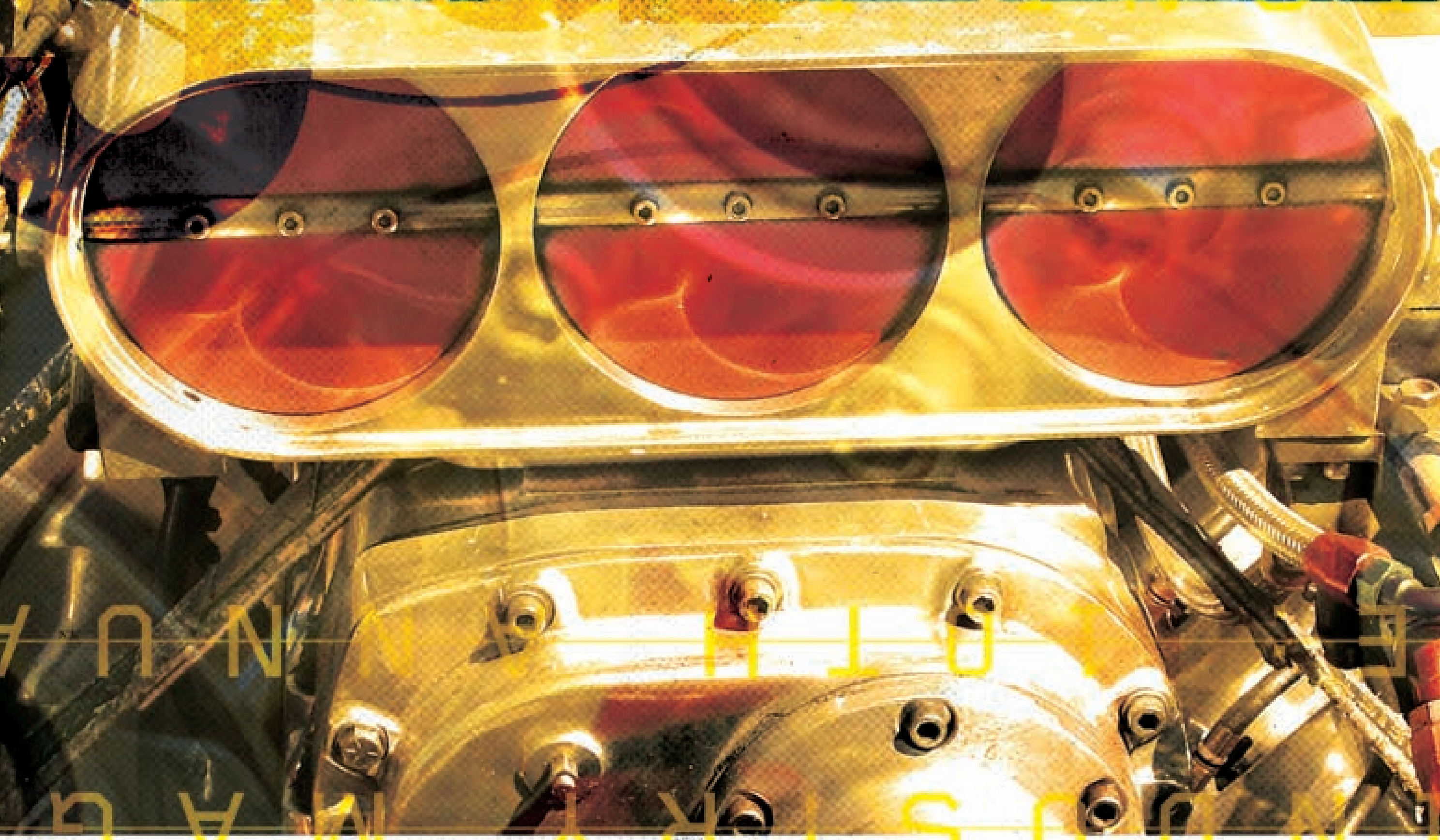
THE LEADING GAME INDUSTRY MAGAZINE VOL18 NO5
MAY 2011 INSIDE: BROWSER TECH MEGA-ROUNDUP



gd GAME DEVELOPER MAGAZINE

XBLIG Postmortem Explosion: Cthulu Saves the World, Shoot 1up, Epic Dungeon, ZP2KX, Soulcaster II

Game Engine Survey 2011

A Salute to Real Game Dev Heroes

havok™

SERIOUS TECH AT PLAY

WORLD CLASS SUPPORT | NOW AVAILABLE ON SMARTPHONES | PROGRAMS FOR ALL BUDGETS

Learn more about our cutting edge game technologies:
blade@havok.com

CONTENTS.0511
VOLUME 18 NUMBER 5

GAME ENGINE 2011 INSIDE

ACCEPTING FREE-TO-PLAY

LEARNING TO FLOW WITH THE CHANGING TIDE

IN THIS MONTH'S DESIGN COLUMN, you'll find Soren Johnson talking about free-to-play game design. He recently completed work on Dragon Age Legends, an F2P realization of the Dragon Age world for Facebook. In the column he mentions several successful implementations of F2P design, wherein designers must keep players interested and engaged on a moment-to-moment level, but also provide them with compelling reasons to spend money. The reason this topic is so valid is that a lot of game industry folks, myself included, just get plain old queasy when we think about creating games this way.

MONEY TALKS

>> There's a perception that the powerful metrics and feedback provided by these platforms turns designers into spreadsheet managers. Though it may feel that way on some level, it's not totally true, since you have to devise the systems in the first place, then test them extensively. It's like getting the ultimate feedback loop, and if your goal is to please players, there's nothing better than instantly knowing whether your new system is a success.

More than that, one of the stickier issues is the (perhaps unfounded) idea of paying to unlock the "fun" of the game. In League of Legends, for instance, the character roster rotates weekly, and if you want to specialize in one of them, you have to either pay real money to keep it unlocked, or play long enough to save up in-game money to unlock that character. There's nothing inherently bad about this! But just thinking about it makes me feel like it's "wrong." My gut tells me that all characters should be available from the get-go, and players should have a choice of who to use. It makes me even more uncomfortable to know that you can buy an item with real money that helps you gain in-game money faster. But you know what? I'm sort of wrong to feel this way.

First of all, a rotating roster inspires players to try out characters they might otherwise never touch. When presented with some 72 characters at once, it's difficult to know where to start, and one is less inclined to experiment. In that sense, the designers have chosen to showcase all their characters individually, because otherwise a good percentage of them may have gone unused. The game also rolled out these characters gradually over time, making the rotation feel natural for long-time players.

I think that some of us feel designers of these games build a complete experience, and then decide what they can chop out to make people pay for. That's not how the most successful games are designed, necessarily. They take a core experience that has the idea of payment integrated into it as it's designed. The preceding sentence made me uncomfortable even as I typed it, but all commercial games are paid experiences somehow or other. Whereas many of us still make a game and sell it once, these games keep on selling—but the idea of selling our games to customers stands firm in both cases.

It's important to remember that the best among these games really are free. You could play League of Legends for free forever, if you had the time to invest. Almost everything except the visual flair items is purchasable with in-game money, earned by playing over time. This is just like the "unlocks" we've been building in games for decades. But the player never had to pay, and they never pirated the software.

In most cases, paying enhances the experience for players by making it more convenient, and that's another troublesome point. If the designers know how to make a game more accessible, more fun, and more convenient for players, why should that be behind the pay wall? The answer is that if you don't, nobody will pay. But there's no way around the queasiness some of us feel knowing we're withholding convenience that could have been built into the game natively.

GIVE THEM WHAT THEY WANT

>> I've been casually following the F2P industry for nearly as long as it's been around. I started paying attention to the Korean industry back in 2001, and the seeds planted then have taken fruit worldwide. Free-to-play is quite simply one of the most lucrative business models, if not the most, in games today. This has expanded to include much greater Western appreciation, and looks to become a dominant model for online games here as well.

I mention all this because these games aren't just being forced onto us by a bunch of executives. I've watched the industry grow, and these games are succeeding because players are paying for them. They're voting with their wallets and showing us that they like these games. This is, in fact, something they want to play, and a payment method they feel good about. Those who dissent and rail against this change are a vocal minority.

I would love to see more artistry and narrative design in the free-to-play market, but we have to make that happen. If you want to create a narrative in a traditional setting, there will continue to be a way for you to make a comfortable living doing so. But really, those of us who don't like this change, those of us who want to preserve traditional narrative and gameplay, it's up to us to make the F2P model work on our terms. Customers want to play this way. If we want to preserve traditional design, we are going to have to meet them halfway.

Not all games have to be free-to-play, but likewise, not all free-to-play games have to make us uncomfortable. If we want to help shape this industry, we've got to do more than just complain about it!

—Brandon Sheffield
twitter: @necrosofty

GAME DEVELOPER MAGAZINE
WWW.GDMAG.COM

United Business Media
303 Second Street, Suite 900, South Tower
San Francisco, CA 94107
t: 415.947.6000 f: 415.947.6090

SUBSCRIPTION SERVICES

FOR INFORMATION, ORDER QUESTIONS, AND ADDRESS CHANGES
t: 800.250.2429 f: 847.763.9606
e: *gamedeveloper@halldata.com*

FOR DIGITAL SUBSCRIPTION INFORMATION
www.gdmag.com/digital

EDITORIAL

PUBLISHER
Simon Carless I *scarless@gdmag.com*
EDITOR-IN-CHIEF
Brandon Sheffield I *bsheffield@gdmag.com*
PRODUCTION EDITOR
Jade Kraus I *jkraus@gdmag.com*
ART DIRECTOR
Joseph Mitch I *jmitch@gdmag.com*
DESIGNER
Jessica Chan
PRODUCTION INTERN
Tom Curtis
CONTRIBUTING EDITORS
John Graham
Jesse Harlin
Soren Johnson
Vincent Scheib
Steve Theodore
Matthew Wasteland
ADVISORY BOARD
Hal Barwood **Designer-at-Large**
Mick West **Independent**
Brad Bulkley **Neversoft**
Clinton Keith **Independent**
Brenda Brathwaite **Lolapps**
Bijan Forutanpour **Sony Online Entertainment**
Mark DeLoura **THQ**
Carey Chico **Independent**

ADVERTISING SALES

GLOBAL SALES DIRECTOR
Aaron Murawski e: *amurawski@think-services.com*
t: 415.947.6227
MEDIA ACCOUNT MANAGER
John Malik Watson e: *jmwatson@think-services.com*
t: 415.947.6224
GLOBAL ACCOUNT MANAGER, RECRUITMENT
Gina Gross e: *ggross@think-services.com*
t: 415.947.6241
GLOBAL ACCOUNT MANAGER, EDUCATION
Rafael Vallin e: *rvallin@think-services.com*
t: 415.947.6223

ADVERTISING PRODUCTION

PRODUCTION MANAGER
Pete C. Scibilia e: *peter.scibilia@ubm.com*
t: 516-562-5134

REPRINTS

WRIGHT'S MEDIA
Ryan Pratt e: *rpratt@wrightsreprints.com*
t: 877.652.5295

AUDIENCE DEVELOPMENT

TYSON ASSOCIATES Elaine Tyson
e: *Elaine@Tysonassociates.com*
LIST RENTAL Merit Direct LLC
t: 914.368.1000

United Business Media
WWW.UBM.COM

around the Arab world there are more than **180 million** people under the age of 25*.

yet Arabic game development is still in its infancy.

your opportunity is right here, right now at twofour54° in the heart of Abu Dhabi.

The Arab world is one of the world's fastest growing media markets. With a young population of 180 million under the age of 25, more than 80% with mobile phones*, strong broadband take-up and new gaming innovations, it's a prime opportunity for Arabic gaming businesses.

We empower businesses across all media platforms from production, gaming, digital, animation, broadcast and publishing – with world-class training from **twofour54° tadreeb**, state-of-the-art production facilities with **twofour54° intaj** and venture funding and support for Arab creative entrepreneurs from **twofour54° ibtikar** – to seize every media opportunity the region has to offer.

It's all part of our vision at **twofour54°**, creating a centre of excellence for Arabic content creation in Abu Dhabi.

we are twofour54°. are you?
find us. join us. create with us.
+971 2 401 **2454** **twofour54**.com

twofour54°
Abu Dhabi

content creation community

Sources: Arab Media Outlook 2010. Media on the Move 2009. A.T. Kearney. Introduction to Gaming. Michael Moore. Screen Digest. IDC.

The Video Game Crowdfunding Cheat Sheet

\\\ Crowdfunding is a natural fit for an independent game developer who needs to connect with an audience and secure funding at the same time. Rather than having to "prove" your game to a publisher, you're "proving" it directly to your customers, and you don't have to go out on a limb funding it yourself with no idea of whether you'll sell enough copies to recoup your investment. But not all crowdfunding is the same.

An important distinction is the funding model, which divides these services into two different camps. "All or nothing" funding means that you get the money raised only if you've met your funding goal by the funding deadline; otherwise, no money changes hands. This is a good fit for projects where you have to raise a specific amount of money to pay for an engine or assets, and you'd rather get no funding at all than get half the funding you need and try to cobble together your game with that. "Keep it all" funding means that you immediately get any money your project raises as soon as it's contributed (minus fees), regardless of whether you've reach your funding goal. It's a good fit for projects where you're committed and able to make the game regardless of how much money is raised, especially if cash is needed right away to feed yourself and pay rent while you work on the project.

Both funding models encourage projects to re-apply if they don't reach their funding goal. A failed "all or nothing" project can often succeed by scaling down and re-applying with a smaller funding goal, and a failed "keep it all" project can re-apply with the same funding goal minus any money that the project has raised in previous "rounds" of funding. The chart that accompanies this article shows the major crowdfunding services, as well as their relevant details. A larger version of this report, including comments from the organizations themselves, will be available on Gamasutra.com in April.

—R. HUNTER GOUGH

ILLUSTRATION BY ROBIN ALLUM-CORNFORTH

NAME	KICKSTARTER http://kickstarter.com	INDIEGOGO http://indiegogo.com	ROCKETHUB http://rockethub.com	ULULE http://ulule.com	8-BIT FUNDING http://8bitfunding.com
LAUNCHED	April 2009	January 2008**	January 2010	October 2010	January 2011
SUCCESSES	6,500+	400 (22,000 total)	94	80	1
VG SUCCESSES	67	1	1	3	Same
FUNDING MODEL	All or Nothing	Keep It All	All & More (Keep it All)	All or Nothing	Keep It All
SERVICE FEE	5%	4% (9%*)	4%(8%*)	0%	5%
PAYMENT FEE	3-5%	3%	4%	3%	3%
TOTAL FEE	8-10%	7%(12%*)	8%(12%*)	3%	8%
PAYMENT METHOD	Amazon	PayPal/credit card	Credit Card	PayPal	PayPal
PERKS	Popularity and Curated Packages	Partners	Badges and Opportunities	Low Fee and Message Board	Video Games Only
INTERNATIONAL	No	Yes	Yes	Yes	Yes

* HIGHER FEE FOR UNSUCCESSFUL PROJECTS **FILM PROJECTS ONLY UNTIL JANUARY 2010

Game Developer magazine wins a Silver Eddie

\\\ The editors of *Game Developer* are pleased to reveal that the magazine has won a Silver Eddie Award for our March 2010 issue which featured an UNCHARTED 2 postmortem, as well as the second edition of the "Dirty Coding Tricks" article series. The Eddies are put on yearly by Folio, which bills itself as "the magazine for magazine management," and celebrate the best in editorial and design for the magazine publishing industry.

In the B-to-B space, *Game Developer* was a silver winner in the category Media/ Entertainment/Publishing Full Issue. The editors wish to thank the contributors and authors that made the award possible.

—STAFF

Game Developer donates to Strong National Museum of Play

\\\ The National Museum of Play is an exhibition devoted to all types of play, from toys and dolls to board games and video games. The museum is hosted at the Strong in Rochester, NY, and already boasts a large archive of video and computer games, design documents, magazines, and more.

The video game division is known as the International Center for the History of Electronic Games, which is already home to over 25,000 video and electronic games, including over 1,000 PC games, and almost the entire run of *Computer Gaming World* magazine, which was recently donated by Frank Cifaldi of 1up.com.

To aide the museum's efforts, *Game Developer* magazine has donated a near-complete archive of its back catalog, along with a subscription going forward into the future, so that the sharing of game development knowledge from throughout the ages can be preserved for future generations.

Developers interested in donating historical artifacts or collections can contact the International Center for the History of Electronic Games and its director Jon-Paul Dyson through its website at www.icheg.org.

