireless Game Development: Coding Without a Net



Developing for new platforms is business as usual in the electronic gaming industry, but in today's relatively mature marketplace, the number of platforms and standards to develop for is quite small, and new devices appear with a relatively low frequency.

The infant field of game development for wireless devices is a different story. As with any up-and-coming industry, there are many competitors hoping to set the new standards. Even within given platforms and SDKs, there are numerous variations in everything from screen resolution to control-key layouts.

The transition from developing titles for PCs and consoles to developing for wireless and mobile devices is a challenging one; it is also rewarding. At NuvoStudios, we have developed (or are in the process of developing) titles for a number of these new platforms. By the time you read this, we should have 12 titles for Binary Runtime for Wireless (BREW), 8 titles for J2ME, 7 titles for Palm OS, and 10 for Windows CE complete and in the marketplace.

Our primary experience in wireless devices has been with mobile phones, principally with the J2ME and BREW environ-

ments. However, and although most of the examples herein relate to our mobile phone game development, observations about Palm and Windows CE are included throughout.

The Control System

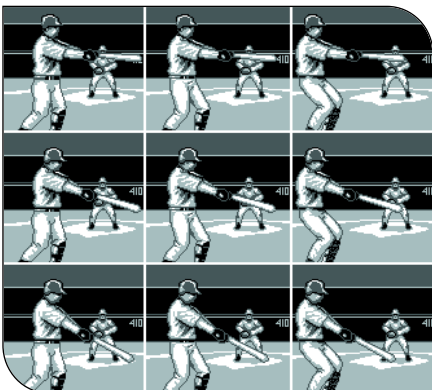
Developers of console games in particular know that control systems are not a strange place to begin discussing development. The controls are of such priority that it's vitally important to figure out how the user might operate the game before even considering the technical feasibility of executing the title. After all, if there's no way to control the game, there's no point in doing it.

For example, while a mobile phone might at first glance appear to have more possibilities as an input device than even a dedicated game controller due to its large number of buttons, these keys are obviously not designed or laid out like a joystick or other game controller. The buttons are often recessed to prevent accidental presses and are typically clustered very tightly together. Furthermore, while many recent models have joysticklike directional buttons, these are often small and imprecise.

When we developed a suite of solitaire games for the BREW-enabled Kyocera

QCP 3035 phone, the clumsy soft keys made a PC-like point-and-click cursoring system difficult, and the card layouts precluded using the buttons to move a highlighter from card to card in any kind of intuitive fashion. Our solution was to address the most obvious interface of the phone, the number keys. We placed small numbers alongside each playable card, and players simply press the indicated button to act on the card. As cards are removed, these numbers "follow" the card formations as if on small tracks. This means no action in the game requires more than one or two button presses, whereas attempting to move a cursor around would have been awkward and frustrating, if not visually unpleasant due to some of the other limitations of the phone's graphics (which we'll discuss later).

Palm and PocketPC devices can be just as bad or worse than mobile phones. Palm-compatible devices typically have a two-directional button pad (up and down) and four additional buttons. The buttons immediately flanking the up-and-down buttons can be used as left and right to simulate a typical four-way directional joystick, but their placement isn't very suitable to this. And, as with the phones, they are often recessed and difficult to press. PocketPCs typically have a four-way directional pad, but these often have their own



Lack of transparency forces use of composite graphics containing all elements required for that portion of the screen.

DALE CROWLEY | As founder and CEO of NuvoStudios, Dale was responsible for the studio's initial push into the wireless and mobile space. He established early relationships with Motorola, Nokia, and Qualcomm and has guided NuvoStudios through the production of more than 75 games on six platforms. Dale can be reached at dale@nuvostudios.com.

MAURICE MOLYNEAUX | Maurice is the director of production for NuvoStudios. With more than 13 lucky years of game development experience, Maurice applies his extensive background in art, animation, scripting, design, and project management to all of Nuvo's games. Maurice can be reached at maurice@nuvostudios.com.

WAYNE LEE | Wayne is senior engineer at NuvoStudios and has been responsible for many of the company's key technology innovations. Wayne can be reached at wayne@nuvostudios.com.

RAMESH VENKATARAMAN, PH.D. | Ramesh is NuvoStudios' CTO and has helped lead NuvoStudios through the many technology hurdles that an emerging-market game developer faces today. Ramesh can be reached at ramesh@nuvostudios.com.

problems, such as the difficulty of actually depressing the buttons. A traditional joy-pad offers comparatively little resistance.

A further consideration beyond the ergonomics of the available controls on mobile and wireless devices is that the underlying hardware has limitations that no self-respecting game-controller designer would allow. A common one with hand-held devices is that you cannot press more than one key simultaneously. The iPaq H3650 suffers from this, and on that device the problem is so serious that it effectively prevents the developer from using its directional pad analogously to a videogame joystick, because while both are four-switch devices, the iPaq cannot read two switches as being pressed at once to make a diagonal, whereas a joystick can.

Not being able to read input from two or more buttons simultaneously can be a severe limitation for action games. For example, when we ported a version of TETRIS to the J2ME platform, one of the issues that arose was that the arcade and home versions of the game allow you to simultaneously move and rotate a piece by working two controls at once. Since the target phones did not allow for two control inputs, we had to consider slowing down the game in order to allow the player to make these moves separately.

Device Speed and Memory

The processor speeds on devices currently on the market are obviously much lower than what we are used to when writing games for PCs — with the possible exception of the newer PocketPCs. This affects everything from the speed at which data can be loaded to the frame/refresh rate, and seriously limits the ability to animate characters or otherwise rapidly change the screen.

As with most small devices, there are two kinds of memory, the storage memory (usually flash RAM or static RAM), and the memory in which the application runs (dynamic RAM).

On mobile phones the storage RAM is usually extremely limited, and, because each phone allows the user to store multiple applications, the allowable maximum sizes for a given program can be extreme-

ly small. One device we considered developing for permitted an application size of no more than 10K of storage space!

Others are more generous, with upwards of 60 to 80K. Upcoming color devices promise more memory still, but if their 8-bit color graphics take up eight times as much space, then the problem might be just as severe.

Another reason application sizes have to remain small is that these devices typically have limited heap space, restricting the amount of dynamic memory for variables, stack, and graphics.

BREW is especially nasty with memory currently, because, as on the Kyocera QCP 3035, the application isn't just executed in dynamic RAM; the entire application is copied from storage RAM into dynamic RAM and then executed. You're effectively penalized twice for a big application.

The memory constraints of the Palm OS are nowhere near as restrictive as for BREW, but if you're supporting older hardware and older versions of the Palm OS, be aware that heap space is very limited, as low as 36K for Palm OS 3.0. On the plus side, any Palm-powered device running OS 3.0 only supports black and white or grayscale graphics, reducing your memory requirements. Any color Palm device uses at least Palm OS 3.5, which has a much roomier heap. And, like the Kyocera 3035 phone, images stored in resources do not use any of the dynamic RAM until they are loaded into the heap, so you should unload images as soon as you are done using them. Also, data stored in databases are placed in storage memory, so consider putting large tables in .PDB (Palm Database) files.

Since all shipping Pocket PCs have at least 16MB of RAM, storage and heap space is less of an issue on these devices than others.

Power-Save Considerations

Virtually all wireless devices have power-saving modes where they shut themselves off, deactivate the screen, or turn off the screen's backlighting. On most devices this doesn't cause any loss of data or necessarily terminate the application running.

LISTING 1. On the Kyocera QCP 3035 BREW phone, example B is an order of magnitude faster than example A.

```
// A: Draw background with a bitmap

IMAGE_Draw(f_bg, 0, 20);

// B: Draw equivalent background with just rectangles

IDisplay *d = a.m_pIDisplay;
AERect r;
r.x = 0;
r.y = 20;
r.dx = SCREEN_WIDTH;
r.dy = 45;
IDISPLAY_FillRect(d, &r, RGB_BLACK);
r.y = 65;
r.dy = 14;
IDISPLAY_FillRect(d, &r, RGB_WHITE);

r.dy = 1;

r.y = 47;
IDISPLAY_FillRect(d, &r, RGB_WHITE);
r.y = 53;
IDISPLAY_FillRect(d, &r, RGB_WHITE);
r.y = 59;
IDISPLAY_FillRect(d, &r, RGB_WHITE);
r.y = 63;
IDISPLAY_FillRect(d, &r, RGB_WHITE);
```

However, on some devices not only does the power-save mode turn off the screen's backlight, but the entire device slows! We first noticed this on our WWF MOBILE MADNESS game for the J2ME Motorola i85s, which has a "spectator" mode that lets you watch a match without playing. Unless you thump a key now and then to keep the phone in an active mode, the match starts to look like a slow-motion

instant replay. Because the device keeps shifting gears on players, it can be difficult for them to get into the rhythm of the game. There's no simple solution to this, because while you might consider tuning your game so that the maximum time between required button presses is less than the inactivity timeout for the device, fortunately, sometimes these timeouts are user-adjustable.