—STAFF

Gamestorm celebrates paper prototyping

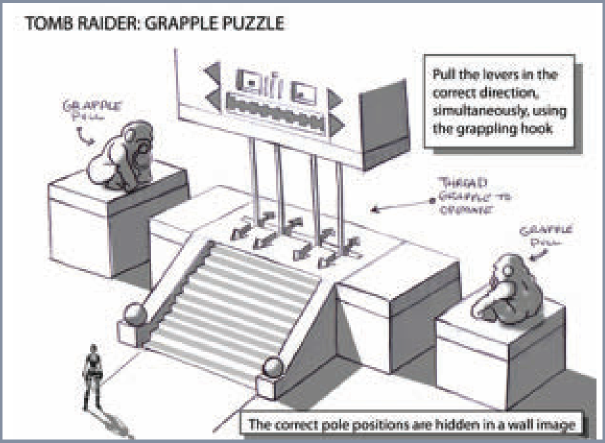
\\\ A lot of developers sketch out their ideas, whether that sketch is a bullet-point list of plot points or an entire level drawn out in meticulous detail. Although prototyping in code is a great way to prove a concept, many developers still choose to germinate their ideas with paper and pen (or tablet and stylus, as the case may be). It's this stage of development that has aroused the interest of FLASHPUNK creator Chevy Ray Johnston, who created the Gamestorm blog to gather and showcase the thoughtful sketches of game developers around the world.

"While developing games, almost all my ideas (whether they be art, programming, math, or layouts) find their way onto paper before they ever hit the computer screen," Johnston told us. "I love looking back at older pages, whether it be traveling back in time to the state of mind I was in, or simply trying to decipher what each mess of scribbles, notes, arrows, and doodles even mean. Brainstorming notes are amazing pieces of art to me, especially where game developers are concerned, because they deal with problems and designs of such fun and eclectic types."

The genesis of the blog came from a Global Game Jam he participated in this year, where brainstorming pages were in steady supply. It's those pages that were first featured, but the project soon grew. "I launched the site initially planning to just keep uploading new doodles and sketches," he said. "I have a lot of developer friends, and also help teach students of game development programs, so I was in no shortage of source material, but I added a submissions page anyway."

It turns out he's not the only one with interest in both viewing and sharing this information. "The idea really struck a chord, and just under 24 hours, I already had over 100 submissions from developers all over the world," Johnson said. While the site has only been up for a month and a half as of this writing, well over 100 documents have been shared, from amateur and professional game developers alike, with no signs of slowing down. Why has this captured the interest of so many? Johnston says it best: "Nowhere else can you see complex math equations next to an astronaut riding a dinosaur, or grocery lists on world maps." Visit and submit entries to Gamestorm at http://gamestorm.tumblr.com.

—BRANDON SHEFFIELD


TOMB RAIDER: GRAPPLE PUZZLE
GRAPPLE PULL
Pull the levers in the correct direction, simultaneously, using the grappling hook
THREAD GRAPPLE TO OPENING
GRAPPLE PULL
The correct pole positions are hidden in a wall image

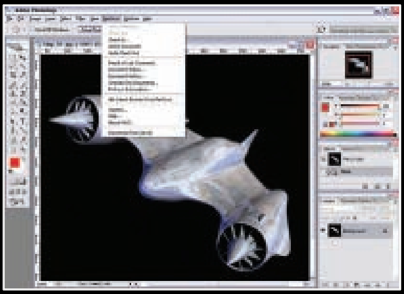
corrections

\\\ In the April 2011 issue, we misspelled Autodesk in our Tool Box section. Additionally, we neglected to mention that a version of Joshua Tippetts' article, "Creator of Worlds", originally ran on GameDev.net. The editors regret the errors.

Perforce | *Fast* Software Configuration Management

Introducing P4GT,

a productivity feature of Perforce SCM.



P4GT

The Perforce Plug-in for Graphical **Tools,** P4GT, makes version **control** painless by **seamlessly integrating Perforce** with **leading graphical tools. Drop-down menus allow access to Perforce from within 3ds Max, Maya, Softimage XSI, and Adobe Photoshop.**

Art and development teams can standardize on Perforce to version and manage both source code and digital assets. Enhanced collaboration during the design process helps teams to work together in real time to release small patches or create whole new worlds.

P4GT is just one of the many productivity tools that comes with the Perforce SCM System.

PERFORCE
SOFTWARE

Download a free copy of Perforce, no questions asked, from www.perforce.com. Free technical support is available throughout your evaluation.

WHAT DO GAME DEVELOPERS REALLY THINK OF LICENSED GAME ENGINES?

game engine survey 2011

MARK DELOURA

One of the most remarkable changes in the game industry the past few years has been the massive growth of casual, mobile, and social games. A few years ago I gave a presentation at CoFesta in Tokyo lamenting how difficult it was for small independent developers to distribute and sell their games—that's certainly not the case anymore! Now, it's more difficult to decide which of the many, many platforms and marketplaces to create your game for, because there is so much opportunity for independent developers, especially on iOS and Facebook.

With such huge changes in the industry, it seemed like the perfect time to revisit the game engine survey. I was curious to see how the change in the game development ecosystem would be reflected in the distribution of survey responders, and the information they shared about their use of game engine middleware.

When the survey was complete, the change in the make-up of responders was quite telling. Compared to 2009, a roughly equivalent number of developers who responded were working on big-budget titles, but about three times that number reported to be working on smaller titles! I've split the survey responders into two groups, as in some cases their survey answers differ in very interesting ways. The needs of a developer working on a social or mobile title are somewhat different than that of traditional game developers, due to the smaller size of games, shorter production length, or need for frequent updates. For the purposes of this survey we will refer to developers working on larger-budget titles, those on consoles and handhelds, as the *traditional* developers, and we'll call those working on casual, mobile, and social titles the *casual* developers.

FIGURE 1

PLATFORMS

/Traditional platforms/

PS3	75.7%
Xbox 360	73.9%
Wii	20.0%
PC	66.1%
Mac	14.8%
Linux	3.5%
NGP	7.0%
PSP	6.1%
3DS	10.4%
NDS	7.0%

There's quite a large number of game engines available for developers to use, from licensed high end to free open source, so the engines covered in this survey are those most well known or used. They are all also multi-platform across PC/Mac and consoles, all licensable (not completely free for commercial use), and are all available for license without an agreement with a platform manufacturer. The full list is in the sidebar to the right.

Survey Responders

/// The developers I'll concentrate on for most of this survey make traditional big-budget games. These games largely target the PC and high-end consoles (see Figure 1) as most large-budget titles are produced for the PC, PS3, and

ENGINES FEATURED

Blitz Games' BlitzTech
www.blitzgamesstudios.com/blitztech

Crytek's CryEngine
www.crytek.com/cryengine

Digital Extremes' Evolution Engine
www.digitalextremes.com/evolution

Gamebase's Gamebryo
www.gamebryo.com/en

Epic Games' Unreal Engine
www.unrealengine.com

GarageGames' Torque
www.garagegames.com

Stonetrip's ShiVa3D
www.stonetrip.com

Terminal Reality's Infernal Engine
www.infernalengine.com

Trinigy's Vision Game Engine
www.trinigy.net

Unity Technologies' Unity
http://unity3d.com

Valve's Source Engine
http://source.valvesoftware.com

Vicious Cycle Software's Vicious Engine
www.viciousengine.com

FIGURE 2

ARE YOU USING A LICENSED ENGINE ON YOUR CURRENT PROJECT? WHICH?

TRADITIONAL	
Yes	58.7%
No	41.3%
Unreal	27.8%
Trinigy	19.6%
Unity	11.4%
CryEngine	8.2%
Gamebryo	6.5%
CASUAL	
Yes	83.7%
No	16.3%
Unity	26.0%
Trinigy	17.8%
ShiVa	8.8%
Torque/Unreal	3.0%

FIGURE 3

GAME ENGINES USED

TRADITIONAL	
Unity	61.8%
Unreal	60.7%
Gamebryo	29.2%
CryEngine	20.2%
Torque	18.0%
CASUAL	
Unity	68.4%
Unreal	43.7%
Torque	38.9%
ShiVa	24.3%
Trinigy	23.2%

FIGURE 4

ENGINE AWARENESS

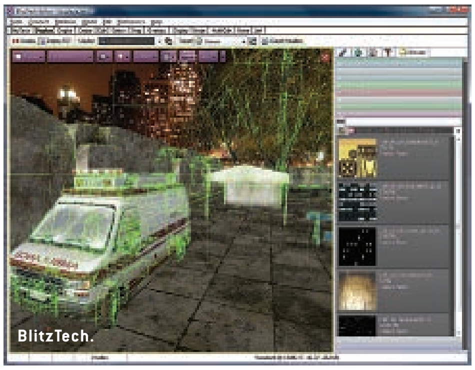
/Ordered/

TRADITIONAL

1 / Unreal
2 / Unity
3 / CryEngine
4 / Source
5 / Gamebryo

CASUAL

1 / Unity
2 / Unreal
3 / Torque
4 / CryEngine
5 / ShiVa


BlitzTech.

Unreal.

Xbox 360. These machines have roughly the same technology level, so the amount of art and technology retargeting between the platforms is minimized, and the potential audience is maximized. Most games at this level ship on all three platforms, though some go for just the consoles, and generally first-party platform-owned studios are the only ones who can really afford to specifically target just one of the high-end platforms.

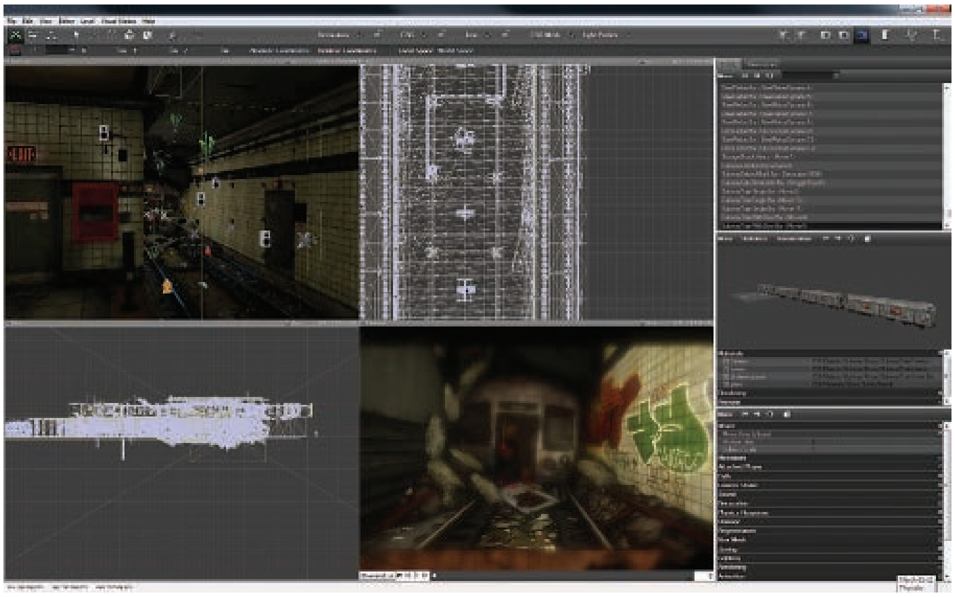
Development for the Mac is slowly picking up, whether that's done through cooperation with companies like Aspyr and TransGaming or as part of an in-house pipeline parallel to the PC version's development. In 2009, only 2% of these developers reported working on a Mac version, versus nearly 15% in this year's survey.

Rough development costs for games produced by the traditional group have a broad range. Average development costs for a title in this group are between $15 and $20 million, with 25.6% claiming costs over $20 million, and 7.2% over $40 million.

Use and Usefulness

/// What are the most popular engines in use today? We asked both the traditional and casual developers what engines they're using on their current projects. Of traditional developers, 58.7% are using a licensed game engine, up from 55% in 2009, with the largest number of those using an engine (27.8%) using Unreal. Among casual developers, a substantially larger number use a licensed game engine (83.7%) with 26% of those using Unity, the most popular choice. (See Figure 2.)

Which engine they're using now doesn't reflect all the engines they've ever worked with. We asked about that as well (See Figure 3.), and among both traditional and casual developers, the engine most used is Unity: 61.8% of traditional developers have used it, and 68.4% of casual developers. This is a big shift from two years ago, when Unreal was the clear winner. Now Unreal runs a close second for traditional developers at 60.7%, and third place is quite a distance away. Unity's business model, ease-of-use, and multi-platform nature are paying dividends here with all developers, not just casual developers, enabling people to try out the engine for fun or for side projects. It's a smart strategy that has resulted in a big boost to the


Evolution.

number of both traditional and casual developers actively using Unity on projects.

Unity and Unreal are the most used engines, but I was also curious to know which licensed engines developers have heard of, and which they think might be valuable for their current projects. In the traditional space, Unreal, Unity, and CryEngine are believed to be most valuable for current projects, with Unity, Unreal, and Torque making up the top three among casual developers. (See Figure 4.) In terms of perceived usefulness, Trinigy emerged among both groups of developers, showing up third among traditional developers and first among casual developers. In 2009, CryEngine had a similarly strong showing in the survey results, and we wondered why more games weren't being made with it. Perhaps Trinigy's embrace of component middleware libraries and their recent launch of WebVision are helping them gain mindshare among developers. (See Figure 5.)

I have to note that a surprising number of C4 Engine users wrote into the survey comment fields this year about how much they like using that game engine. The C4 Engine is created by Terathon Software and founder Eric Lengyel, creator of the Game Engine Gems book series. C4 Engine wasn't covered by this survey since it is exclusive to PCs and Macs. However, if you need an engine for just these platforms, you should know that we received a lot of feedback from happy users.

FIGURE 5

PERCIEVED USEFULNESS

Scale of 1 to 5

TRADITIONAL	
Unreal	3.35
CryEngine	3.15
Trinigy	3.12
Evolution	2.56
Unity	2.45

CASUAL	
Trinigy	3.46
Unity	3.43
Unreal	3.22
ShiVa	3.02
CryEngine	2.61

Engine Financials

/// In 2009, our survey asked developers what they typically expected to pay for a game engine, and the responses varied widely. The same is true this year. It's unclear whether this is due to a lack of awareness of the price of a licensed engine, or the growing disparity between the price of casual engines versus high-end engines. That said, 41.2% of the developers who responded to our survey hoped to pay less than $100K for a game engine, for all platforms that their game is shipping on. Certainly this is limiting them to a very small subset of available engines, as there are just a few in that price bracket—and even fewer that target the high-end consoles. Significantly fewer respondents (21.9%) were looking to pay from $100K to $500K, and 9.6% from $500K to $1 million. Only a small fraction of survey responders, 9.7%, expected they'd need to spend greater than $1 million for the engine they're using for their game. However, for the high-end game engines, a three-platform game (PC, PS3, and Xbox 360) does typically cost more than $1 million. With a growing number of strong and less-expensive licensable game engines these days, perhaps we're all just hoping to save a little money.

Of course, a flat-rate purchase price up front isn't the only way to license a game engine. While it is the most traditional way, other possibilities have been

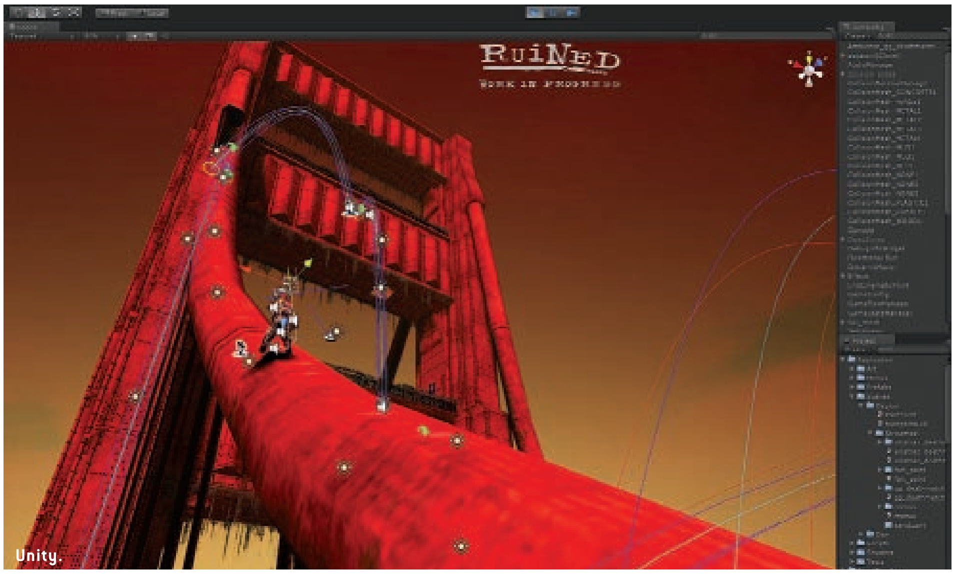
FIGURE 6

PREFERRED WAY TO BUILD GAME TECHNOLOGY

TRADITIONAL	
Use our own engine framework and purchase a suite of middleware	38.0%
Create all tech ourselves	34.0%
Purchase a complete game engine	22.0%
Purchase a game engine that uses other middleware libraries	6.0%

CASUAL	
Purchase a complete game engine	60.3%
Purchase a game engine that uses other middleware libraries	17.2%
Create all tech ourselves	12.9%
Use our own engine framework and purchase a suite of middleware	9.5%

Vicious.