Resolution and Screen Technology

The limited resolution of wireless devices is as much a factor on phones as PDAs. Palm compatibles have a typical screen resolution of 160x160 pixels, with palettes of varying ranges. Some are four grays only; some are 16 grays without OS support for 16; some are 16 with OS support for 16, and some are 256-color. Mobile phones have resolutions as low as 89x99 pixels in black and white, and then again up to 128x142 in 256 colors. However, while the difference between a high-end phone and a low-end PDA might not seem so pronounced, most users of such devices don't have the top-of-the-line models. The reality is that the mass market is more likely to be in the low and middle ranges for each.

The type and quality of the LCD screens is another factor. Aside from the speed at which the hardware can update the screen memory, there is the issue of image persistence. Active-matrix displays refresh much more quickly than cheaper, passive-matrix displays. This often leads to momentary after-images and serious blurring problems on the passive-matrix devices. So what might look fine on a Palm m505, which has an active-matrix screen, will blur significantly on a passive-matrix device such as the Handspring Visor Deluxe. This means you either have to avoid having lots of fast-moving objects or scrolling, do different games for different devices (which no sane product manager will allow), or just shrug your shoulders and accept the fact that the game will look great on some devices and messy on others.

Another big factor is that many of the devices in question do not have native support for images with transparency, nor do the SDKs available for them have any tools



Test of graphics developed for multiple devices of different bit-depths.

for simulating it. This means that writing games that include traditional sprites is difficult. On some platforms, custom sprite engines can be used with acceptable performance hits, but on others, such as the J2ME phones, the amount of processor time required for doing transparent blits was prohibitive for us.

For WWF MOBILE MADNESS we first came up with the solution of keeping the characters moving on a single plane with a uniform background, and included parts of the background into the characters' art, thus giving the impression of transparent blitting where none exists. This worked fine on the Motorola i85s J2ME phone, but not so well on the Qualcomm QCP 3035 BREW phone. The former could push upwards of 12 frames per second, but on the latter we were lucky to get two frames per second doing the exact same animation. To eke as much speed as possible out of the QCP 3035, we abandoned restoring the background bitmap when characters moved and instead used BREW's rectangle-drawing function to rebuild the background quickly, effectively doubling the frame rate. This kind of solution requires appropriately designed art in order to be feasible.

Graphics Development

Whereas PC and console graphics have become increasingly detailed and colorful, the types of graphics displayable on most wireless and mobile devices are more akin to computers circa 1984 to 1989. Making good-looking graphics with these limitations requires skills that for years have been out of

demand in the mainstream. The ability to draw clear, concise images in extremely low resolutions and with very small palettes is very important. And, if you're animating characters at these small sizes, you need artists who can draw dynamic, easily read poses; otherwise a kick might look like a dance step, and a punch like an arm wave. Your typical Photoshop-trained artist is not necessarily suited to this realm, where the placement of a single pixel is often the demarcation point between clarity and mud.

Naturally, the most challenging screens to develop for are the 1-bit black monochrome displays. The difference between 1-bit black and white and 2-bit grayscale is surprising. In fact, it can be enormous. The antialiasing provided by the latter can make a huge difference in the clarity and detail of images, especially on low-resolution devices.

Portability

Another key design issue that needs to be considered when developing games is the often dramatic difference among various handhelds within the same family in terms of their speed, memory, and other specifications. For example, while there are several BREW and J2ME phones on the market or coming to market, the differences between the low and high end can be considerable, especially where graphics are concerned. As a game developer, you do not always have the luxury of developing different versions of a game for each of the target devices. Often this means that you end developing for the lowest common denominator — the cheapest phone or PDA. That becomes your baseline, and then you make adjustments for specific devices.

Graphics are a place where a lot of adjustments are necessary in both production or art assets and game code. For instance, the version of MUMMY MAZE we did for JAMDAT required entirely different art for the 89x99 1-bit QCP 3035 and the 128x142 8-bit Sharp TQ-CX1. Furthermore, the differences in screen resolution required different coordinate offsets for all the graphics, and even the help system text had to be reworked to make the best use of the larger screen.

Case Studies: Developing for a New Generation of Phones

Developing games with BREW. The BREW SDK, developed by Qualcomm, is intended to facilitate the development of applications that can be easily ported to various handheld devices. The BREW SDK consists of an emulator and a set of APIs that can be used to develop the required application. With the BREW toolkit (which we found only works well with Visual C++) you first build binaries for the emulator, which take the form of Windows DLLs. Once the DLL has been created you can begin testing the application on the emulator.

The emulator is a Windows program that implements the BREW API. However, we found that it does not truly emulate the configuration found on the phone. For example, we found that the sound on the emulator did not correspond with the sound that came out of the phone. Similarly, we found that refresh rates on the emulators were different from the actual refresh rates on the phones.

In BREW, we found it useful to put large tables into a file, because these files are not copied into dynamic RAM (unlike the code for the application itself).

When you are using bitmapped graphics in BREW, loading images from a resource is reasonably quick. As a plus, images that are sitting in such a resource don't take up dynamic RAM, so you can often get away with loading images only on demand and discarding them soon after you're done, even during an animation. For example, if you have a large image strip (one graphic containing many cels), consider splitting it up into several smaller image strips and then loading each strip just as it's needed.

Once you are ready to test the game on the actual phone, you need to compile and link the code into ARM binary form using the ARM BREW developers' pack, which includes a compiler, linker, and assembler. It is this binary that you actually download into the phone using a serial link. While in theory, whatever compilers for the BREW emulator should also compile when using the ARM developer

kit, there are eccentricities in the compilers that can cause headaches for the developer. For example, the use of global variables was not disallowed when we were writing for the BREW emulator. However, this was flagged by the ARM compiler and required us to rewrite some of our code.

In general, expect the same kinds of general differences you would expect when going from one manufacturer's C compiler to another.

Developing games with J2ME. In J2ME, consider using public variables in your classes, rather than using accessors (`getValue()`, `setValue()`, and so on). Yes, technically doing so is "bad" programming practice, but it saves you space because in Java, the implementation and the use of accessor methods add to the size of the code.

To maximize performance, be extra careful to have things in memory only when they're in use. For example, if you have a splash screen for the game intro, discard it after it's done displaying. Furthermore, consider reducing the number of classes used if your design allows it. You can combine classes into one if they vary only slightly in behavior. There's an unavoidable size overhead for each class you use, even if it does nothing.

It's also important to remember that loading and installing applications into J2ME phones is a relatively slow process. If you have to do numerous revisions in QA, this may result in delays because you can't quickly test new builds.

Development on the Edge

Developing applications for small handheld devices brings with it a unique set of challenges and opportunities. The newness of these devices means that many questions about them (such as how to develop for them) have not yet been fully answered. Sometimes we have found ourselves developing for devices that do not exist outside of someone else's lab. As a result, while developing content we have often found ourselves trailblazing in just doing things such as shaking the screen. Frustrating, yes, but ultimately rewarding. Such are the drawbacks and benefits of working on the cutting edge. 🚧

LISTING 2. `SmallerClass` results in compiled code that is 30 percent smaller than `BiggerClass`.

```
public class BiggerClass
{
    private int f_health;
    private boolean f_isGroggy;
    private int f_forwardDir;
    private int f_index;

    public void setHealth(int h)
    {
        f_health = h;
    }

    public void setGroggy(boolean groggy)
    {
        f_isGroggy = groggy;
    }

    public int getIndex()
    {
        return f_index;
    }

    public void setForwardDir(int f)
    {
        f_forwardDir = f;
    }

    public static void
    ExampleUsage(BiggerClass obj)
    {
        obj.setHealth(100);
        obj.setGroggy(true);
        obj.setForwardDir(obj.getIndex()
        == 1 ? 1 : -1);
    }
}

public class SmallerClass
{
    public int f_health;
    public boolean f_isGroggy;
    public int f_forwardDir;
    public int f_index;

    public static void
    ExampleUsage(SmallerClass obj)
    {
        obj.f_health = 100;
        obj.f_isGroggy = true;
        obj.f_forwardDir = obj.f_index ==
        1 ? 1 : -1;
    }
}
```

Wireless Game Content: How to Think Big In A Small World

hai·ku (hī'kōō) *n. pl.* haiku, also hai·kus

A Japanese lyric verse form having three unrhymed lines of five, seven, and five syllables, traditionally invoking an aspect of nature or the seasons.

Here's an example:

*Cell phone games are fun
A VC falls in the woods
Alone, will kids pay?*

Developing for the current U.S. marketplace for wireless gaming is not far from writing three-line Japanese poetry in terms of limitations. In fact, on some phones the text will probably have to wrap and the user will be required to scroll down to see the whole thing. O.K., on some other phones, the 96-pixel-wide by 30-pixel-tall black and white bitmap of the tranquil forest will also be displayed above the poem. Of course, depending on the network, the phone may also require a connect sequence between the text and the picture, and you can burn another three to five seconds waiting. And that's the way it is on WAP today.

The BLAM! team has developed titles for the PC, Playstation, 3DO, desktop browsers, Palm Pilots, Commodore 64, Atari ST, Amiga, and Sega Saturn platforms. Coming from today's world of high-end graphics, relatively large amounts of memory, and custom processors on consoles, it is easy to see only the limitations of what can be achieved on phones. Admit it, on any platform it's hard to know what makes those fickle audiences out there like or dislike a particular game. And considering the extremely narrow canvas of cell phones, you might wonder why you should even try to make games on these tiny devices at all. The bottom line is that the formula for a fun, compelling wireless game title is difficult at best to define. Yet there are undeniably compelling reasons for making games for cell phones.

First of all, the cell phone market is continuing to grow. The numbers are there from the highly paid market research

firms, so I won't take the time to justify that claim. More importantly for the game development community, people with phones are looking to fill with entertainment that short sliver of time spent waiting for the bus or in line at the bank. Even more importantly, kids will always play videogames on any device that has a screen — just because they can.

Second, the cell phone is a personal, connected communication device, making it a truly unique platform for game development. The traditional console or even the handheld videogame experience has predominantly been a solo or at best a very small group experience (four players or fewer, unless you were lucky enough to experience eight-player



BLAM!'s SnapShot Live Football, published by Sorrent, is available on SprintPCS and AT&T Wireless.

BOMBERMAN on the Saturn). In contrast, your average cell phone game player has the potential to be connected with a community of millions of other users, all the time — most people leave their PCs, game consoles, and even handheld consoles at home much of the time.

The growth in the mobile phone market makes it a lucrative venture to try to capture even a small part of the massive potential audience. Also, it

doesn't take 2-plus years of slogging away at art, design, and programming to complete a project and get to market. But creatively, it is the connectivity and personal relationships that people have with their cell phones that compels some of us mobile game developers to wade through the sea of incompatible standards, middleware, and platforms to create the content. The challenge is to make a game that captures the imagination (often that's all you get) and turns game players into addicts.

Early examples of WAP titles such as Hangman, Trivia, and Solitaire have evolved and given way to the current generation of WAP titles, all of which have elements of community and/or persistent character development. Audiences have burned many minutes of their precious air time playing games, because they were given an opportunity to care about their standing in the community on a high score board, for example. On the persistent-character development front, once players invests some time in a character and can continue to make that character more powerful by spending time playing, they will then obsessively burn time to keep going. Ask any Everquest junkie with a 50th-level or better character. That is true in the wireless world as well, where popular games allow users to create and build their character's power indefinitely. Even on a cell phone, players can, for example, build up their character's Broadsword Skill and thus, be able to take on more powerful enemies, thus opening up new areas of the game, and then getting more items, and so on.

Remember, today's talk about WAP is only about today's content development. As we see phones enabled with J2ME (Java 2 Micro Edition) and BREW (Binary Runtime Environment for Wireless) become adopted by the marketplace, we can easily expect to see cell phone games look and play more like Game Boy games. Soon we

JAY MINN | Jay co-founded BLAM! in 1995. He and his staff are currently focusing their efforts on developing entertainment content for desktop browsers and mobile devices. BLAM! has alliances with publishers, distributors, and carriers, including JAMDAT Mobile, Digital Bridges, Pogo.com, Iwon.com, Qualcomm, Sun Microsystems, Sprint PCS, and AT&T Wireless.

will be able to play almost anything that the early handheld games were capable of displaying, with scrolling, tiles, sprites, collision systems, particles, and animations. And they will all be connected.


As audiences start upgrading their cell phones to more powerful hardware, with more memory, color screens, Java or BREW enabled, and faster, always-on connections, we will quickly see games reach a whole new, increased level of pervasiveness. The choice of games that a user downloads or subscribes to will be part of the customization of their phones, as much as ring tones, background pictures, and custom plastic cases are today.

The games will be with you wherever you go and will alert you, for example, if a neighboring clan of warriors is attacking your territory. You will meet your teammates to participate in a large-scale air attack in your jet fighter as part of a massive campaign. You will be part of a live soap opera, administered by a real-time editor at the server integrating your input and sending back the resulting script via SMS. This is all possible today.

Farther off in the future, with global positioning information becoming part of the available technologies, we'll get the chance to tweak the content formula. Players in a game will then be able to opt in to allow for their physical location information to be a part of the game world. As you drive away from a certain city block, you move out of range of your opponents' virtual weapons, for example. Voice-activated menus or even whole adventures based on voice interactions may also be available to augment the wireless gaming experience. Finally, fatter wireless pipes will let developers create truly robust games with massive client applications delivering everything from audio to streaming video.

All that is coming, some say sooner, some say later. But we must develop for today, and build up for the future. In the course of doing so, we must remember that we cannot approach wireless game development from trying to figure out how to replicate the traditional videogame experience on a personal device. That approach will inevitably result in frustrating, unplayable titles that do not take advantage of the uniqueness of this new medium: connected and mobile users.

Our job, then, becomes the task of creating an intuitive interface that does not require a whole lot of explanation on a very limited screen space and hardware while capturing the essence of connected, community-based games on the cell phones. Interface design, immediately following game design and technology itself, will be the make-or-break factor in the

adoption rate of a given title. If audiences don't get it, or they have to fumble around to find which buttons to press, then the title hasn't got a chance. Add to it that we are also trying to capture a multiplayer, community-based experience, and therefore has more complex requirements for interface, we have the true challenge for game development in the wireless space. 



The Business of Mobile Games

Distributing games through wireless networks is a new frontier for both the online and handheld game developer. As with any new platform, there is a great deal of hype about the potential for the mobile games business coming from hardware vendors, phone companies, and Wall Street analysts. How does a game developer (or publisher) identify the real opportunities in this nascent market?

Rule #1: Follow the Money

The mobile games market represents the largest new interactive entertainment opportunity for developers and publishers. Around the world, a staggering number of consumers have access to wireless devices — currently upwards of 600 million people subscribe to wireless voice services, and that number is projected to grow to well over 1 billion by 2005. In certain Asian and European countries, wireless services have achieved population penetrations greater than 60 percent.

While wireless phone manufacturers currently embed a few primitive games in their devices (such as Nokia's ubiquitous SNAKE), the real opportunity for mobile gaming is tied to the growth of wireless data services and the corresponding proliferation of data-enabled phones.

Consumers can use data-enabled phones to access a vast array of content and services. Phone technology is advancing rapidly — new phones have large color screens, memory, processing power, and an always-on connection to the Internet at decent data rates. Games will be one of the leading applications driving this new market — currently, games are generating as much as 25 percent of data usage in certain territories — and some analysts have projected a global mobile games business as large as \$6 billion annually developing over the next five years.

True, we've all seen these kinds of pie-in-the-sky projections before in connection with online gaming — and that certainly didn't turn out the way the analysts predicted back in 1995. So why should we

expect mobile gaming to fare any better? There is one key difference between the mobile games market and the online games market: money. Unlike online gaming, which is either supported by advertising or which requires consumers to hand out their credit card numbers on the Internet, mobile gaming leverages the pre-existing billing relationship between consumers and wireless phone carriers.