Unity.

created as the number of small and independent developers has grown. These developers are more price-sensitive at the beginning of a project, yet still need great technology at a great price. Paying back-end royalties is one way to achieve this. Paying royalties allows for a smaller cash outlay at the initiation of a project, and then a percentage of revenue from sales of the game. However, royalties are not an option many publishers appreciate. Overall, 80.7% of traditional developers prefer the flat-rate model, compared to 74.5% of casual developers. Survey responders who commented on this question encouraged game engine manufacturers to move toward a hybrid model: a free trial, then a small flat-rate fee up front, with either an adjustable royalty as game sales pass certain revenue targets or, if the game is picked up by a publisher, an upgradable license fee based on the game's budget.

What other costs are developers looking at with licensed game engines? Responders reported that the biggest hidden cost is in adapting engine features to suit the particular game they're creating (86.8%), so flexibility and ease of modification are key qualities. Employee training costs were also called out as quite substantial (67.9%), in terms of both time and money. This definitely encourages developers to go with the more popular engines on the market, so that hiring staff with previous experience becomes easier.

Capabilities

/// Let's step back a bit from specific engines. Since game technology has evolved over the past two years, perhaps what people expect of game engines has also evolved. To start uncovering some answers, the survey posed the question: "How would you optimally like to build your game technology? Would you rather buy an all-in-one engine, build all your technology yourself, or do something in between?" Back in 2009, we saw that just over 9% of survey responders wanted to use a full game engine; the most popular choice among developers was to create all the technology themselves (46.5%). This year it's still true that licensing a full game engine is not ideally what most people want to do. (See Figure 6). While a much larger percentage of traditional developers now want to purchase a complete game engine (22%), what most are still interested in is building the engine framework themselves, and plugging specific middleware libraries into it based on the particular needs of their game (38%). Casual developers had a quite different reaction: over 60% wanted to purchase a complete game engine.

Game engines are definitely continuing to grow in acceptance, but why are traditional developers still wary? Developers using game engines see some clear benefits (see Figure 7), yet there are undoubtedly things to watch out for (as seen in Figure 8). Worries over code quality and architecture, a lack of support or documentation, and dependencies on other technologies that are difficult to identify during the vetting process are some common concerns.

The most important thing when looking for a game engine is to do a thorough job examining all of the potential candidates. Each engine has its own plusses and minuses, and they're different for different platforms and genres. But developers in our survey were clear about the most important practices for all engine providers. Out of a scale of one to five, with five being the highest, these were the highest voted practices: provide source code (4.35), offer live preview on target platforms (3.98), and provide ongoing access to builds in development (3.77). The need to integrate easily with other middleware libraries, as mentioned previously, is also quite important to developers (3.67).

As far as systems and tools within an engine, multi-processor / multi-threading capability popped up this year as the most important (4.43), just as it did in 2009. The current generation of consoles brought this technology to the foreground, since to gain maximum performance from those machines it has been incredibly important to master the art of multi-threaded programming. Other important systems for engines include physics (4.15), streaming (4), profiling (3.92), and camera control (3.69). These are all very sensible demands.

A newer technique that survey responders found important to include is deferred rendering (3.34), as opposed to forward rendering. Developers commented that "...all these systems should be easily replaceable with custom [technology] / middleware." What if your game engine's physics system doesn't have the

FIGURE 7

ENGINE BENEFITS

/Scale of 1 to 5/

Our artists/designers can begin working right away without waiting for tech	4.20
Allows us to focus on game-specific code	3.98
State-of-the-art tech from engine specialists	3.72
We can develop our game more quickly with less tech to develop	3.71
Time to first working version of game is significantly reduced	3.71

feature set I need for the game I'm making, but everything else is perfect? It'd be sad to have to choose a different engine or do a lot of heavy lifting to replace the physics. Ideally, the engine developer should provide simple integration with Havok, PhysX, or Bullet.

The tools our survey responders found important are also all fairly logical: a standalone world editing tool (4.21) and a particle system editor (4.02) got top votes. Developers also look for engines to provide a run-time script debugger (3.6), shader editor (3.56), and cinematics tool (3.53). New to the list of priorities, but still important, are an animation blend tree editor (3.51) and global illumination solver (3.06). Clearly some of these tools can be purchased separately and integrated by the developer, but the stronger engines should provide most of them.

Back in 2009, 73.2% of survey responders noted that they use a standalone world editing tool, as opposed to using their DCC tool for world editing. This year that number has increased to 84.8% of traditional developers, so it's understandable that this would be rated the most important tool for a game engine to provide. In the casual space though, using a DCC tool is more common, and only 64.2% are using a standalone application. Leveraging your DCC tool for world building is an inexpensive way to go, for sure, but as game level sizes or team sizes increase, using the DCC tool in this way can quickly become unwieldy.

Another hot button issue in this year's survey was the importance of having a runtime script debugger. The use of scripting languages in games has continued to increase, and Lua (44.7%) and visual scripting tools (31.9%) are the popular leaders in this regard. One of the most important reasons to implement scripting is to enable rapid iteration, yet many game engines don't offer real-time script preview or run-time debugging. Over 50% of developers rated "live preview of scripting" a 5, or "most important." As one developer commented, "It's a shame that almost no engine in the industry fully supports this yet. Fast iteration time is *the* most important factor for any kind of tool development we do in-house."

Tool and Language Use

/// As part of the survey, I like to find out what other tools, languages, and processes game developers are using in their engine development. Through this anonymous sharing we can all learn about some of the more esoteric tools available, and we can showcase some solutions that most developers might not be aware of.

Respondents made clear that they like being able to

FIGURE 8

ENGINE CONCERNS

/Scale of 1 to 5/

Availability of engine source code	4.16
Code quality and architecture	4.09
Lack of support or documentation	3.67
Engine has dependencies on other assumed tech so it is difficult to extend	3.62
Incomplete feature set across multiple platforms	3.62

Geomerics

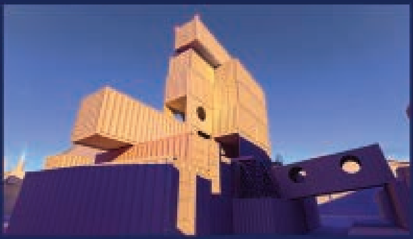
Enlighten delivers world class lighting in real time.

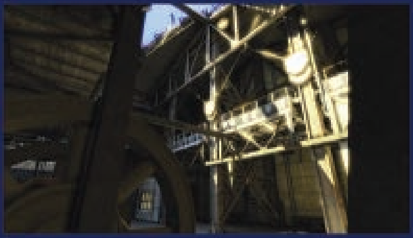
This year the leading games you'll be playing will be built with Enlighten technology.

Stay ahead of the game. Upgrade your engine's lighting pipeline with Enlighten.

ENLIGHTEN

www.geomerics.com





Infernal.

CryEngine.

select middleware libraries to incorporate into their preferred game engine. For traditional developers, Scaleform and Bink are the kings of runtime middleware, both of which solve particular problems extremely well (user interface design and video playback, respectively). For casual developers, the most popular middleware libraries are both good and inexpensive, or they are free: FMOD, Bullet, and PhysX are the most popular in this category. In general, far fewer casual developers (48.6%) use middleware libraries, versus 91.5% of traditional developers. (See Figure 9.)

What about DCC tools? The popular conception is that 3ds Max and Maya are the most popular 3D tools out there, but there are other options as well. Maya turns out to clearly be the most widely used DCC tool for traditional developers (71.7% versus 3ds Max's 45.7%), yet for casual developers 3ds Max (46.7%) is just slightly more popular than Blender (42.9%). A large variety of tools exist in this space, with some of the other frequently used tools being ZBrush, Mudbox, SketchUp, LightWave, Nevercenter Silo, and Modo.

When developers make their own tools, what languages do they find themselves drawn to? I was curious this year whether C# was still the most popular. Sure enough, 71.7% of our responders attested to C# as being the language they are most productive in for tool development. C++ runs a close second (69.6%), with Python, Lua, and PHP filling out the top five favorites.

How about in the early stages of a project for prototyping? At the beginning of a project the last thing you want to do is spend all your time knocking out tools. You need a language or tool you are very familiar with, that you can use to quickly put together demos to prove out your design. Pencil and paper will always rule for prototyping of course (73.9%), although C++ apps follow a close second (71.7%). Other popular systems for prototyping include previously used game engines, Lua, C#, and Flash. For casual developers, Unity is also a noticeably strong choice for prototyping.

Automation

/// One of the clearest differences that showed up between traditional developers and casual developers in the survey is the degree of automation in their production process. Among traditional developers that responded to our survey, 67.4% now use continuous integration systems like Jenkins or CruiseControl, with 69.6% using automated build systems of any kind. These are less important techniques for casual developer teams, but are still valuable practices.

Continuous integration systems that kick off new builds on a build farm each time someone checks in code, and automated overnight build systems that ensure clean fresh builds are available each morning are systems which ease development with complex codebases. Among traditional developers that responded to our survey, 69.6% use automated build systems and 67.4% now use continuous integration systems, like Jenkins or CruiseControl. Casual developers use these types of systems much less frequently, with only 12.4% reporting use of automated build systems.

Automated testing is another way to make large-scale development more straightforward. Tests which run automatically on each build, such as unit tests, gameplay tests, visual comparisons, and stress tests, can quickly pinpoint the introduction of a bug or other undesired change in the codebase. Automated tests are less popular than automated builds among traditional developers, but still used fairly frequently (58.4%). Casual developers, on the other hand, use automated testing more frequently than automated builds (23.1%), but it is still fairly uncommon.

Engines on the Rise

/// Licensed game engines continue to increase in popularity, especially among casual developers. The increase in popular of Unity over the past two years has been remarkable, and Trinigy is making strong gains in mindshare among both traditional and casual developers. Still, it's clear that using a licensed game engine is not for everyone, as many would rather build either the technology framework or the entire game engine themselves. What's important is to be aware of the benefits and limitations of using engine middleware, and to thoroughly vet the engines you're considering. Ultimately, we're all trying to create better games less expensively and with higher fidelity. Licensed game engines are an increasingly popular tool for developers working hard to realize their vision. GD

MARK DELOURA *is VP of technology at THQ, creator of the* Game Programming Gems *series of books, and was formerly editor-in-chief of* Game Developer *magazine. Follow his tweets at @markdeloura.*

FIGURE 9

TOP 5 MIDDLEWARE LIBRARIES USED

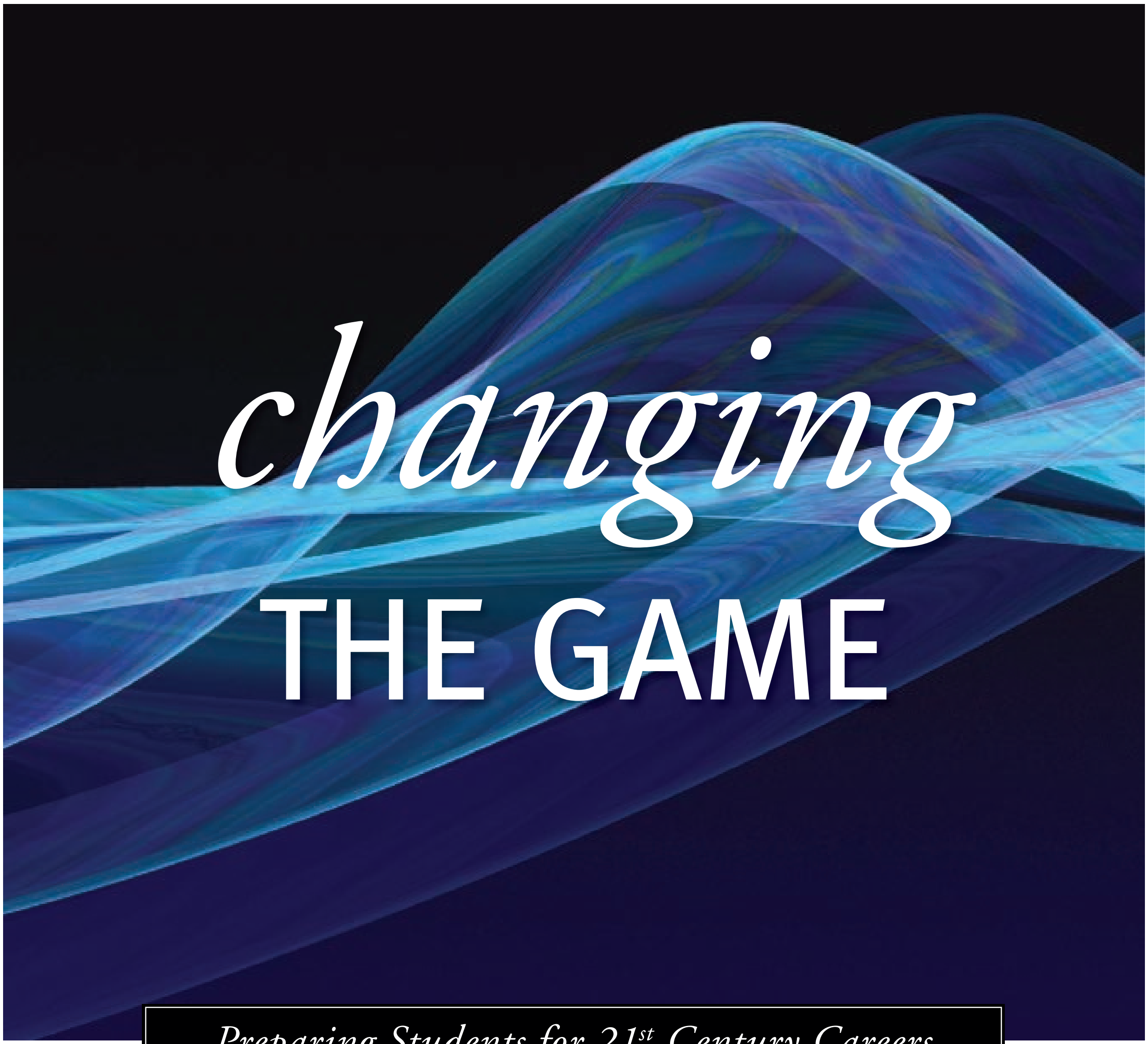
/Ordered/

TRADITIONAL

1 / Scaleform
2 / Bink
3 / FMOD
4 / PhysX
5 / Wwise

CASUAL

1 / FMOD
2 / Bullet
3 / PhysX
4 / SpeedTree
5 / RakNet



Preparing Students for 21st Century Careers

DeVry University's bachelor's degree program in Game & Simulation Programming (GSP) positions students for success with an innovative experiential education.

Our graduates are well-rounded and ready to make an impact on today's ever-changing, demanding simulation and video game industries. The GSP curriculum includes training in a broad range of programming languages and software applications. These courses are integrated with a general education curriculum to reinforce essential critical thinking skills.

Learn more at devry.edu

DeVry University

ADVERTISEMENT

UNREAL TECHNOLOGY NEWS

BY Mark Rein
Epic Games, Inc.

LEVERAGE THE POWER OF DIRECTX 11 WITH UNREAL ENGINE 3

Epic recently presented what we would like to see in the next generation of games with our Samaritan real-time Unreal Engine 3 demonstration at the Game Developers Conference in San Francisco.

To take Unreal Engine 3 to the next level, we implemented DirectX 11 support along with NVIDIA's APEX physics technology, and these features were made widely available in the March 2011 release of the Unreal Development Kit (UDK).

Commercial UE3 licensees have source-level access to these additions. Here's a round-up of the DirectX 11 features accessible to anyone who downloads the latest UDK build from www.udk.com.

Bokeh depth of field (DOF)**:** Real world images captured with a camera lens often depict scenes with some parts more in focus than others. The out of focus objects form shapes, such as circles or pentagons, called Bokeh. Artists can control Bokeh shapes and textures in the UE3 post-processing chain.

Doing DOF as a post process works quite well for opaque objects, as each pixel has a depth associated with it. Translucent rendering, however, cannot work well with a post-processing method. Ignoring the problem can result in translucent objects that are either too much in focus or too blurry, depending on their background.

We solved the problem by giving control over which translucent objects are affected by DOF within UE3's material settings. Additionally, we've implemented a new material node that allows artists to adjust shading by fading objects out or blending them to a blurry state.

Tessellation: Because the DirectX 11 tessellation pipeline is programmable, it can be used to solve a large number of graphics problems.

Tessellation works especially well for natural objects with medium-scale details, and UE3's material input enables developers to adjust geometry tessellation on both the edges and the insides of triangles. Tessellation amounts can be controlled by distance from a camera or setup to place more triangles along silhouettes.

A popular refinement algorithm, PN-Triangles, softens the look of coarse models. PN-Triangle tessellation mode smoothes hard edges, converting low-resolution models to curved surfaces, which are redrawn as a mesh of finely tessellated triangles. UE3 also has support for crack-free tessellation and displacement.

Image-based reflections: Image-based reflections are a part of UE3's DirectX 11 rendering pipeline. This technique is used to render real-time whole scene HDR reflections as seen in Samaritan. The technique works by reflecting an image which is an approximate version of a scene at each pixel, which makes it more efficient than previous reflection techniques, such as planar reflections.

UE3 enables reflections on any surface with varying glossiness and blurriness. This is useful for visuals such as wet roads where puddles mirror reflections, while other parts of the road appear glossy.