Every month wireless carriers send out millions of bills to consumers. Those bills contain voice charges, as well as other service charges (such as directory assistance calls, or monthly fees for data services). The carriers' ability to generate cash by billing consumers provides the basis for the revenue opportunities of game developers and publishers (and I might add provides the leverage carriers use to maintain their position as gatekeepers of content).

In Japan, wireless carriers such as NTT DoCoMo are capable of billing on behalf of content publishers. The major European and North American carriers are all working on their own "bill on behalf of" systems, which will begin rolling out over the next year. Once these systems are in place around the world, game publishers will be able to get paid for their content directly. In the absence of new billing systems, content providers have only indirect revenue opportunities — sharing the revenue carriers derive from per-minute or per-message access fees, or monthly service charges.

Where's the Platform?

Perhaps the most overused term in the wireless games business is "platform." Companies large and small have concentrated on creating game platforms, attempting to generate predictable revenue streams through platform license fees and other associated toll charges. In fact, nothing like a console platform exists in the mobile games business.

Most companies trying to create platforms for mobile gaming misunderstand the way console platforms work in the videogame business. In the videogame context, companies like Sony and Nintendo absorb massive capital losses manufactur-



JAMDAT's GLADIATOR is an multiplayer wireless experience based on ancient Rome

ing hardware, develop a retail channel for software, and spend hundreds of millions of dollars on marketing. The proprietary control of content and per-unit license fees commanded by console manufacturers acknowledges the huge financial risks they have taken to propagate their platforms.

In the mobile game business, no single company stands in the position of a Sony or Nintendo. Instead, handset manufacturers, applications environment providers, Internet portals, component suppliers, and middleware companies have an interest in defining platforms for mobile games. In general, publishers and developers should be skeptical of proprietary platforms at this early stage of market development.

The lack of standards makes the mobile games business look more like the PC games business, without a Microsoft to set de facto standards like DirectX. For example, there are at least a half dozen incompatible browsers in mobile phones. Java Virtual Machines are sometimes implemented inconsistently in mobile phone handsets sold by the same carrier. The wide diversity of wireless applications environments and devices, and the corresponding lack of compatibility, has required companies such as JAMDAT and others to invest heavily in middleware technologies to enable multi-platform development, data gathering, and billing integration.

MITCH LASKY | *Mitch is CEO of JAMDAT Mobile, a leading provider of mobile entertainment and enabling technologies. Prior to joining JAMDAT, Lasky was executive vice president of worldwide studios for Activision.*

Alphabet Soup

In the absence of broadly-accepted platforms for content distribution, developers need to be familiar with the most popular messaging systems, browsers, and next-generation applications environments. These technologies provide the base level of functionality needed to enable mobile games, but our view of the marketplace today is that no one platform will dominate. You will need to diversify your development chops to survive.

The simplest way of distributing games to data-enabled phones is through the Short Messaging Service, or SMS. An SMS message is typically a 160-character message that can be read on a mobile phone.

Next step up in the wireless food chain are data-enabled phones that contain Internet “mini-browsers,” which offer a significantly better platform for gaming. There are currently more than 40 million phones with Internet browsers (WAP or HTML-derived) in the Asian markets, approximately 5 million in North America and perhaps 10 million in Europe.

Then there are the newest standards, J2ME and BREW. J2ME and BREW represent a transition from browser-based gaming to gaming based on downloaded, or pre-loaded, executables. Using phones equipped with these applications environments, consumers can download games to their phones and run them locally.

The download model for J2ME and BREW games also offers the most compelling revenue model for game providers. In Japan, for example, consumers can download Java games and play them for some specified amount of time for ¥100 to ¥300 (US\$0.85 to \$2.50) — with 80 to 90 percent of the purchase price flowing back to the game publisher. A similar download and revenue model for BREW content will launch this fall in the United States.

Carriers Rule

There are many key players in the mobile content value chain. Developing and publishing mobile entertainment applications involves navigating a complex web of carriers, handset manufacturers, operating system and application environment providers,

component manufacturers, portals, content publishers, and tools companies.

The wireless carriers are the most powerful players in the mobile game marketplace. They not only function as the retailer for software (like Best Buy, Wal-Mart and Electronics Boutique in the retail games business), they also subsidize and sell the hardware devices that can work on their networks. Often, they dictate the applications environments that will ship in those devices. In Asia and North America, carriers are particularly powerful. In Europe, however, the widespread use of SIM cards to enable network access has put handset manufacturers like Nokia in a much stronger position versus carriers.

The high cost of building and maintaining nationwide wireless networks has led to a lot of carrier consolidation, leaving only a few important carriers in each territory. For example, three carriers (NTT DoCoMo, KDDI, and J-Phone) control Japan; four carriers (Sprint PCS, AT&T, Verizon, and Cingular) hold approximately 70 percent of the U.S. market. Therefore, in order to have a successful distribution strategy, publishers have to have solid relationships with the major carriers. It follows that developers have to have solid relationships with publishers.

The Role of Publishers

Publishers (and other aggregators of entertainment content, like portals) play an important role in the mobile game business. Publishers maintain the hosting and serving environments that carriers rely on to provide games to end users. Also, because carriers are unwilling to underwrite the cost of creating content, publishers provide capital to developers in the form of development advances, just as in the retail game business.

Quality assurance testing is one of the most valuable services a publisher can provide. And as in the videogame console business, a publisher’s experience navigating the certification process can be extremely helpful. Publishers also provide access to localization services — which are vital to reaching the global consumer base.

How should developers evaluate potential publishers? The quality of a publisher’s

carrier relationships is probably the most important criterion. Are they getting paid? How is their reputation in the carrier community? Are they supplying something other than games (such as enabling technology) or participating in trials of next-generation systems?

What about the existing retail game publishers? In Japan, major publishers like Bandai, Namco, and others are distributing mobile games. To date, of the U.S. and European publishers only THQ, a traditional powerhouse in handheld gaming, has announced official plans to publish mobile games.

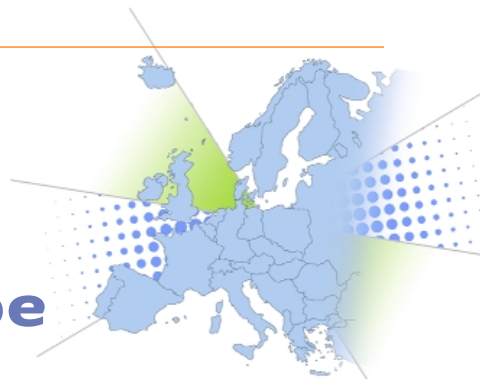
Taking the Plunge

Is developing for the burgeoning but untamed wireless sector for you? Creatively, mobile games offer some unique advantages. The games themselves are typically small (under 100KB for downloadable games), and it is possible for an individual or very small development team to create a product from scratch in a small fraction of the time it takes to develop a full-featured PC or console game.

While the current audience for mobile games is small, it is growing rapidly. Wireless carriers are investing heavily in data services, because the profitability of voice services has been eroded by stiff competition, and carriers view data services as the way to recapture this lost revenue. Thus, carriers are aggressively promoting data capabilities and are looking for “killer apps” — like games — to drive consumer usage of data services.

Most importantly, the fundamentals of the mobile game business are very attractive. Mobile gaming combines the best attributes of online gaming (lower development costs, rapid time to market, no physical inventory) with the best attributes of the retail games business (access to an efficient distribution network, the availability of paying customers), and has the added benefit of a global mass market of consumers that will number in the hundreds of millions in the next five years — an order of magnitude larger than the most successful videogame consoles.

The opportunities are there. Choose your platforms, publishers, and partners with care. But don’t delay; the sector is picking up speed with each passing day. 🚀



The Wireless Scene in Europe

Europe has long been at the forefront of the creation and adoption of the wireless Internet, but Europe offers several significant differences from the other major markets in the world, both technically and from a business standpoint. What can U.S. developers learn from our experiences in the U.K. and on the continent?

Technology. Unlike the U.S., where systems such as CDMA (Code Division Multiple Access) and TDMA (Time Division Multiple Access) prevail, and where moving between states can make your cellphone as useful and attractive as a brick, the whole of Europe, as well as most of the Middle East and Africa, benefits from a single, unifying standard, the Global System for Mobile Communications, or GSM. This provides a common underlying structure for all mobile communications, including voice and data calls. Users cross from cell to cell, network to network and country to country (almost) seamlessly. However, the call charges will probably kill you. GSM is making progress in the U.S., so Americans may yet come to know the joy of SMS.

The current generation of wireless Internet phones make use of the Wireless Application Protocol (WAP) to access the Internet. Graphics are limited, with monochrome screens offering an average resolution of approximately 90x40 pixels. Transmission speeds throughout Europe are currently based on the standard rate of 9.6K per second. Networks offering so-called 2.5G transmission using the General Packet Radio Switching (GPRS) system are rolling out in many territories across Europe, although GPRS-enabled handsets are not yet widely available.

Before the end of 2001, phones offering on-board Java processing and grayscale/color screens should become widely available throughout Europe. In the wireless industry, this is as exciting as the move from 8 to 16 bit. No, really.

The marketplace. Here's where things get more familiar. As in the U.S., almost all of the operators throughout Europe have created their own portals to provide wireless (and fixed) Internet services to their sub-

scribers. Recent research has shown that most of the traffic on the wireless Internet in Europe to date has come through these operator portals. In addition, several major mobile network operators in Europe cover multiple territories.

These portals offer the best possibility for making money from game and entertainment services, since they tend to have operator backing and can offer revenue-sharing deals for inclusion on their menus.

Currently, most of the operators in Europe are charging users for the air time they generate while playing games. This is already in the process of changing, with most of the portals somewhere along the line of introducing specific payment schemes such as monthly subscriptions, micro-payments, or voucher systems. This is fairly important, since it creates a direct value chain with the developer at one end and the end user at the other.

Gaming. Almost all of the games available to European mobile phone users to date have made use of Short Messaging Service (SMS) or WAP services. All of the major WAP portals now offer a selection of both types of games to their users.

To date, the SMS games available have been fairly limited, offering a variety of quizzes and adventure games, but more complex and sophisticated games are due out later this year and in early 2002.

STAR TREK: PRIME DIRECTIVE developed by **Wirelessgames**, being played on a **Panasonic GD93**



Even WAP games have grown increasingly sophisticated over the past 12 months as more

imaginative developers have entered the market.

There are several companies throughout Europe who are already working on titles that offer advanced new services such as massively multiplayer capability (**STAR TREK: PRIME DIRECTIVE**, for one) and location-based gaming (**BOT FIGHTER**). Each user connects with a central server whenever they make a move, therefore creating multiplayer games is much simpler on mobile devices than on almost any other platform. Add to this the fact that all players are having the game delivered to them, regardless of different devices, optimized for different screen sizes and input devices, and delivered to each player in his or her native language, well, you see why wireless is going to be such a big deal globally.

To date, Europe has suffered from a misguided perception that services such as stocks and share prices and weather reports would really pull in the punters, but this is looking less likely. Recent research carried out by the European wireless trade journal, *Mobile Internet* found that on one of Europe's top WAP portals, gaming accounted for more than 16 percent of the total traffic generated on the portal, beating services such as messaging, sports, and even pornography in terms of popularity.

The future. The European mobile market is evolving at a staggeringly fast rate. Before the end of 2001, new handsets with improved graphics, browsers, GPRS support, and Java processing will be available. The operator-run portals will be charging users via micro-payment or subscription schemes and advertising key titles available through their portals. As awareness of mobile gaming grows, we expect the number of major publishers and media companies involved in the industry to multiply. Within the next 12 months, several multinational games and media companies will set up their own branded gaming channels to take advantage of new users.

The bottom line. Time to get over the screen size, guys. ☺

BRIAN BAGLOW | *Brian started his games career at DMA Design in Scotland. After spending a highly educational year and a half at Take 2, he returned to life in Scotland and the cutting edge of gaming by joining Digital Bridges as global communications manager.*

Japan Gets Hooked on Java



Now, don't take it personally, but Japan is way ahead of the pack when it comes to wireless gaming. Japan leads the world in wireless technology and, more importantly, in crafting mobile Internet access into a hugely popular, revenue-generating industry, and games are poised to be one of the major drivers of technology adaptation and usage.

Games coded in the Java programming language represent ground zero of the second wave of Japan's mobile game industry. The past 10 months have seen the three giants of the cellular space, NTT DoCoMo, KDDI, and J-Phone, announce or roll out Java-based services. For the first time anywhere, Java on Japan's wireless Web is enabling compelling, massively multiplayer games to be delivered direct to players' pockets, packaged in tiny *keitai* (cell phones) and weighing in at under 50KB.

NTT DoCoMo launched its i-Appli Java service in January 2001, and i-Appli usage has been rising with i-mode usage. As of July 31, DoCoMo had 26,085,000 i-mode subscribers, and Java users accounted for some 21 percent of all i-mode users, according to the company.

Japan's wireless webs. DoCoMo's i-mode, KDDI's EZweb, and J-Phone's J-Sky services are packet-switched, low-bandwidth, and always on, and all deliver text, graphics, and other web content to a micro-browser resident on the handset, which also serves as the user interface for e-mail, short mail, and telephony control functions.

Carriers allow official site owners to charge subscribers between ¥100 and ¥300 (US\$0.81 to \$2.44) per month to access their sites, which are listed on default menus preprogrammed into the phones. The fees are added to the monthly cell phone bill, and, since the carrier serves as billing agent, the site owners pay a commission, presently 9 percent on i-mode, for example; the balance goes to the site owner.

Java rules. Java games appear to be well suited to the existing mobile infrastructure. For Java game sites, a typical fee is ¥300 for three downloads per month. In addition, carriers earn revenue from data packets sent to and from handsets — which is one reason why Java, and in particular, data-swapping, multi-user, and role-playing gaming, is being pushed by the carriers (developers, take note!).

The advent of mobile Java has seen a tremendous burst of activity by major and smaller game developers and publishers alike this year. Two-year-old Tokyo-based Cybird is a good example of the new breed of mobile content developers that focus

solely on wireless. The upstart content developer supports some 22 separate sites spanning shopping, sport fishing information, surf conditions, maps, and ring tones.

Traditional console and PC companies like Bandai are also jumping on Java. Bandai's i-Appli channel features the GUNDAM and YAMATO series of Java games, with Java GUNDAM being an excellent example of co-branding between game platforms (versions already exist for

Playstation and Dreamcast), and other media (there's a cartoon series on VHS and DVD as well).

What's important for creating successful games, Bandai PR executive Yukiko Takahashi points out, are features that include player rankings (national and regional), frequent updates, integration of real-time information (such as location data), and making sure that games are "cool." Tokyo-based wireless analyst Andrea Hoffmann agrees, adding that Java games are more successful if they "have different levels and rankings, can be easily started and stopped, and if game status can be transferred to a larger PC or console version."

Perhaps the most exciting development in Japan's mobile Java industry has been

the emergence of wireless-only, Java-based, massively multi-user games. Independent developer Dwango appears to have scored a hit with SAMURAI ROMANESQUE, in which players take on the role of 15th century samurai, foot soldiers, generals, and other archetypes from Japan's era of the warlords. Dwango claims the game can accommodate up to 500,000 players, implying a significant degree of server-side development, and making the game highly network-dependent.

One of the game's most interesting features is the integration of real-time weather data provided by the Japan Weather Association. Game settings vary as local, real-world weather conditions change, so when it is (really) raining, a character may not be able to use firearms (the gun powder would be wet) or travel very quickly (the roads would be muddy). Bandai has also integrated weather information into several of its games, using data provided by Weathernews.

But a realistic assessment of Japan's mobile gaming market reveals that not all is happiness and light. The carriers accept only a few companies as official content providers, who can then list their Java games on the official menu and earn download revenues. The rest are no better off than on the Web at large when it comes to turning their titles into cash.

And while wireless is fast upending the tried-and-true console game distribution channels, in the new free-for-all turf of wireless content development, independent game developers worry about where they stand in the value chain of game content distribution. Absent the established developer-publisher relationship, will brand names become even more important?

Nonetheless, mobile gaming offers tremendous opportunities for growth and the subscriber numbers are certainly impressive — there's no other market like it, anywhere. ☞



SAMURAI ROMANESQUE, a Java-based RPG, was developed by Dwango for NTT DoCoMo's i-mode service in Japan.

DANIEL SCUKA | Daniel (daniel@japan-inc.com) has lived in Japan since 1994. He is editor-at-large for *J@pan Inc* magazine and edits the magazine's "Wireless Watch" weekly e-mail magazine covering wireless in Japan.