UE3 also supports anisotropic glossiness, where reflections are streaked more in one direction, and dynamic components enable all parts of the reflection except for static shadowing to be changed at runtime. UE3 supports dynamic object shadowing as well.

Deferred shading: Deferred shading allows dynamic lights to be rendered much more efficiently. Traditional UE3 lighting is called forward shading because the dynamic lighting is calculated while rendering a scene's meshes.

With deferred shading, material properties such as diffuse color are stored in render targets, called G-Buffers, while rendering the meshes in the absence of lighting. Later, in a deferred pass, each light looks up the material properties from the G-Buffers for a given pixel and calculates lighting based on that.

Lights rendered with deferred shading are about 10 times faster than lights rendered with forward lighting. In Samaritan, the opening scene had 123 dynamic lights, and all lighting was done using deferred shading except on character skin and hair.

Full scene anti-aliasing: UE3 supports full scene anti-aliasing in DirectX 11 through multisample anti-aliasing. MSAA is a hardware feature that shades pixels only once but evaluates the depth test multiple times per pixel.

In UE3, deferred passes like lighting and shadowing work correctly with MSAA by detecting geometry edges and shading per-sample along those edges.

UE3 materials have a feature that multisamples the edges of masked materials. This is especially useful for creating realistic hair and foliage.

Screen-space subsurface scattering: Subsurface scattering refers to light that penetrates the surface of an object, scatters through its interior and exits at a different location. This is why subsurface scattering makes skin appear more luminous.

UE3's subsurface scattering is a screen-space effect that blurs the light incident on the object's surface to other nearby points on the surface. The blur attenuates the lighting based on the world-space distance between the incident and exitant points to model absorption of light by the interior of the object.

For help getting started with these features, visit http://udn.epicgames.com/Three/DirectX11Rendering.html.

Mark Rein
Epic Games, Inc.

Canadian-born Mark Rein is vice president and co-founder of Epic Games based in Cary, North Carolina.

Epic's Unreal Engine 3 has won Game Developer magazine's Best Engine Front Line Award five times along with entry into the Hall of Fame. UE3 has won three consecutive Develop Industry Excellence Awards.

Epic is the creator of the mega-hit "Unreal" series of games and the blockbuster "Gears of War" franchise.

Follow @MarkRein on Twitter.

UPCOMING EPIC ATTENDED EVENTS

E3 Expo	**Develop**	**Comic-Con**	**GDC Europe**
Los Angeles	Brighton, UK	San Diego, CA	Cologne, Germany
June 7-9, 2011	July 19-21, 2011	July 21-24, 2011	August 15-17, 2011

Please email: mrein@epicgames.com for appointments.

WWW.EPICGAMES.COM

BRANDON SHEFFIELD

REAL GAME DEV HEROES

FOR THOSE WHO GO ABOVE AND BEYOND THE CALL OF DUTY

////////// WE'VE ALL HAD GAME DEVELOPMENT PROJECTS WHERE SUDDENLY EVERYTHING SEEMED TO GO WRONG—NPCS ARE FALLING THROUGH THE FLOOR, BULLETS BOUNCE OFF INVISIBLE BARRIERS, THE SKYBOX COLORS HAVE INVERTED—AND NOBODY CAN FIGURE OUT WHY.

THEN, SUDDENLY, SOMEONE STEPS IN WITH THAT "A-HA!" MOMENT, THE PROBLEMS ARE SOLVED, AND THE GAME GOES ON TO MAKE ITS SHIP DATE. OR PERHAPS THE GAME WASN'T EMBATTLED BUT SIMPLY BLAND, AND THEN A CLEVER DESIGNER REALIZED THAT ADDING A TIME LIMIT UPPED THE TENSION TWO-FOLD. OR PERHAPS THERE'S SOMEONE WHO'S JUST DAMN GOOD AT HIS OR HER JOB! IN THE FOLLOWING PAGES, WE CELEBRATE HEROIC MOMENTS OF GAME DEVELOPMENT, WHEN A PROJECT WAS SAVED, MINDS WERE ENLIGHTENED, OR PROCESS WAS STREAMLINED.

SPLATTERED HOUSES

✓ As they labored on SPLATTERHOUSE, the development team had been working for months in a custom character-scripting tool that sat on top of Microsoft Visio as a plug-in. While the interface looked slick, the team was plagued with general production woes whenever they worked in the tool. Clicking on a graphical node would bring up a scrollable list of parameters that could be tuned, but the interface was extremely slow to the point of absolute frustration. The tool would take five seconds per mouse click to update. Scrolling the parameter window would force the tool to re-query the parameter list and force a full redraw. Once you entered a value and pressed the Enter key, you needed to wait three to five seconds for the whole program to finish updating. For a game that was meant to induce nightmares, none were as frightening as attempting to work within the character scripter.

Enter Justin Pease, an engineer who had been recently assigned to the gameplay programming team from the studio's central tech group. After getting up to speed in about a week, Justin worked a couple of extra nights, unbeknownst to the production team, focusing his efforts on improving the scripting tool. Why, you ask? Probably because he was going to have to use the tool to help improve the main character, Rick!

Regardless, one evening Justin sends out an email detailing that he has "improved" the character scripter. The results were staggering... suddenly the tool functioned as intended! No more waiting for updates and refreshes. Not only was the tool usable, but it actually allowed for production gains in terms of overall velocity. In trying to quantify the efficiency gains Justin made in the tool the numbers were ridiculous ... something like 3,000 percent or one man-year. On the tight production schedule Namco faced after taking the project internal, Justin Pease's tools improvements are truly the stuff of coding heroism.

—Michael Boccieri

WARHAMMER: AGE OF RECKONING.

HAMMER OF THE GODS

✓ I once had the pleasure of working with a man named David Scott. David worked at me on the Warhammer Reckoning project as an engineer (Software Engineer II to be specific). During my time he discovered a small flaw in the code that had existed for years. It was buried deep within, and had been passed over so many times that all of us just assumed it was an inherent fluke of the engine we could not overcome. By correcting it, he had solved a mystery that spanned years, eliminating the single cause of client-to-server latency in the game. Warhammer is a PvP (RvR)-oriented game, so this bug was one of the single most frustrating things about the player experience. *—Haley Chivers*

DOOD, WHERE'S MY DESIGN

✓ Dood's Big Adventure is a game we developed at THQ Digital Studios Phoenix (formerly Rainbow Studios) and was released in November 2010 as one of the games for use with the Wii uDraw tablet. But the published game that you see today on store shelves is drastically different from the type of product we originally began developing.

Game development is typically much more about execution than singular ideas. In other words, it's the process of developing and refining an initial idea over time that results in something meaningful. Normally, the original idea itself is just a kernel of a starting point, and often those ideas are transformed enough during the development process that the majority of weight and credit should and does go to the process, as opposed to the idea. However, during this particular project we enjoyed one of those rare moments where a single compact idea completely saved a game that was in crisis by all accounts—resource-wise, morale-wise, and direction-wise.

In January of 2009, we began work on what would eventually become Dood's Big Adventure for the uDraw tablet (which we were also simultaneously designing). Seven months later, about halfway through the development cycle, we were not quite out of pre-production, and the extent of our accomplishments thus far was contained within one side-scrolling "vertical slice"-style demo level. At this point, the plan was to build a story-based game with a linear progression structure, lots of narrative, and some collection of mechanics that utilized the tablet in hopefully essential, interesting, and usable ways.

The demo level we had built thus far looked great, but was completely unsatisfying on numerous levels. Because the various control mechanics we were developing were so different from each other, the only way the player could switch between them was to pass between specific choke points in the level. In one section of rooms the avatar was walking and jumping, in another section he was contained in a bubble and could float in any direction, in yet another section you would connect points on the screen with the pen to produce a physical object that would allow further progress. The level was completely linear, save for the standard "key and door" chicanery that games sometimes use to trick people into thinking they actually have a meaningful choice about where to go.

Besides these design problems, the entire level was wrapped around a "giant robot" narrative where the player was moving through different sections of the robot's innards to activate it and save a fictional planet from an evil alien invasion. This fundamental dependence on narrative was exceptionally risky on two big counts: it meant that we still had to create a large volume of non-interactive content in the first place, and it cornered us in terms of development flexibility. It meant certain parts had to come before others in order to keep the narrative coherent, and if something didn't work, well, we would have had to just re-do it or keep changing it around until it did. On top of this, all the different mechanics were sufficiently underdeveloped that none of them were really fun to play, and we didn't have an adequate contingency plan in the event that any or all these mechanics proved to be garbage.

As if that weren't enough, our art style and visual grammar for interactions was not yet well defined. The original principal designer, technical director, and senior producer each had independently left the company by that time. We were also working exclusively with tablet emulators and had no idea about how the final tablet product would ultimately affect playability. Oh yeah, and we also had six months to finish the game. Things were not looking good.

It was around this time that my lead designer, Devin Knudsen, came up with a simple and brilliant idea that ultimately proved to be the singular project-saving decision. We called it a "refocus," and it was a fundamental structural change to the game that allowed us to minimize all the big scary risks by modularizing as many components of the game as possible. In this way, if any part of the design proved not to work, we could just scrap it and the rest of the game would be unaffected. The details went like this:

1. We cut out most of the non-interactive narrative elements. This way, we could focus all our remaining development efforts on the actual interactive game, which was more important. This also guaranteed that no portion of the game would have to depend on a specific narrative element to justify its existence.

2. We changed the level-structure plan from nine long, linear and heavily interdependent gameplay sequences to (ultimately 60) short and totally independent "challenge" style levels. Each level would just use a single input-control mechanic from start to end, and the game would score the player in terms of how skillfully and quickly they could navigate each level. The time constraint was essential to making otherwise boring interactions interesting and replayable, and the new structure allowed for a single designer and artist to concentrate on one level at a time without affecting anyone else's work.

Eventually, we decided to stick with four particular gameplay mechanics that proved themselves to be worthwhile. Our new modular structure meant that we didn't have to make these systems work with each other, which was a much simpler problem to solve. We ended up with 60 total levels, across 4 different mechanics (15 levels for each mechanic). The wide variety of play modes, combined with the "short and numerous" level structure made the game much more accessible and lively than it would have been otherwise. On top of that, it allowed us to finish the game in a way that was satisfying to all of us with the little resources we had left.

—Ara Shirinian (special thanks to Scott Blinn and Trapper McFerron)

GAME JAMMED

✓ In May 2010 Wild Pockets hosted a 24-hour game jam. We entered as a team of three with hopes of taking home the $3,000 grand prize. Our team had two designer-scripters (Daniel Bryner, Brad Johnson), and one artist (Chris Webb). Chris had the pleasure of doing all the art for two games since both Dan and I wanted to work on separate games. One of the requirements of the winning game was to have an end screen listing the game jam sponsors.

Chris actually finished all the art for both games (including the sponsor screen textures) with half an hour to spare, so he put his head down on the table and started sleeping. This whole time Dan and I were scrambling to put the finishing touches on our games, such as making sure the sponsor screen is displayed at the end. I had made a UI script that would do various things, one of them being the display of the sponsor screen. Dan had already put a level complete screen in and was waiting for me to help him implement the UI script, but I kept telling him I didn't have enough time to finish my game and help him, so he was on his own. Long story short, we had about five minutes left, and Dan and I both start to panic because we still have to finish our games and then post a link to finalize our submission, which neither of us have done, and Dan still doesn't have the sponsor screen in, which means the game will be disqualified from competition. Things get heated and talking turns to shouting.

I submit my game with three minutes left, but what do we do for Dan? There's absolutely no way we can get his script working in time!

Chris pops his head up off the desk, renames the sponsor screen texture so it overwrites the level complete screen and hits upload with about one minute left to spare. Our games end up going on to tie for first place so we take home not only the grand prize of $3,000, but the $600 for second place as well. Chris Webb saved the day. It was definitely an Indiana Jones-saving-his-hat-from-under-the-crushing-wall moment.

—Bradley Johnson

SPORED

✓ *Game Developer*'s original "call for heroes" asked for people who saved the day during a game's development with a heroic effort, or who had an "a-ha!" moment that turned the tides. My personal game development hero did plenty of day-saving and a-ha-ing when I worked with him on SPORE, but those aren't the reasons he's my hero. Tom Bui (now at Valve Software) is my

QUALITY ASSURANCE ON DEMAND

TRY US FREE!*

*Get $1,000 credit toward your first project with us.

Game testing since 1999
Licensed for all game consoles

Functionality • Localization • Compatibility
Security • Mobile • Beta Test Management
Compliance • and more!

303.627.1110

www.iBeta.com • sales@iBeta.com

Restrictions apply. Call for details.

FAR CRY 2.

SPORE.

game development hero because he's the most productive programmer I've ever met, and I find that incredibly inspiring.

It must be said: I struggle with pretty bad productivity problems. I realize this isn't an issue everybody has to face, so this may not resonate with you personally. Some people can just sit down in the morning and start typing code, stop at lunchtime, and then do it again in the afternoon. I am not one of those people, unfortunately. But just being around Tom made me more productive and focused. Seeing the endless stream of his check-in notices go across the team mailing list, taking a break from a conversation with him to go to the bathroom, only to find he's fixed a bug during the brief intermission while I was gone instead of checking email or Facebook or Twitter, finding he's added a feature in an hour that people thought would take a week—the man is just a machine! Better yet, he's incredibly kind and self-deprecating, so instead of this amazing productivity being intimidating or humiliating, it's completely inspirational. You want to work harder and faster around Tom—not because you have a prayer of working as hard and fast as he does, but because you feel like you can draft him a little bit, like cycling behind Lance Armstrong or something. *—Chris Hecker*

TRIALS HD.

TRIALS, TRIBULATIONS

✓ Although it's not widely known, early in the development of TRIALS HD, the original design called for a "Crash Mode" that was to be just as important as the timed obstacle course driving we have all come to know and love ... or curse and throw our controllers because of, as the case may be.

Crash Mode was originally meant to be a BURNOUT-style game. The goal was simply to gain points by making as spectacular a crash as possible. Many on the team were excited by it, our partners were enthusiastic, and all around, we thought it was a great idea, as it was believed that its simpler mechanics would help bring the game to a much wider audience.

The problem was it just wasn't very much fun in large doses. With our core action designed for a 2D plane, there simply weren't enough ways to make the crashes varied enough on any particular level. There were only so many routes we could create through any given warehouse.

Realizing how this limited the game's potential, but not wanting to abandon the idea altogether, project manager Jorma Sainio, lead programmer Sebbi Aaltonen, 3D guru Sami Saarinen, and creative director Antti Ilvessuo, worked hard trying to find ways to somehow utilize the potential of Crash Mode in small doses.

After many iterations and discussions, the core team came to the conclusion that Crash Mode shouldn't be in the actual game. However, some parts of the Crash Mode prototypes were a lot of fun. So, they turned the question upside down: how could we take advantage of that small set of fun, short gameplay moments and make TRIALS HD a better game overall? When Crash Mode was abandoned, these small ideas didn't have to be limited to fit Crash Mode anymore. After this decision, the team came up with many small, fun ideas, that on their own would not have made a game mode, but could be turned into a small, fun collection of events comprising the Skill Games that many riders have come to love. *—Jason Bates*

A CRY TOO FAR

✓ To provide some context, at the same time that we were developing FAR CRY 2, the technology pipeline team was building the game's engine, which came to be called Dunia, in parallel.

The major technical challenge for the game was to convincingly deliver a realistic open world across 50 sq km of typically African terrain types, including desert and savanna. In other words, the engine was being asked to support large draw distances in situations where natural features like mountains couldn't be used credibly to block sight lines. The old-gen solution of simply cranking up fog effects wasn't compatible with the game's next-gen environmental graphics. And in a few regions, from readily accessible vantage points, we discovered that it was theoretically possible for players to see from one end of the world to the other.

The problem lay in the way Dunia streamed in chunks of the world. Each sector that was rendered inflicted a performance burden on the game. At the time, Dunia lacked a per-sector LOD system (although individual game assets had multiple LODs), so in order to protect the framerate, the engine imposed caps on the number of sectors that could be simultaneously loaded, visible and active in the game.

We found ourselves confronted with the prospect of needing to make fundamental changes to large—we're talking multiple kilometer—swaths of the game world, late in production. It was a potentially catastrophic setback that could have easily derailed a title that had already been delayed by several months.

After listening to the level design and art technical teams debate the relative costs of various fixes with the producers, FC2's technical architect, Philippe Gagnon, spoke up and asked for a few hours to investigate a possible solution. He came back with something that was astonishingly elegant: he curved the world.

Or at least he created the illusion of a curved world. Specifically, the game's renderer was modified to apply a curve function at runtime, so that the sectors that were the furthest away from the player's POV fell below the game's virtual horizon, causing them to be culled automatically. Once the solution was proven to do the trick, Gagnon put in additional code that raised and lowered any other visible non-terrain entities in order to match the pseudo-curvature.

For players viewing their surroundings from inside the game (and in the absence of any powered aircraft, which FC2 didn't feature), the effect was too subtle to be noticed. Our level designers still needed to restrict access to some of the taller mountain vistas and obviously hang gliders needed to be placed where they weren't going to suddenly give the player a disconcertingly sub-orbital view of the world.

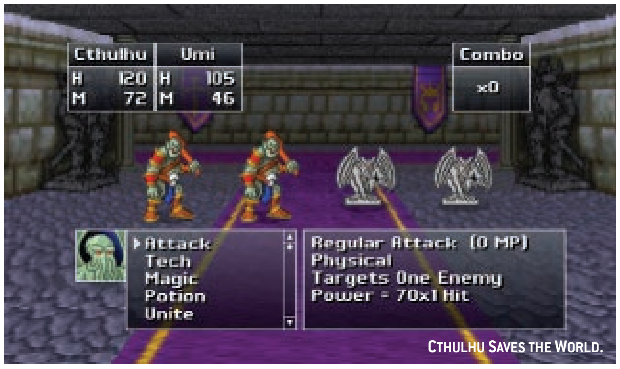
Philippe's cognitive leap saved our collective asses. It was one of those moments that becomes immortalized in three and a half years' worth of production lore. *—Patrick Redding*

xbox live indie games postmortems

CTHULHU SAVES THE WORLD

On April 22nd, 2010, Zeboyd Games released its first title, BREATH OF DEATH VII: THE BEGINNING. BoD was patterned after the popular 8-bit RPGs of old but also featured modern elements, like branching level-ups, a combo system, and random encounter limits. It was a big success for us. The game skyrocketed up the top-rated list on the Xbox Live Indie Games U.S. Dashboard (it ended up in the #2 spot for several months) and sold over 45,000 copies in its first year. Admittedly, the game only sells for $1, so it was not making us rich, but it was a very promising start for our new team. It also raised the question: what next?

We had a few promising ideas for new games already, but all of them felt too ambitious for a second project. We decided to do something a bit simpler—a spiritual sequel to our first game. It wouldn't be a direct sequel, since we wanted to gain a reputation for good games and not be labeled "the BREATH OF DEATH guys." Rather, it would build on the foundation we had created with our first game. This idea eventually turned into CTHULHU SAVES THE WORLD.



CTHULHU SAVES THE WORLD.

WHAT WENT RIGHT?

1 /// Concept. I had been playing around with the idea of taking a classic piece of literature from the public domain and transforming it into a comedy RPG. I also like the horror genre. One day, the two ideas mixed, and the concept of Cthulhu losing his powers and having to regain them through an RPG quest was born.

When you have a strong concept, it makes development so much easier. We had a wealth of material from Lovecraft's fiction to work with, which gave us plenty of great monsters and locations right from the start. Writing amusing dialogue was relatively easy since the concept was so strong. And since the internet is filled with Lovecraft and Cthulhu fans, getting people excited about the game was easier than it might have been otherwise.

2 /// Building on the past. If we had decided to make CTHULHU SAVES THE WORLD as our first game, I don't think we would have ever finished it. As our second game, however, we had an existing RPG engine and some experience with coding, art, design, and writing. Since we could focus on making the game better, and not just on making it work, an

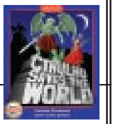
The Xbox Live Indie Games service was launched as part of the XNA Creators' Club (as "Community Games") back in May of 2008. It was viewed as a way for developers to release games directly on the Xbox 360 without having to get a publishing deal. Since then, the service has grown to include some 1,600 titles, with a number of standouts. We've compiled postmortems of five of the more interesting games on the service to give curious developers some idea of the XBLIG development environment.

otherwise insurmountable task became much more manageable.

3 /// Presentation. Our artist and musician really nailed the nostalgic feel of a classic 16-bit game while instilling their own personality into the presentation. Due to the small size of our team, we had to cut some corners, so we decided the game would have very little animation in it. Still, we did a lot with what resources we had. Since we didn't have the budget for big FMV sequences, we patterned our cutscenes after the non-animated comic book-style scenes used in PHANTASY STAR IV. This allowed us to have dozens of scenes in the game that were far more visually interesting than they would have been if done in the game engine. To make combat more interesting, we gave each enemy a unique "insane mode" sprite. And to draw attention away from the simplicity of our animation frames, the animation we did use was very quick.

GAME DATA

CTHULHU SAVES THE WORLD



PUBLISHER	Zeboyd Games
DEVELOPER	Zeboyd Games
NUMBER OF DEVELOPERS	Robert Boyd (Design, Code, Story), William Stiernberg (Graphics, Level Design), and Gordon McNeil (Music)
LENGTH OF DEVELOPMENT	8 months
RELEASE DATE	December 29, 2010 (XBLIG), Spring 2011 (PC)
PLATFORM	XBLIG, PC

4 /// Accessibility. I love really complex games, but I'm not everyone. As a developer, I want my games to appeal to a wide variety of people. "Easy to learn, hard to master" is our motto. We had done some work with accessibility for our first game, but we went even further with CTHULHU SAVES THE WORLD, adding a save-anywhere feature, easy town-teleportation, 1-ups to retry battles, a new Easy difficulty, and an unlockable Overkill mode (in which the player levels up very quickly at the start). At the same time, we didn't want to neglect the gamer in search of a challenge, so we added a new game mode (Highlander) where you can only bring one character into combat, in addition to the Hard difficulty and Score Attack mode from our first game.

5 /// Marketing. In addition to our regular marketing efforts, we also helped to organize a special multi-game promotion called Indie Games Winter Uprising. I believe this promotion was a big help to the game's visibility, and is probably responsible for the increase in reviews from bigger and more prestigious sites (Eurogamer, Joystiq, and GamesRadar among others) compared to our first game.

WHAT WENT WRONG?

1 /// Underestimating the amount of work. We'll just take BREATH OF DEATH VII and make everything better—how much time could that take? A whole lot, as we soon found out. Our initial goal of three to four months of development time kept getting pushed back, and our estimated release dates became a joke. In the end, we just narrowly managed to get the game released before the end of the year, after eight months of work. I even ended up spending a good chunk of Christmas day working on the game. How's that for self-inflicted crunch?

When you add an extra layer of sophistication to one aspect of a game, your workload increases a bit. When you add an extra layer of sophistication to several aspects of a game, your workload increases exponentially. More work in one area

CTHULHU SAVES THE WORLD.

would result in the need for more work in another. It was a mess.

If we had planned the game out better beforehand and limited our improvements to a few key areas, development would have gone much smoother.

2 /// Lack of tools. With BREATH OF DEATH VII, we did all the maps directly in code. My artist would make them using a free map editing program, he'd convert them into arrays, and then I'd plug them directly into the game. This wasn't ideal, but it worked.

With CTHULHU SAVES THE WORLD, we upped the size and complexity of our maps, and our old process of making maps quickly proved inadequate for the task. Rather than stopping to make more efficient tools, we just plowed on. Creating maps was a time-consuming process, and modifying them was a pain.

If we had just taken a month or two at the start of the project to make some efficient map-editing tools, we could have made higher quality, better-tuned maps in drastically less time than it ended up taking.

3 /// Lack of funds. This was a direct result of our failure to plan appropriately. We were expecting a new source of income once CTHULHU SAVES THE WORLD came out after four months of development, but since the game's release date kept getting pushed back, the new revenue stream was delayed as well. I had to borrow money and find work elsewhere during development just to keep things going.

Thanks to the game's eventual release and a kickstarter fundraiser that we did soon after, Zeboyd Games has enough funds for the time being. And hopefully, our PC port of the game will help bring in enough additional funds so that we won't need to worry about running out of money while we work on our next game.

4 /// Not different enough. Although we felt that CTHULHU SAVES THE WORLD was a huge improvement over our first game, we received complaints that it wasn't different enough. Those complaints weren't completely without merit. Most of the changes that we made to gameplay were subtle, so that though everything was bigger and better, the core gameplay and interface weren't that far off from our first game.

For our next game, we will be making much more drastic changes to overall style and gameplay, as well as an overhaul of the interface.

5 /// Not enough iteration. We do plenty of theoretical iteration at Zeboyd Games. We spent weeks coming up with different insanity systems before finally choosing the one that we used in the actual game. However, we didn't do enough iteration with actual playable prototypes. CTHULHU SAVES THE WORLD wasn't playable in any way that resembled an actual game until about a month or two before completion. Had we worked faster on getting out a playable prototype in order to get feedback earlier, I think the overall quality of the game would have been much improved.

ELDER GODS

CTHULHU SAVES THE WORLD was an excellent learning experience for us. Although it was much harder work than we had anticipated, we are pleased with the results. Despite the higher price tag ($3), the game sold about 9,000 copies in its first 30 days (though daily sales have dropped off noticeably since then). Reviews have been overwhelmingly positive, and perhaps most important of all, the game has unlocked future doors of opportunity for us.

ROBERT BOYD *was an English teacher before he decided to go into game development, starting up Zeboyd Games with BREATH OF DEATH VII. He was the creator of the Winter Indie Uprising in late 2010.*

EPIC DUNGEON

Originally, EPIC DUNGEON started off as a self-imposed challenge to see if I could create a classic roguelike in two weeks. Those two weeks quickly turned into months (so I guess I failed the challenge). But the result was a successful twist on the roguelike that combined some of the genre's core mechanics, such as perma-death and random level generation, with more modern action-RPG elements—real-time gameplay and the development of stats and skills—as well as a dash of character. All of this was set within a text-based encounter system, wrapped up in a retro 8-bit package.

As a one-man development shop, having to switch between coding, writing, artwork, music and sound effects, then on to marketing was difficult, but ultimately rewarding.

I am happy with the result, and it has performed well for an Xbox indie title, selling over 20,000 units in its first three months.

WHAT WENT RIGHT?

1 /// Evolution. By starting out as a turn-based roguelike, EPIC DUNGEON retained a lot of the feel of its classic inspiration. If it had been planned as an action-RPG from the beginning, it would have been a very different, and, I suspect, less successful game.

For example, even though character movement is smooth, it is still based on an underlying grid system. Not only did this give it some retro flavor, but it also made it easy for players to quickly see and understand all the action quickly.

2 /// Playtesting. Playtesting really helped with the final level of polish. Beyond serving as a source of inspiration and motivation, it helped with the most important aspect of EPIC DUNGEON, which was gameplay balance. Most notable was the amount of discussion and work that went into balancing the economy. Acquiring better equipment is an important part of the game, but finding the perfect point where players can buy a few things, but not everything, was a real challenge given the random nature of the game.

3 /// Accessibility. Nobody reads instructions, and tutorials are boring. So my basic approach was to provide information on screen as it was needed. Tips occasionally fade in to highlight specific gameplay features as the player moves deeper into the dungeon. Thought bubbles are used to communicate when a button can be pressed to interact with an object. Icons for the face buttons (A/B/X/Y) are displayed on the HUD, and behave contextually (e.g., if a skill is on cooldown, the button will look like a timer).

Beyond that, I also tried to reduce useless, repetitive tasks. Giving players the ability to "Sell Junk" with a single click has been very popular. It basically just looks at all items and equipment in the inventory and does stat comparisons automatically, reducing the need to go through every item in the inventory manually.

As a result of all this, players get a pick-up-and-play experience that still has depth.

4 /// Level of difficulty. It seems like hard games are making a comeback, particularly among indie titles. The fact that EPIC

DUNGEON embraced perma-death, rather than providing some kind of "respawn" mechanism was met with much wider approval than I anticipated. It was something that I originally did out of principle, and I think that people have responded to that adrenaline rush you get when you know you've invested hours into a game only to see that you're being swarmed by enemies, your lantern is running out, and you are low on health potions.

5 /// Marketing. EPIC DUNGEON benefitted greatly from the Indie Games Winter Uprising promotion, a joint marketing effort by Xbox indie developers. Only a small percentage of the Xbox user base visits the indie game section with any regularity, so getting the word out played a significant role in EPIC DUNGEON's sales.

EPIC DUNGEON was the first title released during Indie Games Winter Uprising and as a result got early reviews and additional press, which, unfortunately, some of the other titles did not receive.

Not only did it generate early sales, but it created a critical mass of players referring EPIC DUNGEON to their friends, giving the game a much longer lifespan than I had originally anticipated.

GAME DATA

EPIC DUNGEON

PUBLISHER	Eyehook
DEVELOPER	Evehook
NUMBER OF DEVELOPERS	1
LENGTH OF DEVELOPMENT	9 months
RELEASE DATE	November 30, 2010
BUDGET	$0
PLATFORM	Xbox (Indie Games)
LINES OF CODE	30k

WHAT WENT WRONG?

1 /// Lack of planning. EPIC DUNGEON was developed in an iterative fashion, starting with a quick prototype, and then quickly moving into increasingly full-fledged implementations. Very little time was spent with a pencil and paper, and ultimately that caused problems.

First, early assumptions resulted in unnecessary limitations later on in development. For example, there are fundamentally two types of tiles—walls and floors—and later on, when I wanted to implement water and cliffs, the cost in time of doing so was just too great. Unfortunately, the existing tile concepts were just too integrated in my AI and lighting to easily be modified.

Secondly, I made a huge mistake developing the shop and inventory screens iteratively in code. It would have been much more effective to have spent some time sketching out the screens with pencil and paper first. Originally, there were separate screens for a player's inventory and a player's stats (DMG, DEX, DEF, LCK). Unfortunately, after getting everything pixel-perfect, I realized that I had to combine both screens, since (obviously!) it's important to see how equipment affects stats. Similarly, it became clear that it would be necessary to toggle between the shop screen and a player's inventory. So, I basically had to throw away the original shop screen and create a new shop screen that more closely mimicked the inventory screen to make the behaviors consistent between the two.

2 /// Music. I have mixed emotions about the music in EPIC DUNGEON. I'm proud of what I created, but I believe it could have been much stronger. All the music loops are roughly a minute long and change every level. Unfortunately, I used the same instruments for many of the loops, and they're all in the same key, so they can sound repetitive after playing for a few hours. I will definitely focus on having longer, better, and more varied music in my future releases.

3 /// Not having a deadline. During its development, EPIC DUNGEON didn't have a real schedule or plan for deliverables. This gave me the time to explore a lot of different ideas, but it ultimately just ended up putting the game into a state of unending development. If I hadn't participated in Indie Games Winter Uprising, which finally gave me a solid deadline, there's a good chance that EPIC DUNGEON would still be in development.

4 /// Building everything from scratch. A lot of development time was wasted reinventing the wheel—most notably, the amount of time I spent creating custom font, text layout, dialog box, storage, and screen management classes.

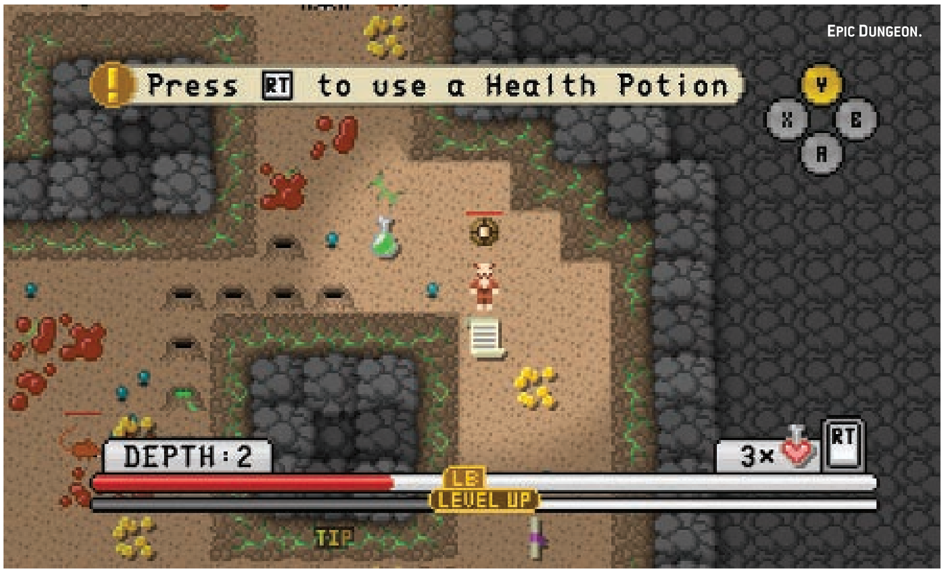
The upside of having written my own entire engine is that it's quite easy for me to debug or optimize should the need arise; however, research and a better understanding of existing technologies would have significantly sped up the process and probably improved my code.

5 /// "Epic" Dungeon. Using "Epic" in the title generated a bit of initial backlash, which really surprised me. Using a superlative in the title seems to suggest bad quality to gamers. I thought I was giving it a slightly ironic, but generally descriptive title: it's a dungeon, it's going to be big, it's going to be hard. Regardless, it literally is a player's first impression, so I wish I had spent a little bit more time thinking about the title.

ROGUE WAVE

I'm proud of what I accomplished with EPIC DUNGEON. It was a lot more work than I had planned on, but I truly enjoyed creating it and still enjoy playing it. So, for me, it's been a success. I'm looking forward to leveraging the knowledge I gained and libraries I've written to make something even bigger and better in the future.

MIKE MUIR *is the sole member of Eyehook Games, a small independent game development studio located in New York City. You can follow Eyehook Games on Twitter @eyehookgames.*



EPIC DUNGEON.