Digital Eclipse's RAYMAN ADVANCE



GAME DATA

PUBLISHER: Ubi Soft Entertainment

NUMBER OF FULL-TIME DEVELOPERS: 6

NUMBER OF CONTRACTORS: 1

LENGTH OF DEVELOPMENT: 8 months

RELEASE DATE: June 5, 2001

DEVELOPMENT HARDWARE: Standard fat PCs (1GHz Athlons, etc.) and G3 and G4 Macs

DEVELOPMENT SOFTWARE: Photoshop, Debablizer, and Nintendo's development stuff

Making sure a console game is a launch title — one that's on shelves the day a new piece of game hardware goes on sale — can make a difference of tens of thousands of units sold. So it's no wonder that publishers do everything they can to get their top-tier titles ready for launch, especially a launch as hotly anticipated as that of Nintendo's Game Boy Advance. The downside, of course, is that making a game for hardware that doesn't yet officially exist can often bring a huge number of unforeseen challenges.

When Ubi Soft first approached Digital Eclipse about doing a version of RAYMAN for the GBA, there were four priorities for the publisher (once we established that we would convert the Playstation version). First, the game had to live up to the quality of Ubi's flagship mascot. Second, it had to be a launch title. Shipping even a day after Nintendo's new handheld did was simply not an option. Third, it needed to be a faithful port of the Playstation version. The final requirement was that we needed to tweak the difficulty.

Rayman had only starred in three games up to that point: the original 2D platformer for PC, Playstation, Saturn, and Jaguar; a Game Boy Color game; and RAYMAN 2, a 3D action adventure for Dreamcast, Playstation 2, Nintendo 64, and PC. Ubi Soft was interested in a 2D, side-scrolling version of the game for GBA, and any design we came up with would have to live up to an august heritage: one of the last major 2D platformers, the original RAYMAN is widely regarded as one of the most challenging and well-designed side-scrollers ever.

When we first talked to Ubi Soft, we hadn't gotten our hands on a dev kit yet. According to the specs we'd seen, however, the system was about as powerful as a Super Nintendo, with a couple of additional features — scaling and rotation were possible on sprites, not just background levels. Initially, we didn't expect to be able to convert the Playstation version of the game. Rather, we thought we'd create a new game that looked similar but was done within the power of the Game Boy Advance. In fact, Nintendo's hardware proved to hold a lot more power than we expected.

In the process of determining the best way to do the animation, the project's lead artist, Granted Q. Savage, tried two different methods. One was hand-animating a new version of Rayman from scratch, the second was actually pulling the animation data directly from the original game. The conclusion: With some work, Savage felt that he could get all the graphics from the PSX or PC version onto the GBA, using its 256-color mode. The next issue was speed. Programmer Cathryn Mataga felt that the system could be fast enough, with the big question being access speeds to various parts of RAM.

In the end, rather than propose an original game, we felt that the GBA was powerful enough to do a conversion of RAYMAN 1. This also made sense because of the short (less than eight-month) schedule; by using the original game's design, we knew that we already had a fun (and challenging) game on our hands. Now all we had to do was remake it.

Adding stress was the deadline — we knew the game couldn't go a day late, so almost every decision we made was irre-

CHRIS CHARLA | *Chris Charla is production manager at Digital Eclipse. Prior to joining the company, he worked at Imagine Media in a number of roles, including editor-in-chief of Next Generation and Official Dreamcast Magazine.*

versible. Luckily, we made (nearly) all correct decisions. Our nerves weren't calmed, though, when Nintendo pushed the release date (and thus, our deadline) up by several days!

With some herculean efforts by the art and programming teams, and tons of testing by Digital Eclipse's internal senior producer Renée Johnson and associate producer Bill Schmidt, we made it. We've naturally been pleased by how the game has been received both critically and commercially. In some ways, though, we're sometimes afraid we did too good a job: most reviews and comments describe the game as "just a straight port." In reality, doing the conversion was anything but straightforward.

What Went Right

1. Publisher support. Ubi Soft is famous for the care that it takes with RAYMAN games (the two console games have a total of eight years of development time between them). To say that the company is fastidious about the mascot is an incredible understatement. Luckily for us, that care goes beyond lip-service.

Re-creating the game on a small screen and toning down the difficulty were both major design challenges for us. While we certainly felt up to the task, when we started asking questions about various issues, Ubi Soft sent one of the designers of the original Playstation game to our door. Lionel Rico flew in from Montreal to spend a solid week consulting on a number of game-play issues. This was an incredible help in ensuring that while the gameplay in our version was easier, it would remain faithful to the feel of RAYMAN.

2. Smart graphics choices. The Game Boy Advance has two options for color choices. You can either choose to use a single 256-color palette for all the sprites and a different 256-color palette shared among the four background planes, or you can choose to use



16 palettes of 16 colors for each. In theory, it shouldn't matter to the hardware which you use, and because "16 16s" can be more limiting than "one 256," we had initially planned to use two 256-color palettes, one for the sprites, and one for the background planes.

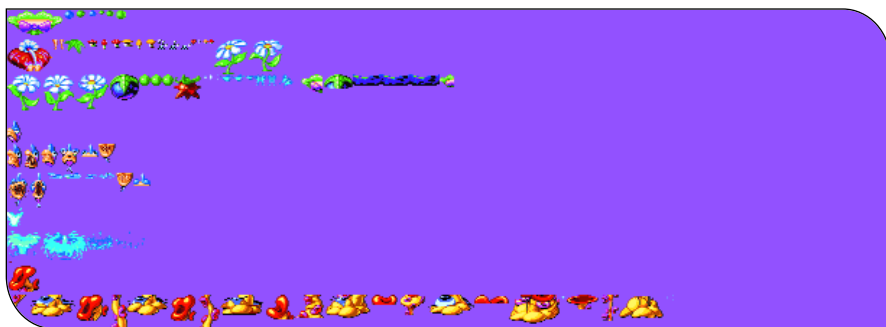
Once we got our hardware, however, we quickly learned that "16 16s" has a significant, though undocumented, speed advantage. Although it required much more work on the part of our art staff, we quickly changed plans to "16 16s."

Cathryn's original decision to go with 256-color mode was partially for the sake of trying to do things as simply as possible. Using 256 colors meant the art would look exactly like the original game. However, we discovered that the 256-color art loaded too slowly and ate up too much ROM space and VRAM. Switching to 16-color RAM allowed her to page-flip the sprite VRAM, and all of sprite VRAM was reloaded each frame.

It's no understatement to say that the biggest factor in the success of the project was the successful conversion of the graphics from 8-bit to 4-bit color. Our CTO, Jeff Vavasour, wrote an excellent tool that automated this process. Then Granted went in and hand-tweaked every palette (around 120 different palettes of 16 colors were used across all the levels of the game, including sprites and cutscenes) to make sure the game was compliant.

Unfortunately, while using 16 palettes of 16 colors each is faster than using a single 256-color palette, individual sprites can still use colors from multiple palettes, and ours did. To get the final speed boost needed, Granted had to go in and make sure that each sprite used colors from only one 16-color palette. Fixing one sprite usually required a palette change that broke a different sprite, and so on down the line.

The net result was a particularly brutal challenge, necessitating long nights with Photoshop and Debabelizer to get everything optimized. Ultimately there's no substitute for this kind of meticulous, back-breaking pixel work, and the work paid off in the look of the final game. As for Granted and his assistant artist Eric Calande, they looked on the task with



TOP. Sprites from the PC version of the game, each rendered using a 256-color palette.

BOTTOM. Our GBA versions of the same sprites, each using a single 16-color palette.

pride, not dread — at least the first time they had to do it (more under What Went Wrong).

Converting the palettes was just the first step in what needed to be done to bring the Playstation graphics to the GBA, since the original game did things with sprites that simply weren't possible on GBA. The original RAYMAN had large sprites that far exceed the GBA's sprite limits. Cathryn developed a small utility to assemble the large RAYMAN sprites from a number of differently shaped GBA sprites. The code optimized for the number of tiles, rather than sprites. On the levels where the sprites overwhelmed VRAM and the sprite limits, we switched to low resolution for some large graphics, such as the save-point graphic or certain bosses, then scaled them up using the GBA's hardware sprite-scaling.