SHOOT 1UP

There's something extremely satisfying about mowing down tons of alien spacecraft with your streamlined fighter ship. I've always loved shoot'em ups, putting tons of hours into LIFE FORCE, R-TYPE, LIGHTENING FORCE (aka THUNDER FORCE IV) and SUPER ALESTE, and dreamed that one day I'd make my own insane shoot 'em up with a unique twist.

SHOOT 1UP is a forced-scrolling shoot'em up which bends the convention of 1UPs and allows players to control all their ships simultaneously. Any time they gain a new life, they get it as an active ship, which contributes to their firepower. Players can risk spreading out their ships, and endangering them, for the ability to fire a super laser. There's also giant spiders being milked. And dead whales. And a flying, naked lady-bot.

WHAT WENT RIGHT?

1 /// Unknown, gut-reaction design. My designs are usually pretty freewheeling, but for SHOOT 1UP it was even more open than normal. Fortunately, for a project in which I was trying to out-crazy Japan, or at least get close, this approach worked. I would simply brainstorm while running, in the shower, driving, or wherever else, and then write down any ideas and accept or reject them based on whether they were interesting, and if they fit the general feel of the game. I was careful not to think them through too much, which is what I think stifles some of my designs.

2 /// Supporting disabled gamers. I worked with accessibility organizations Able Gamers and One Switch to hone the menus and controls for the game to support as many options as possible for handicapped and non-handicapped gamers alike. It was relatively easy to support button remapping, and it greatly increases the game's flexibility and audience. We also gave gamers the option to control the overall gameplay speed, which made it much easier for frightened newbs or old-timers chastened by brutal shooters of the past.

3 /// Critical prototype. I came up with the core idea of "all your ships at once" in response to a challenge for the "Experimental Gameplay Project" web site. I quickly made a prototype of the gameplay based on an existing tutorial in XNA. Using existing technology to prove the fun of the idea was a massive benefit, and helped get the gameplay up and running quickly without getting bogged down in low-level coding issues.

4 /// The price was right. SHOOT 1UP was priced at 1 dollar, which helped all the reviews and made it easier for everyone to consider checking it out. It's likely the game enjoyed more buzz and a higher "critical mass," leading to more exposure than we could have gotten at $3–5 price points.

5 /// Great playtesting sessions. Although we had an original idea, we weren't shy about letting people play early versions of the game and integrating useful feedback. Lots of aspects, such as an improved story introduction (which now includes animated pictures), more enemy variation, and better play controls came out of playtesting, and heavily improved the final product.

WHAT WENT WRONG?

1 /// Didn't know what we had. SHOOT 1UP came together in a weird period in which I was "supposed" to be working on GRAPPLE BUGGY, but was also devoting time to a paying contract. By the time the game shipped, I realized it was pretty cool, and people might like it. Turned out it got some really good reviews and a lot of attention, which left me wishing I had done a better job on the art and animation, and had just generally added more content to the game.

2 /// No design and no timeline. While having limited initial designs in place for development ultimately worked creatively, it didn't work well for the timeline. We were trying to create the game quickly in an effort to get back to our "real" project. With an open-ended design, it's very hard to call it done. That's definitely a benefit of concrete game design documents and milestones.

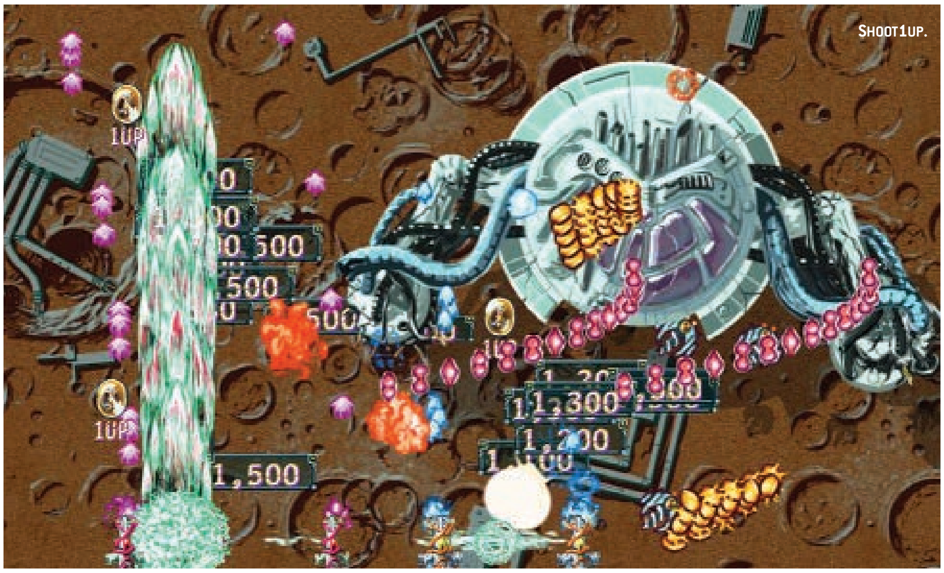
3 /// The price was wrong. SHOOT 1UP is only a dollar, which undervalues the game and others like it. I have no crystal ball to see if we could have made the same money, but I often wonder if we should have priced it at the next point ($3) instead. But our success here does set us up to release a new, more expensive shooter of higher quality.

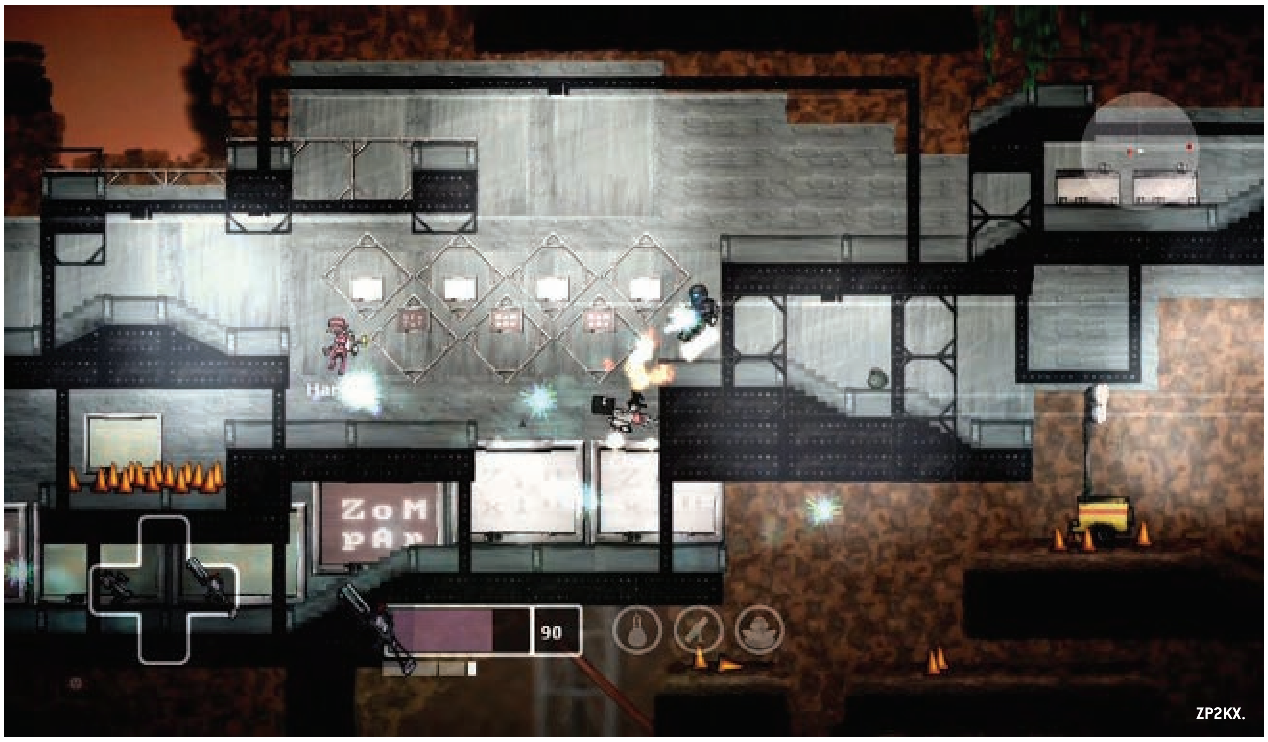
4 /// Poor underlying technology. While using a starter-kit-style codebase to get things going is very good for jumpstarting a project, at some point, it's possible to diverge so far that you

GAME DATA

EPIC DUNGEON

PUBLISHER	Mommy's Best Games, Inc
DEVELOPER	Mommy's Best Games, Inc
NUMBER OF DEVELOPERS	Hamdija Ajanovic (music), Amy Fouts (business and producer), and Nathan Fouts (all the game stuff)
LENGTH OF DEVELOPMENT	4 months off and on
RELEASE DATE	February 25, 2010
SOFTWARE	XNA, Photoshop 7, tiny Wacom tablet, SD and HD televisions, tons of SNES/TG-16/Genesis shooters, lots of robotics and bug photos, and some porn (for reference of course)
PLATFORM	Xbox LIVE Indie Games


SHOOT1UP.


ZP2KX.

need to rewrite underlying code to improve performance. I try not to over-engineer my projects in an effort to keep development time down. This approach mostly works, but here it started to hinder performance as the original starter kit was never meant to support so many moving objects and interacting collisions. Taking a few weeks to better optimize underlying systems, such as collision (for all the interactions between bullets and enemies), could have replaced cuts I had to make to bullet and enemy counts.

5 /// Forgot my homework. This was my first sprite animation game, as our first game, WEAPON OF CHOICE, was skeletally animated. It was very exciting to get to start playing around with traditional cell animation, and in my excitement, I forgot the basics, such as onion skinning to better line up progression in an animation. Had I taken a week or two to brush up on traditional animation methods, the final results would have been much better. Fortunately, I made good on the idea of studying traditional animation when creating our next game, EXPLOSIONADE, and the quality obviously improved.

SHOT' EM UP

SHOOT 1UP was my first shoot 'em up, created after playing and loving the genre for years. At over 25k sales, we've had lots of gamers, even lots of non-shooting game players, write to say they love the game, which encourages me going forward. The design was loose, and my primary hope is that I can remain open to more interesting ideas I may have for new designs, rather than analyzing them into vanilla.

NATHAN FOUTS is president of MBG and 8 Bit Horse. Read more about his love of 2D games on Twitter @MommysBestGames.

ZP2KX

Ever since broadband gave us deathmatches without LANs, I've dreamed of making my own online multiplayer shooter. I tried this in 2005 with ZP2K5, an ugly game with terrible lag that basically no one played. When XNA 2.0 nailed easily implementable online multiplayer over Xbox LIVE, I made ZP2K9, which did pretty well, selling 25,000 copies. A little over a year later, as part of my XBLIG-for-fun development methodology and just in time for the Indie Winter Games Uprising promotion I released ZP2KX, which sported an overhauled graphics engine, new maps, character customization, new weapons, classes, levels, and more.

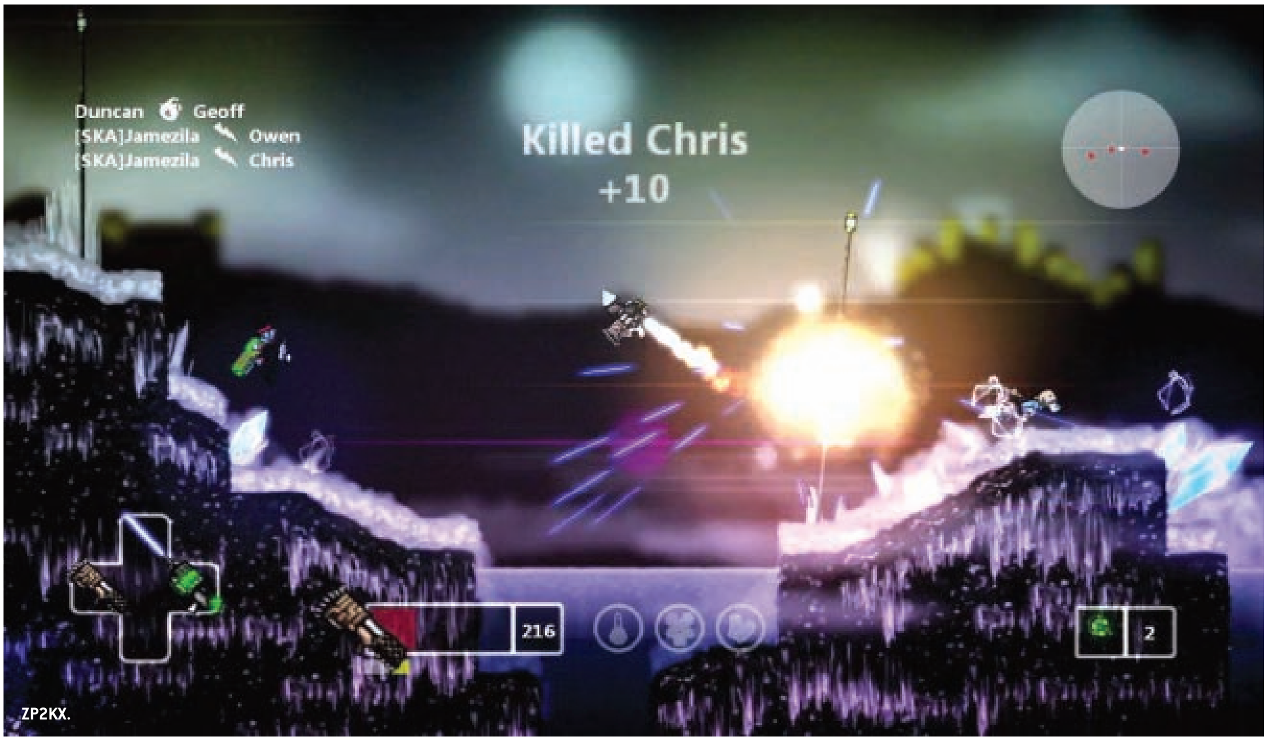
WHAT WENT RIGHT

1 /// Recycled tools. ZP2K9 maps used a pretty old-school tile system; tiles could be painted as either empty, sloping up, sloping down, or full, and the game would then figure out which of about 16 tiles per palette (grass, wood, metal, and so forth) to draw per tile so that each would mesh with its neighbors.

For ZP2KX, I rigged in the map editor from the XBLA game I've been working on for the past two years, THE DISHWASHER: VAMPIRE SMILE. The new format kept the tile-based collision map but used a multiple layer drag-and-drop system and definable sprite sheets, which allow the designer to create far more organic, expressive maps with stuff like sprite-based animated lighting, swaying grass, and flowing fog all on a bunch of parallax layers.

2 /// Distributed development. ZP2KX is, to date, the most highly collaboration-oriented game I've released. About a year ago, I brought on Dustin Burg as marketing coordinator, basically to organize for PAX, handle community, work with press, and things like that. After a bit of THE DISHWASHER: VAMPIRE SMILE development, I got him set up to use the map editor so he could create some arcade levels. Since ZP2KX now used the same map editor and format as THE DISHWASHER: VS, I was able to draw up some map sprites and then let Dustin churn out some maps. ZP2K9 had just 4 maps while ZP2KX has 11, which is a far more respectable amount of levels for a multiplayer-only game.

3 /// Hobby mentality. Ever since my earlier game THE DISHWASHER: DEAD SAMURAI got the go-ahead for XBLA, I've stuck with a system



ZP2KX.

for dividing my time and energy over multiple outlets, all made possible by Xbox, and it basically goes like this: XBLA is my day job, XBLIG is my hobby. ZP2KX is the latest product of this system, but it's also yielded XBLIG games ZSX4: GUITARPOCALYPSE, ZP2K9, and I MAED A GAM3 W1TH Z0MB1ES 1N IT!!!1, as well as XBLA games THE DISHWASHER: DEAD SAMURAI and the soon-to-be-launched THE DISHWASHER: VAMPIRE SMILE.

The aim of this system is to keep momentum high and burnout low. XBLA development during the end of a cycle tends to alternate between "hurry up and wait" and "stare numbly at the bug tracking database that is taunting you," so being able to pour some energy into another creative outlet helps ease frustration that otherwise would go in the Nervous Breakdown Bank.

4 /// XBLIG promotion FTW. ZP2KX was part of Indie Winter Games Uprising, a somewhat experimental and extremely good idea from Robert Boyd. The idea behind IGWU was that developers who had a track record of quality and valuing creativity over profitability (i.e., no Avatar games) released Xbox Indie games during the same window, so that we could all benefit from the promotion. Microsoft jumped on the promo, giving us an awesome chunk of dashboard space.

5 /// XNA Multiplayer is mostly awesome. Before XNA, multiplayer seemed like a totally unattainable dream. I'd dabbled in Winsock and DirectX 8 (oof!), but even without the awful execution issues I ran into, higher-level networking tasks, like searching for active games and just maintaining a server without futzing with router settings (forget about invites and friends lists!), were a total pipe dream for an inexperienced hobbyist developer. XNA streamlined all of that to the extent that, given my previous failures, it was actually *exciting*.