Although the difference is noticeable (to us, anyway), it actually looks quite good on the small screen, and we were very pleased that the graphical compromise wasn't mentioned in reviews of the game. The graphics were tweaked until release to adjust between the number of enemies on

the screen versus the sprite resolution. When a conflict occurred, some enemies were removed so they wouldn't blink. Luckily, since one of our goals was to make the game easier, the need to remove enemies was never a major issue on any of the levels.

One of the earliest (and smartest) decisions we made was to not attempt to scale the graphics to fit the screen. While the original was 320×240 pixels, the GBA screen is 240×160. So, not only are the aspect ratios slightly different (4:3 vs. 3:2), but the graphics are one of the most impressive things about the original RAYMAN — scaling them down would, to us, have negated one of the best elements about the game. Even though viewable area of RAYMAN ADVANCE is smaller than that of the original, with the gameplay tweaks we made (more on this later) we found a compromise by which no real playability was lost.

3. Hacker ethic. We made some choices during development that probably weren't what they teach you in game development school. For instance,



This comparison of actual screens from the PC (top row) and GBA (bottom row) versions shows the similarity in the final results.

we had access to the original source art and to the original design tools, but all the art we used we actually took from the compiled game files. Cathryn did this because she felt it was easier to extract the data using the source code she was reading than to try to get the design tools working. This was a little bit of a pain, though eventually she extracted everything out to .BMP files for proper color reduction.

In most of our games, the character sprites are traditionally animated and saved intact to an animation file. Not so with the Rayman character, which is constructed from tiny little pieces (separate files for the hands, the head, and so on with the rest of the body) connected via animation tables. And the original table data was retained exactly. Although it may seem that it would be better for the artists to simply start with the source art, cleaning up and converting the “ripped” animations and backgrounds turned out to be much faster.

Converting the code also took knowledge of languages beyond C — many of the variable names were in French. A modest investment in a French-to-English dictionary solved most of the problems there.

The initial libraries that shipped with the GBA dev kit sometimes left something

to be desired. We made many changes to the library files, which ended up helping us out down the line.

4 • Gameplay tweaking. The original RAYMAN is famously hard. It was released at the tail end of the platformer genre’s heyday and may indeed be the most difficult side-scroller ever released. With platformers no longer in vogue, it was an open question at Digital Eclipse as to whether or not anyone would be able to beat the original today.

Ubi Soft was very open to tuning the difficulty of the game — in fact they suggested it. What the company did want to retain, however, was what Ubi Soft producer Yannis Marat called the “gameplay integrity.” So, things like collisions, jump properties, timing and sync challenges, the speed of animation, and enemy attacks had to be preserved, while actual platform and enemy locations changed dramatically.

Retaining the exact position of all the levels and all the items in the same place wouldn’t have been possible anyway, since the original played at 320×240 and the GBA screen is 240×160. The original game did several things that at the time simply made it a challenging platformer but today would be regarded as cruel. For example, you flew back when you took a

hit, which often (especially at later levels) caused you to fly off a platform to your death. Also, enemies would frequently hit you without you ever being able to plan for it (this also usually resulted in a quick death). Finally, there were several audio-only clues in the game. For instance, when you triggered an attack from off-screen, you’d need to listen for a “ping” sound or else you’d be caught unawares as an enemy spawned.

To make the game easier, especially given the younger audience we expected the GBA to appeal to (when RAYMAN was released for Playstation in 1995, the average age of Playstation owners was 22), we did several things. First, we got rid of the “push” when you were hit. Second, we adjusted all the enemy placements so you’d never get hit by an unseen foe.

The biggest thing we did, though, was create a visual representation of the “pings.” They are now represented onscreen by small sprites. When you hit a ping, you know that you’ve set something off. We also used “tings” (the game’s secondary collectible) to guide players on what would otherwise be “leaps of faith” off the edge of the screen. If a player simply aims for the tings, he will never leap to his death. Unfortunately, some critics knocked the game for the many leaps of faith, so it’s possible that the “follow the

tings” mechanic hasn’t been as well understood for platform hopping as we had hoped.

Rearranging platforms was also a major component of the gameplay tweaking. Putting the original levels on the smaller GBA screen would make the game, even with tings, truly unplayable. We tried to be as unobtrusive with the gameplay tweaking as possible, and we’ve been gratified to see many comments in reviews and from players that say the game is exactly like the original, when in fact the layout of every single level has been altered significantly.

5 • Adequate testing. Testing is another element of game creation where there’s just no substitute for hard work. Considering the many changes we made to the levels, we had to test constantly throughout the development process — not just at the end — to make sure that the game maintained the “feel” of RAYMAN. Realizing this early, and doing it, was a major contribution to making sure the game felt, and not just looked, like the original. Associate producer Bill Schmidt tested and adjusted levels for weeks, and platform placements were changed for gameplay and difficulty reasons almost every day up until the game shipped.

The Ubi Soft test team in Montreal was also excellent. Some of the guys would go

through the entire game from start to finish up to four times a day, and if you’ve played RAYMAN, you know what an achievement that is. Sometimes developers and testers have an adversarial relationship, but this was a case where the testers really came through for us in a big way.

What Went Wrong

1 • Multiplayer. We initially planned to include a competitive multiplayer mode in the game. It was going to be a side-scrolling version of capture the flag featuring Rayman and “Dark Rayman.” While the link capabilities on the GBA are one of the platform’s key strengths, the libraries supplied by Nintendo at the time were less than robust.

When Nintendo moved the release date up, we were forced to shelve multiplayer, because testing and debugging were simply taking too long. Coming by extra dev kits — needed for testing two-player mode — was also tough in the prerelease environment, where kits were strictly allocated by Nintendo.

2 • Music. One of the problems with developing for new hardware isn’t just that developers have trouble coming to terms with it — the hardware makers sometimes do too, especially since they create development tools using even earlier

versions of the hardware than what the software developers eventually get. This showed most dramatically in the music driver that Nintendo supplied. Music took up far more CPU time than we were comfortable with, but our short development time prevented us from implementing a custom sound driver. More importantly to players, because the graphics took up so much of the cartridge size, we were only able to include a limited number of songs from the original game. While they sound great, the game certainly would have benefited from a wider variety of music.

The music driver supplied by Nintendo also caused strange crashing problems for Cathryn. It was tough to figure out if it was us or the driver, and without access to the driver source code, there was no real way to debug the issues. When we reduced the amount of data transferred during the vblank, the crashing went away. We didn’t have time to investigate why that solved the problem, we were just happy it did. Needless to say, while we’ve all been there, these kinds of “voodoo” solutions to problems are less than optimal.

3 • Prerelease hardware. One of the biggest things that went wrong was that the hardware we developed the game on was significantly different from the final GBA hardware. Some of our dev kits simply weren’t as capable as final



The colors on the GBA version (right) are much brighter than on the PC (left); it doesn’t look good on paper, but it displayed much better on the GBA screen than the original palettes did.

hardware. More damaging, though, was that their displays were far brighter than the final LCD screen that made it into the production GBA. We assume that the brighter LCDs in our dev systems probably took more power than Nintendo — legendary for its commitment to long battery life — was willing to expend in the system. Whatever the reason, RAYMAN ADVANCE narrowly missed being unplayably dark, as we had spent weeks tweaking and adjusting palettes for the brighter, prerelease screens.

We got Japanese units just a few days before RAYMAN ADVANCE was due to ship, along with some very dark-looking Japanese launch games. Running our game on Japanese launch hardware practically caused a full-fledged panic attack in producer Renée Johnson, who immediately broke the news to Granted that all his palette optimizations would need to be “reoptimized.”

To call Granted’s reaction unprintable is probably an understatement, but the net result was that, several sleepless nights later, all of the palettes had been reoptimized for the production-level GBA. We were slightly worried that Nintendo would re-tweak the screens for the U.S. release, and so we were probably the only people on the planet who cheered when we saw how dark the U.S. screens were. Bad for CASTLEVANIA, but good for RAYMAN!

4 • Cartridge size. The maximum cartridge size available to third parties is 64 megabits, to use the standard

cartridge-size definition. That’s only 8MB — the same size as a memory card for PS2. The original source code alone was 2.5MB, so even when it was rewritten, getting art and music assets into the remaining space was quite an achievement.

Unfortunately, we didn’t have the space to include all the music we wanted, and wrestling with cart size started early and continued throughout the entire development process. Had we been able to be less efficient about space, we could have been more efficient about time, which would have resulted in a less stressful development cycle. It’s something that’s not apparent to players, and it goes with the territory when developing for cartridge-based systems, but it remains a constant challenge. Luckily, Cathryn’s compression and the artists’ smart tiling means that players see the full game just as it was on the Playstation.