GAME DATA

ZP2KX	
PUBLISHER	Ska Studios
DEVELOPER	Ska Studios
NUMBER OF DEVELOPERS	2.5
LENGTH OF DEVELOPMENT	9 months
RELEASE DATE	December 23, 2010
SOFTWARE	XNA Framework, XACT, FLStudio, Paint Shop Pro, Photoshop, Zka Editor
PLATFORM	XBLIG
LINES OF CODE	80k

WHAT WENT WRONG

1 /// Generosity isn't always the best policy. Because the driving philosophy behind ZP2KX was more of a fun, hobby mentality, whenever there was a choice between generosity and stinginess, I erred on the side of generosity. However, generous game design is not always (or even typically) good game design.

Example: You kill enemies to earn XP, which goes toward increasing your level, which unlocks new clothes, skills (perks), jetpacks, and the like. XP earned is multiplied by killstreaks. When faced with the choice of letting gamers earn XP by killing bots (which, compared to human players, aren't exactly brimming with strategy), I reasoned that while killing real humans is the most satisfying way to play (odd how that comes out), I shouldn't punish gamers who want to practice on bots a bit.

What ended up happening was that someone was able to max out his level within nine hours of the game's launch (I was only able to get to 85 or so of 100 after about two weeks of pretty heavy play). By playing a certain gametype with bots only, players are able to rack up obscene killstreaks by spamming nuke grenades into the center of the map for hours on end. While this didn't bother me too much at first, in time I realized that, no matter how small an online universe seems, every little exploit that you leave in there goes a huge way toward destroying the perceived integrity of the universe for everyone.

2 /// Time to rethink player interaction. ZP2KX was new ground for me in terms of giving humans the ability to interact with other humans, and I erroneously went about it with the philosophy of "everyone will be cool." It turns out that giving the host player the opportunity to change game type and mutator or quit whenever he or she wants can be pretty obnoxious for other players. Giving the host full control is fine for private matches where everyone is friends, but when your competitive edge is at the mercy of what could be a fickle dabbler who is also a sore loser, your experience certainly suffers.

3 /// XBLIG jail is no fun. I put three titles on XBLIG before Microsoft began implementing the "jail" policy, where games that fail peer review have to sit out for a week. The policy is well meaning because on a pre-jail peer review list, developers had a tendency to use peer review as a means of testing, spamming resubmissions until it got through. With jail, developers get a week to iron out bugs, during which time they could put their game in playtest.

The three titles I got on XBLIG pre-jail passed peer review with no trouble. ZP2KX had to be pulled twice, and each time it was over bugs that were found several days into review, so with jail time added, the game spent nearly a month in review. This wouldn't have been nearly as frustrating if I weren't trying to get the game up for a meticulously planned promotion. Fortunately, the game passed in 2010. Barely.

4 /// XBLIG competition is no fun. ZP2KX is an incredibly deep game, from a gigantic, dynamic weapon roster to competent character customization, gamers can formulate any number of unique, incredibly effective strategies to dominate online. Yet we are consistently outsold by AVATAR PAINTBALL WARS, a game where in the trial, every unused button brings up the upsell screen.

5 /// XNA multiplayer is mostly awesome... but not without its problems. I used XNA 3.1 for ZP2KX, and XNA 3.1 has a couple of multiplayer engine flaws that exist at a level where the best I can do is catch an exception thrown at `NetworkGame.Update()` when the game enters a "things have gone all wrong" state. What this means for the gamer is that every time someone leaves a game, there's a 0.3% chance that the game will end. Sounds rare, but it seems to happen just when things are getting good. These issues have been fixed in XNA 4.0, but I've been a bit sidetracked...

HOW'D WE DO?

Overall, ZP2KX was a great experience. Commercially, it did fairly well (at least on XBLIG terms), selling over 19,000 copies in the two and a half months it's been out. As a project, it was a ton of fun to work on. After hours of mind-numbing bug fixes on THE DISHWASHER: VAMPIRE SMILE, it was cathartic to just shut my brain off and draw ZP2KX costumes. And Dustin, who hails from journalism land, can now say he has thousands of people playing ZP2KX levels that he designed and built. At the end of the day, we made a lot of people happy. I think that's a pretty good deal.

JAMES SILVA has been making games for years, but is best known for his Dishwasher series on XBLA, and the XBLIG I MAƐD A GAM3 W1TH ZOMB1ƐS 1NIT!!!1.

SOULCASTER II

SOULCASTER II is the follow-up to my March 2010 debut title as an indie developer. Before that, my industry experience was mostly as a contract musician/sound designer for the Nintendo handheld systems, a job I've had since about 2002. I began work on the first SOULCASTER in October 2009 when a project I was slated to work on was canceled, and I suddenly had some extra time. I had learned C# programming to develop my own audio processing tools, so I figured it was time to make a game of my own, to be released on Xbox Live Indie Games. Both SOULCASTER games were built from scratch in XNA with no external libraries, and I did all design, code, art, and music.

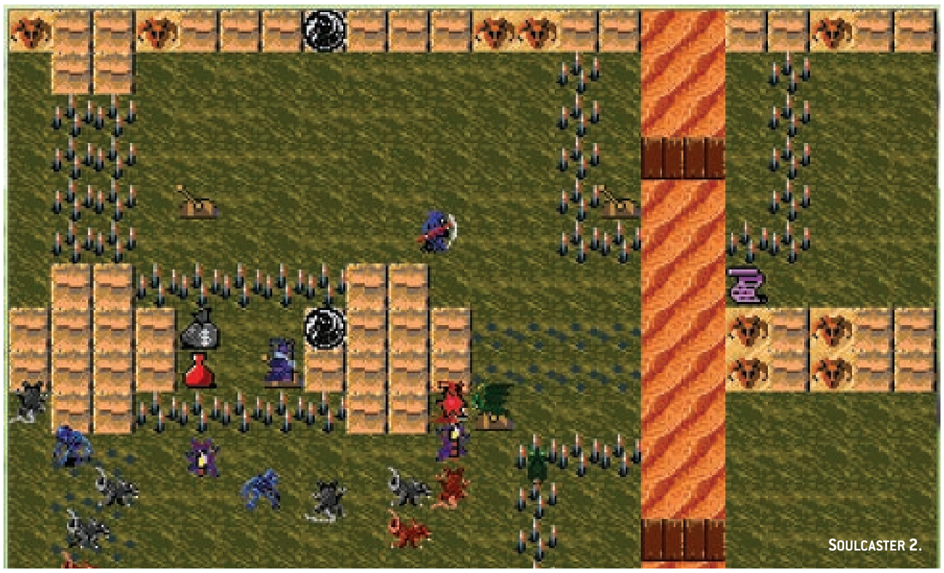
SOULCASTER was successful beyond what I had imagined, selling a few thousand copies in the first couple months, and holding a nice position in the "top 20 highest rated" list. So even though my contract work started picking up, I set out to make a sequel. This postmortem reflects how making a sequel may not be as easy as making the first game, even if you use most of the same technology.

WHAT WENT RIGHT

1 /// Scope. My first design move was to list the potential new features, then slash that list down to the few most important things. While I had grand ideas for features like a town, character inventory, and a persistent world, I knew these would take a lot of time to implement and test. I stuck with the core concepts of dungeon exploration and summoning, and just aimed to have a more high-quality presentation of the same system. The final task list included an improved environment graphics engine, better enemy AI, and a few new enemy types and scripting features. I didn't risk any time tampering with the core mechanics that made the original game fun.

2 /// Tools. Twenty percent of development time for the first SOULCASTER was devoted to making the level editor. In working on the sequel, I spent another month polishing it and making it as easy and fun as possible to design levels with. This time investment paid off big time: I was able to build about 60 levels, and then keep the best 30. I remade the final stage itself five times from scratch. With an editor that was anything less than total slickness, this would have been arduous.

3 /// Playtesting. I was afraid the sequel was going to be a bit too easy when I first had friends test it. Boy, was I mistaken. It's easy to forget how good you get at your own game when you are playing it


SOULCASTER 2.

constantly and know all its secrets. Watching friends play levels for the first time was eye-opening.

For example, the game opens with the hero being ambushed in a swamp by the undead. In the original version of the game, the skeletons waited menacingly on the other side of a river, unable to reach the player, but out of range themselves. Players said things like "How do I shoot them?" while they spent their precious demo time wandering around. So, I put the archer spirit in the first stage, and it made the goal a lot more transparent. Fixes like this, especially in the first few stages, helped get the player right into the action.

4 /// **Music**. My background is in music composition, and the soundtrack to any game is always the most important thing for me. I think I really delivered exactly what the game needed, and many critics have specifically mentioned the soundtrack as a highlight of SOULCASTER II. I also had the songs professionally mastered, which evened out the levels from song to song and added a lot of clarity. You don't want the player reaching for the volume control mid-game.

5 /// **indie games winter uprising.** When it happened, IGWU was the largest-scale grassroots marketing coalition in the history of XBLIG. It paid off in the vast amount of press coverage we received. The extra traffic to my web site almost made up for my own lack of marketing specific to the game, which I'll get into in the "wrong" section.

WHAT WENT WRONG

1 /// **Scheduling**. I started working on SOULCASTER II about one month after the first game was released. I didn't create a solid release schedule because, hey, how long could this really take? It's just a sequel. I'll just spruce up the graphics engine, beef up the editor, and then crank out content. Let's see, one level a day for 30 days, plus some time for graphics and music... This should only take three months, tops. Whoops! If I had broken the project into stages and set time limits, I would have saved myself some time and focused on what was important. But not as much as I would have if I had done more testing on the platform—which leads us to...

2 /// **Testing on target.** Programmers reading this, brace yourself for a tale of terror. I spent one month refactoring code (ironically, for performance), testing on only the Windows build, just assuming things would work fine on the Xbox. When I finally fired it up on the console, the frame rate was through the floor. To make a long story short, I spent a month trying to track down the problem, and ended up rolling back in source control all the way to the original SOULCASTER I code base. (A couple weeks later I found the problem, and it was one line in the sound player initialization which caused some sort of near-infinite loop.) So please, test on target frequently.

3 /// **Work habits**. I do my best work when I put in three to five hours a day consistently, and this practice served me well in making the first game. I lost touch with this in making the sequel, and found myself stuck in the cycle of crunch, recuperate, repeat. I wasn't getting as much done, and it wasn't until later in the project that I found my footing again with this "energy conservation" model of game development.

4 /// **Promotion**. Being part of the Winter Uprising promotion took care of a lot of my marketing needs with free press coverage, but I probably relied on it too much. I could have done more to rally the SOULCASTER fan base in anticipation of the game, put together a proper trailer, and so forth.

1 /// **Self-doubt**. Maybe this can get me nominated for "Most Emo Bullet Point." As a lone wolf indie developer, it's easy for me to get derailed by anxiety over how my game is going to be received. This wasn't a paralyzing force, thankfully, but in a subtle, pervasive way, I found myself judging my work against what I assumed others would say about it. While it was helpful to make something fun for other people, this should have been saved for the playtesting phase. Only when I reconnected with the fun of making a game that I wanted to play, first and foremost, did I get back into full swing.

GAME DATA

SOULCASTER 2

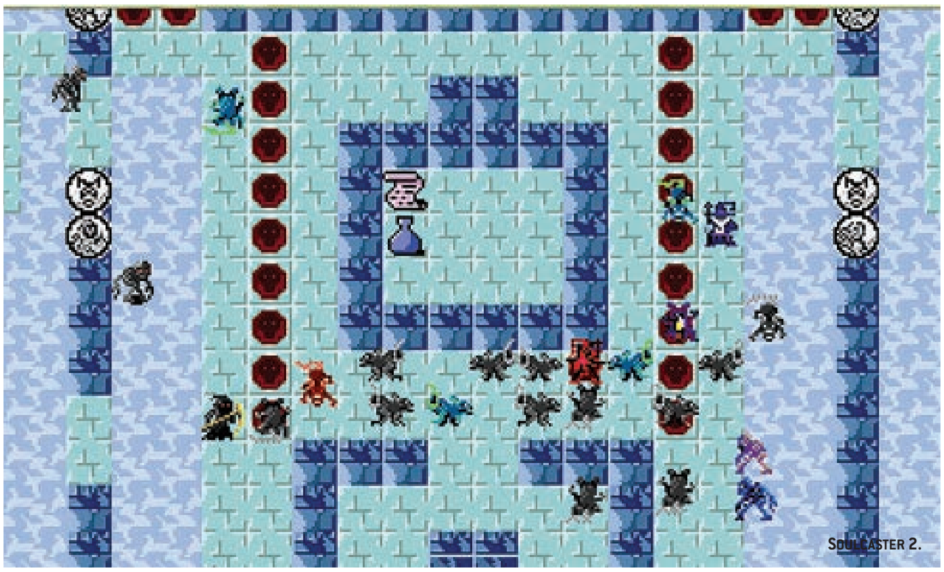
PUBLISHER	MagicalTimeBean
DEVELOPER	MagicalTimeBean
NUMBER OF DEVELOPERS	1 full time, 2 contractors
LENGTH OF DEVELOPMENT	7 months
RELEASE DATE	December 10, 2010
SOFTWARE	Visual Studio 2010, Subversion, GraphicsGale, Paint.NET, Impulse Tracker, Sound Forge, Ableton Live
PLATFORM	Xbox Live Indie Games
BUDGET	$600 + my time
LINES OF CODE	20,561
PIZZA BETA PARTIES	2

GOT SOME RARE THINGS ON SALE...

I am very proud of how SOULCASTER II turned out. Fans of the first game liked it, critics gave it high marks, and it sold reasonably well. Although my original intention was to finish the second game in less time than it took to complete the first, this is not how things turned out. My story proves that a few pernicious things, left unattended, can cost months.

My next project is an entirely new franchise which I hope to release in the first half of 2011. After that, I intend to break ground on SOULCASTER III.

IAN STOCKER *is the creator of SOULCASTER I & II. He has worked as a musician and sound designer for Nintendo handheld systems such as Nintendo DS, Game Boy Advance, and Game Boy Color, and racked up over 50 game credits in the past 8 years. The creation of an authentic 16-bit style console game is the realization of his lifelong dream.*


SOULCASTER 2.

THE **38TH** INTERNATIONAL
CONFERENCE AND **EXHIBITION** ON
COMPUTER GRAPHICS
AND **INTERACTIVE TECHNIQUES**

VANCOUVER
SIGGRAPH2011 MAKE IT **HOME**

CONFERENCE 7-11 AUGUST 2011 **EXHIBITION** 9-11 AUGUST 2011

Wherever SIGGRAPH travels, HOME is where smart people gather for an irresistible mix of energy, artistry, and the latest ideas in computer graphics and interactive techniques.

VANCOUVER CONVENTION CENTRE

Sponsored by ACM**SIGGRAPH**

WWW.SIGGRAPH.ORG/S2011

E3 Expo is the preeminent global trade event for computer and video games. It's all about the innovation, creativity, business, and imagination of the most compelling sector of the entertainment industry.

EXPERIENCE ALL THAT E3 EXPO HAS TO OFFER BY REGISTERING TODAY AT **WWW.E3EXPO.COM.**

esa entertainment software association

LOS ANGELES CONVENTION CENTER

JUNE 7-9, 2011

WWW.E3EXPO.COM

Produced By IDG WORLD EXPO

E3 Expo is a trade event and only qualified industry professionals may attend.
No one under 17 will be admitted, including infants. Visit www.E3Expo.com for registration guidelines.

REVIEW BY DARIUS KAZEMI

DOMINIC SZABLEWSKI

Impact Game Engine

The Impact Game Engine is one of the first professional engines to run on the HTML5 Canvas element for making 2D games in JavaScript. Games created in Impact, like most HTML5 games, will run on "any modern browser," which is code for Firefox, Chrome, Safari, Opera, and Internet Explorer 9. Especially notable is that Impact also supports iOS devices out of the box.

The engine is targeted toward everyone from serious hobbyists to professionals.

THE WELTMEISTER LEVEL EDITOR

)))) Impact's most immediately impressive feature is its level editor, playfully dubbed Weltmeister. Weltmeister itself is implemented in HTML5, so it runs in your web browser. As a tilemap editor, it's fairly simple but gets the job done. It supports an arbitrary number of tile layers. Each layer is associated with one tileset, though at the moment, tilesets are strictly limited to square tiles. Although there is no built-in support for custom brushes, drawing maps feels natural, and the overall UI is every bit as good as you'd expect from a desktop map editor. Levels are stored in JSON format and are very easy to parse programmatically if you need to do on-the-fly level editing or procedural level generation.

Weltmeister is tightly coupled with Impact, so while it would not function well as a standalone editor, it does a great job of fitting into the Impact workflow—deploying levels in your game is a matter of edit, save, and run.