A thornier issue was save RAM. We initially thought that we would have a 64K EEPROM for save data, but that cartridge configuration became unavailable, and we had to go with a 4K EEPROM. Unfortunately, this made the save data too large to fit in the EEPROM. We solved this by packing some data that had been stored in bytes or long words into bits.

5 • Unrealistic scheduling. Eight months is a fairly tight development schedule for GBA. Add the uncertainties and false steps that always accompany developing for new hardware, and

the schedule was very short. The truth is that despite our best efforts, our projections for GBA development schedules were woefully inadequate. While we originally estimated GBA development to be twice that of Game Boy Color, it wasn’t until six months into RAYMAN ADVANCE that we realized it’s more like four times that of GBC.

With the absolute inflexibility of the shipping date (a date that actually moved up a few days near the end of the project when Nintendo moved up the release date), and the incredible art conversion requirements, pressure was mounting as the final deadline approached.

Luckily we managed the stress well. Things did get crazy now and then, but overall everyone on the team emerged from the project unscathed, with burnout levels low. The eventual success of the project also helped everyone bounce back relatively quickly.

Looking Back

Bringing RAYMAN to the Game Boy Advance was overall a great experience for Digital Eclipse. Not only did it enable us to get up and running with GBA as soon as possible, but we also considered it a big honor to be the first non-Ubi team to do a RAYMAN game. The original is truly one of the most refined and challenging platform games around, and the challenge of bringing it intact to the GBA was both fun and rewarding. 🍷



RAYMAN’s whimsical graphic style should appeal to the GBA target demographics.



The cart size (8MB) made fitting in the detailed graphics onto the cart quite a challenge.

Some Assembly Required

So what is the future of mobile games? I'm not talking about SNAKE on my mobile phone; I want to know when I will get to play QUAKE on my phone. I see a lot of difficulties ahead, but there's hope that technology will improve quickly, to get millions of mobile device owners playing excellent games on the go very soon.

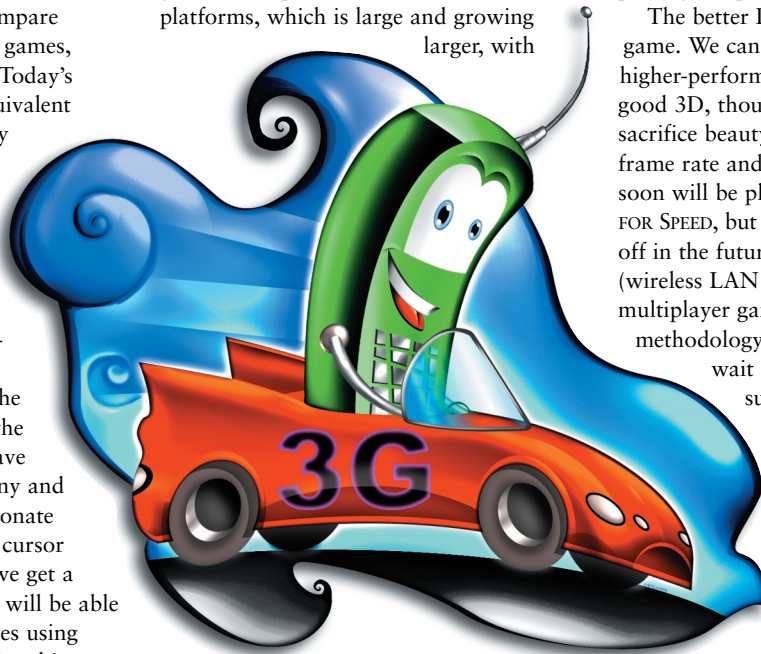
There's an inclination to compare today's mobile games with PC games, which is fair but unfortunate. Today's hottest mobile hardware is equivalent to the PCs of 15 years ago. My cell phone can only dream of being as good as a Commodore 64 one day. Clearly, we won't be seeing PC-quality mobile games for a while.

Mobile devices have a number of impediments to implementing complex games, not the least of which is the fact that the controls suck. Some devices have thumb joysticks, but they're tiny and don't behave well during passionate gaming sessions. There are no cursor keys, and instead of a mouse we get a touch-screen. Now, I think we will be able to create some interesting games using touch, and soon we will start breaking our touch-screens as we stab and slash at them to play our games, but still it won't be as good as a joystick or a mouse.

Furthermore, miserly CPU speeds will limit action, physics, enemy AI, and even rendering. We're a long way from having mobile devices with 3D technology built in (although at Red Jade, we were building just such a device, and I'll bet it's not long before someone tries again). We've got limited audio. No networking means no multiplayer games (though there's room for infrared hacks). Limited RAM means limited game levels, and total game content will be much smaller. We will be counting mobile game sizes in K, not MB, for some time.

Even worse, many game developers have become lazy and fat, because RAM and other resources on PCs are comparatively plentiful and the hardware does all the hard work. When was the last time you counted CPU cycles? Fat and lazy won't work in the mobile game space.

Perhaps the biggest killer to mobile game development is the number of platforms, which is large and growing larger, with



no clear winner in sight. Yesterday's darling is today's has-been. This churning is good for the consumer, but bad for the game developer. How can one make game development profitable when there are so many hardware and OS environments?

So, given these limitations, what games can we make? What religions must we follow? Well, first of all, roll up your sleeves and prepare to get your hands dirty. Developers will not be able to code strictly in C++ or depend on hardware graphics accelerators and such to do their work for them. Mobile game coding will require us to return to the old disciplines. Some assembly required, you bet! With enough

work, however, good 3D is possible in these performance-starved environments. Tight rendering loops can create reasonably high polygon counts with textured surfaces, which look quite good as long as you're not expecting trilinear filtering or curved surfaces. Likewise, although audio will be limited, we can make decent music playing samples with simple mixers.

The better PDAs can handle any 2D game. We can squeeze the heck out of the higher-performing platforms to get pretty good 3D, though developers will always sacrifice beauty as needed for the sake of frame rate and responsiveness to input. We soon will be playing DUKE NUKEM or NEED FOR SPEED, but UNREAL TOURNAMENT is still off in the future. Without network support (wireless LAN or phone), we can't have multiplayer games, and any server-side methodology won't work. We have to

wait for future developments to support real interconnectivity.

We will need new tools, and new paradigms, too. Can't have a full game in memory at once because storage is so limited? No problem: Create that big game and provide a mechanism by which

users can download a subset of levels into the mobile device.

Likewise, the standard distribution and profit models for software to which game developers have become accustomed won't work, especially with too few of any particular target device in the market. We will see more Internet distribution of game content, and games for multiple targets distributed on a single CD. We can solve the problem of too many platforms by creating leveling technology that will make it easy to develop a single game on multiple platforms simultaneously, just as OpenGL solved the problem of too many flavors of graphics hardware.

continued on page 55

continued from page 56

What about the future? In the spirit of going back to our roots, once again there will be kids in garages making killer games on a budget of chips and cola. Costs will be lower, team sizes smaller, development cycles shorter. The mobile arena will allow us to have fun again, to take risks, and to think lean and mean. We will see a return to the idea that game-play is more important than flashy graphics, and I'm looking forward to it. Meanwhile, come on, Moore's law! We're counting on RAM and CPU capacity to double and double again. We need mobile devices to get better LAN support, and we

crave 3G or its equivalent to allow us to play full multiplayer games whenever we feel like it, wherever we are. Microphones and cameras come next, useful for both communication and gaming.

If the device manufacturers would listen to me, they would improve their systems in this order: CPU speed, controls, RAM, graphics acceleration, network support, and then audio (once we get graphics in hardware, we can use more of the CPU for audio). However, they're not yet thinking of these mobile devices as game systems, but as personal productivity enhancers. So what we'll get is improvement in LAN/phone support before CPU speed,

RAM before graphics, better controls later, and finally real audio. Sigh, they never listen to the game developers.

It's a brave old world that's coming, and I'm looking forward to going back there. 🍻

RJ MICAL | *RJ is chief architect at Fathammer, where he is working on the software architecture and development interface for Fathammer's mobile game engine, X-Forge. Previously, he was vice president of software at Red Jade. He was a key member of the Amiga Computer development team, co-invented the Atari Lynx handheld game system and the 3DO entertainment console, and has co-developed more than 15 videogames.*