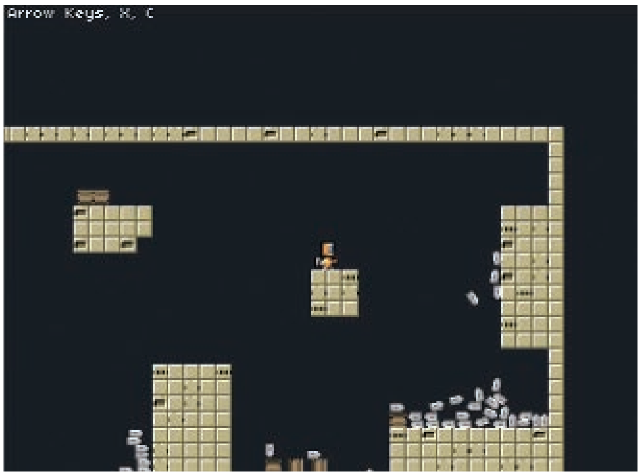
MOBILE DEVICE SUPPORT

)))) One of the major benefits of developing games in HTML5 is that in addition to running on modern browsers, your games will also run on many mobile devices. Impact supports iOS devices including the iPhone 3GS, iPhone 4, iPod Touch (3rd Generation), and the iPad. The engine detects the device it is running on and provides environment variables that make it easy to do any device-specific configurations you want.

The Impact website provides an HTML template that will render well on all iOS devices, as well as a PHP file that will redirect to different templates based on each browser/device, if you prefer to use that method. The website also provides a fairly comprehensive guide, outlining the kinds of optimizations you will want to do to get an Impact game running smoothly on as many kinds of iOS devices as possible.

Starting with a test platforming game I'd made in Impact, I was able to use the provided templates and guides to port the game to the iPad in only a few hours' time.

Impact does not provide official support for Android or other mobile HTML5-ready devices, although many of its features do work on non-iOS devices. An important note is that audio support on iOS is limited to one channel of sound, although that is a limitation of the iOS platform and not Impact itself.



Impact integrates well with external libraries, including the many JavaScript flavors of Box2D.

MODULE SYSTEM

)))) JavaScript does not understand the concept of includes, so Impact provides a module system that allows you to organize your code. Here's an example of how a main game file includes the necessary sub-modules:

```
ig.module(
        ´game.main´
)
.requires(
        ´impact.game´,
        ´impact.font´,
        ´game.entities.
dynamicActor´,
        ´game.entities.
player´,
        ´game.entities.
enemy´,
        ´game.levels.main´
)
.defines(function(){

// main game code

});
```

So even though the game consists of multiple modules in multiple files, the HTML file for your game only needs to include main.js. Seasoned programmers unfamiliar with JavaScript might take this kind of thing for granted, but if you've ever programmed a game in JavaScript before, you'll know that code organization can be a big pain in the neck. Impact's module system takes the focus off of housekeeping and allows you to focus on actual programming tasks.

BAKING YOUR CODE FOR PERFORMANCE AND PROTECTION

)))) Impact comes with a script that "bakes" your game into one big JS file and minifies (compresses) the code. This reduces loading time by decreasing the number of HTTP requests and packing things into smaller file sizes. In addition to reducing load time, the baking process serves the

DOMINIC SZABLEWSKI ANWENDUNGSENTWICKLUNG
IMPACT GAME ENGINE

Georgenstr. 14 A
64572 Büttelborn
Germany
http://impactjs.com/

PRICE
› $99

SYSTEM REQUIREMENTS
› Internet connection

PROS
1 Very easy to make games that run on both desktop and iOS
2 Great documentation and support
3 Weltmeister level editor can greatly speed up workflow

CONS
1 No official Android support
2 No trial license available
3 No online wiki or knowledge base; useful information is sometimes buried in a hard-to-search forum system

The Impact Game Engine's flagship demo, Biolab Disaster, is as impressive as any high-end Flash-based platformer.

additional purpose of obfuscating your code. This is especially important as many developers are rightly concerned about anyone being able to press Ctrl+U in their browser, which allows users to view the source code of a game as they would the HTML of a website.

Baking also happens to be *required*, in order to comply with Impact's license; without it, you will end up distributing the source code to Impact along with your game.

SUPPORT AND DOCUMENTATION

)))) The documentation for Impact is both extensive and easy to use. On the Impact website, the developer provides a reference for each of the main classes and their functions, and most functions are accompanied by helpful example code. There are also articles covering major topics such as collision, animation, Box2D integration, and mobile porting, as well as a pair of helpful video tutorials that can get you started in a matter of minutes.

Support for the engine is also good. While Impact is created and maintained by the one-person "team" of Dominik Szablewski, he is quick to respond on the forums and fix bugs, and often incorporates customer feedback into new builds of the engine.

There is also a fledgling community of Impact users providing plugins, most notably multiplayer support via NodeJS. At the time of this writing, the concept of Impact plugins is only two weeks old, but early contributions look encouraging in both quantity and quality.

PRICING AND LICENSE

)))) An Impact license costs $99, and is granted on a per-developer basis, although bulk rates do appear to be available if you wish to obtain a license for a studio of more than a handful of people. The license grants you unlimited use of the engine, and a discounted upgrade to future major engine releases. With a purchase, you receive access to the latest stable build as well as the git repository of the development branch. There are no trial licenses available.

THE BOTTOM LINE

)))) While there are many free JavaScript game frameworks available online, the Impact Game Engine is clearly worth its license fee. The combination of orderly code, iOS support, and great documentation puts Impact head and shoulders above the competition.

Beyond all that is something a little more intangible: I've tried four other JavaScript game engines, and this is the first one I've used that makes sense and feels like it was developed by someone who understands game development, as opposed to a web developer who is dabbling in games. A programmer should be able to quickly and easily transition to Impact from something like pyGame or XNA. In particular, indie game developers who currently use Flixel to make games in Flash/AS3 will find Impact extremely familiar.

I'm willing to go out on a limb and say that Impact is the first truly professional-grade JavaScript and HTML5 game engine to hit the market. It has set the bar high for future competitors, who will undoubtedly be releasing their own engines.

DARIUS KAZEMI *is lead analyst at Blue Fang Games, located just outside of Boston. He serves on the board of directors of the International Game Developers Association and blogs at Tiny Subversions.*

product news

Morpheme 3, FMOD, Vision Engine, Wwise Add NGP Support

WWW.NATURALMOTION.COM

NaturalMotion says its Morpheme animation middleware will include support for Sony's Next Generation Portable, as the company suggests it's a natural shift from supporting PlayStation 3 to supporting the high-fidelity handheld.

"We are excited by the possibilities that SCE is opening up with NGP," said NaturalMotion CEO Torsten Reil.

"With Morpheme already powering numerous PlayStation 3 titles, it is amazing to now have the performance to run our technology at the same fidelity on a handheld device."

NaturalMotion released version 3.0 of Morpheme in December of 2010 with support for Kinect and Move games. The middleware is designed to allow for real-time graphical authoring and animation previews, asset management, and other features for animators.

It's currently licensed by Ubisoft, THQ, BioWare, and numerous others, the company said. Because NaturalMotion had access to NGP prototype hardware, it says that the NGP version of its middleware is available immediately.

In addition to NaturalMotion, Eningen, Germany-based Trinigy said its Vision Game Engine will support Sony's upcoming handheld. New features for the NGP version of the Vision engine include an "optimized" character skinning system, and full support for hardware features including touch controls, improved multithreading, and an optimized renderer.

Also confirming NGP support is Australian audio middleware firm Firelight Technologies, which said its FMOD audio suite is authorized for NGP development. In late November, Vision and Firelight announced that FMOD will be bundled with Trinigy's Vision Engine 8.

Separately, Montreal's Audiokinetic said the NGP is the first portable console supported by the company's Wwise audio solution, which is already available for NGP development.

–LEIGH ALEXANDER, KRIS GRAFT

Latest EKI One A.I. Middleware Integrates With Unity3D

HTTP://UNITY3D.COM

The Puchheim, Germany-based Artificial Technology announced that the latest version of its EKI One A.I. middleware will be integrated into Unity Technologies' widely used Unity3D game engine.

EKI One 2.6 connects with Unity in the runtime environment as a natural part of game design workflow, Artificial Technology said. Developers can make changes that are automatically applied when running the game and while designing in Unity, according to the middleware provider.

The company said it has also improved user-friendliness with the EKI One Configurator graphical interface, and now offers wide-ranging extensions for online game behavior solution EKI One Server.

–KRIS GRAFT

Scoreloop Adds Windows Phone 7 Support

WWW.SCORELOOP.COM

Scoreloop, provider of social networking and in-game monetization tools for mobile game developers, has launched a beta version of its SDK for Windows Phone 7, adding Microsoft's new platform to a list of Scoreloop-supported operating systems including Android, Bada, Airplay, and iOS.

"Windows Phone 7 is an important step in our goal to encourage and support connected gaming across the entire mobile landscape," explained Scoreloop CEO Mark Gumpinger. He said that the company aims to support as many mobile platforms as possible to help developers expand their reach and revenue opportunity.

Scoreloop, which lets developers add in-game purchases and implement virtual currencies as well as social features, will now be usable with Windows 7 games so that players can "compete and compare" cross-platform in a similar fashion, the company said.

–LEIGH ALEXANDER

OpenFeint Launching Cross-Platform Social Gaming API

WWW.OPENFEINT.COM

Gamers on Apple's iDevices who want to compare scores can already do so through the company's GameCenter. And cross-platform data is just starting to take off. But what happens when their friends are playing the same game on an Android phone, or on the PC, or on many more diverse platforms?

OpenFeint is hoping to build a bridge to address that issue. The company has announced the private beta launch for OpenFeint Connect, an API solution that will allow developers to use OpenFeint's features on any app store, for any device, and incorporate its game data much more easily.

While the system, based on RESTful APIs, will not support cross-platform play, users will be able to compare scores on a centralized leaderboard and find new friends regardless of the system they use. Developers, meanwhile, will be able to send a single widespread message to players of their games to alert them to updates.

"We believe games should connect people, regardless of what device or mobile OS they own," said Jason Citron, founder and CEO of OpenFeint. "With the release of OpenFeint Connect, we give game developers the flexibility they need to take their games and game data to players everywhere."

Since the API is more universal, it will allow OpenFeint to support a number of devices beyond iOS and Android, including Windows Phone 7, PCs, and the Mac App Store. Through OpenFeint Connect, developers can access the platform's biggest features without having to use an OpenFeint-built client, the company said.

– CHRIS MORRIS

Zong Expands Mobile Account Payment System To PC, Consoles, Android Tablets

WWW.ZONG.COM

In-app mobile payment provider Zong has announced a massive expansion of its system, allowing payments linked to mobile phone accounts to be included in "nearly any [application] environment," including game consoles and PCs.

Previously available only on Android and through HTML pages, Zong's new system boasts compatibility with games made in Unity and Flash, as well as titles for interactive TVs, Android tablets, and consoles.

Zong's system streamlines payments on these platforms, the company says, by linking them to a text-message-confirmed phone number provided by the user. Rather than requiring the user to dig out a credit card or sign up for a separate payment account, Zong-enabled payments are billed directly to the player's mobile phone bill with minimum hassle.

It's unclear whether games on console services like Xbox Live Arcade and WiiWare will be allowed to accept such in-game payments, rather than payments through those platforms' exclusive currencies.

Zong faces competition from digital account services like PayPal, as well as other mobile-account-based payment providers including Boku. Last December, ANGRY BIRDS maker Rovio announced plans for its own cross-platform payment solution run through mobile phone bills, dubbed Bad Piggy Bank.

–KYLE ORLAND

HEROES WANTED

Technical | Creative | Digital | Analysis | Project Management

Our website: www.us.playstation.com/jobs
Visit us at our E3 booth in June 2011

THE FUTURE OF BROWSERS

A PRIMER FOR HTML5 AND OTHER MODERN BROWSER GAME TECHNOLOGIES

Step 1. See a link for a game.
Step 2. Click it... click it with desire.
Step 3. ...

Well, let's think about Step 3. What would you like to have happen? My answer: I'd like to be playing the game. No marketing page, no overview page, no screenshots or videos, no download page, no installation, no plug-ins, nothing but the game. I came to play the game. Put anything between me and the game and, well, I might not make it there.

Step 3, you're playing the game. The only way this happens is with browser tech. Browsers are ubiquitous—everyone uses them—and over time having your game run right from a link will become increasingly essential as other services and games all work that way.

Now, I'm an old school dev. My first code shipped on PlayStation. We worried about sorting triangles since there was no Z buffer, and splitting them as there was no perspective-correct texturing. Until a year ago, I shrugged off web tech with arrogant disdain. I've caught wind of this change, and now it's only too clear how things will pan out. It won't be sudden, mind you, but it's coming.

This article is for me a year ago. It's a heads-up on a whole range of technologies relevant for games, all built right into browsers. Most of them are still being standardized and implemented, and a few are still in the R&D phase. It's an interesting time, too, as the developers who push them now will influence the next generation of browser game engines.

Let's get some things out of the way before diving into the tech.

I won't be discussing plug-ins, or web tech that's been widely known for a while. Plug-ins are out for a few reasons: they're a known quantity, they can be a major security issue, and they lose potential game players by requiring installation. Sure, Flash has a great install base... except on iPhone and iPad.

Mobile and handheld devices will drive the markets. Worldwide sales of mobile devices are skyrocketing, outpacing desktops. They're where a lot of the energy and innovation are going. And, for significant parts of the world's population, the only computer a person will have will be mobile.

And finally, what is HTML5? What's with the "Other" in the title of this article? Well, in the end, it's not terribly important. I'm covering technology that is relevant for game developers. Much of it happens to be covered by industry specifications, some of which are part of the HTML5 specification. Each specification is at varying levels of finalization, implementation, and adoption. It's really best to just treat each tech independently. So, do let's dive in.

THE TECH

>> I'll be giving an overview of each, offering code samples and images.

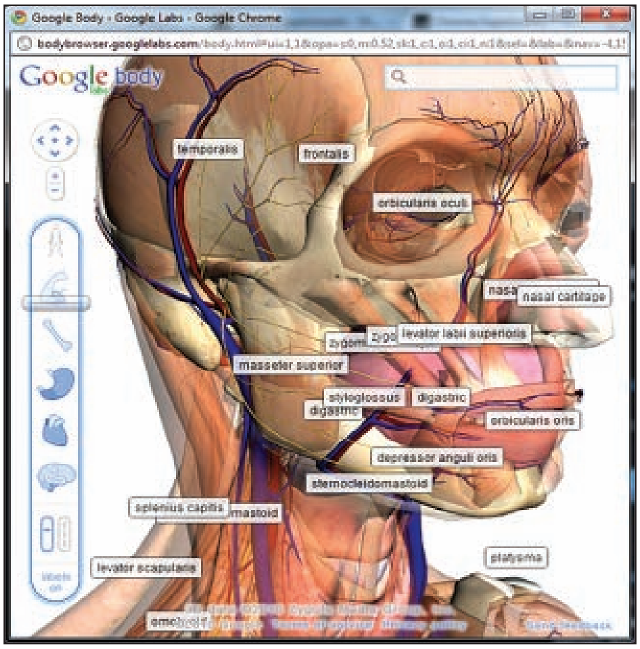
Web GL

Excitement around WebGL has been growing over the last year. Chrome shipped with it in February, and the 1.0 Specification was ratified at the Game Developers Conference in March. Firefox 4, Safari, and Opera have also announced implementations.

WebGL is essentially Open GL ES 2.0 as a JavaScript API, and is notable in the long history of attempted 3D in the browser because it is not a plug-in; it is natively supported. The API consists primarily of GLSL vertex and fragment shaders, textures, frame buffers, and context states such as blending.

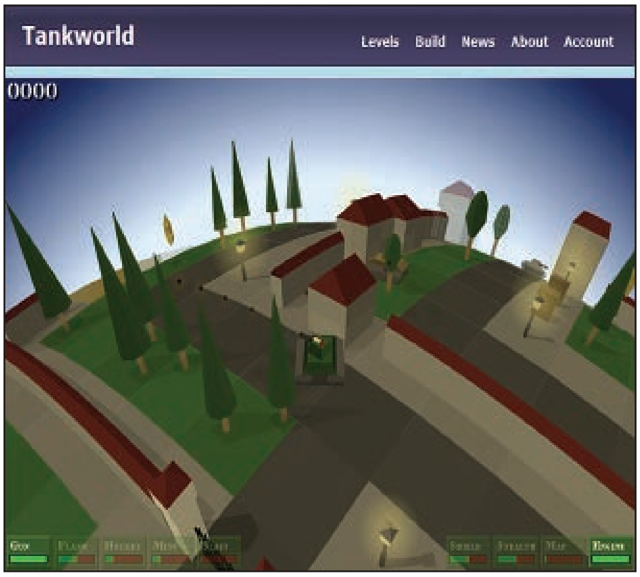
There are many libraries available, ranging from simple matrix and utility code up to 3D engines supporting data loading, animation, shadows, portals, and more. Here's a starter list of libraries: GLGE, C3DL, Copperlicht, SpiderGL, SceneJS, three.js, O3D, Processing.js and XB PointStream, and WebGLU.

WebGL Inspector deserves a special call out. It's your PIX/nvPerfHUD/gDEBugger equivalent, all implemented in JavaScript, and is easy to snap in place with only a few lines of code. It enables you to replay a frame, get the history of a pixel, view data buffers, textures, GL state, shaders, and constants.

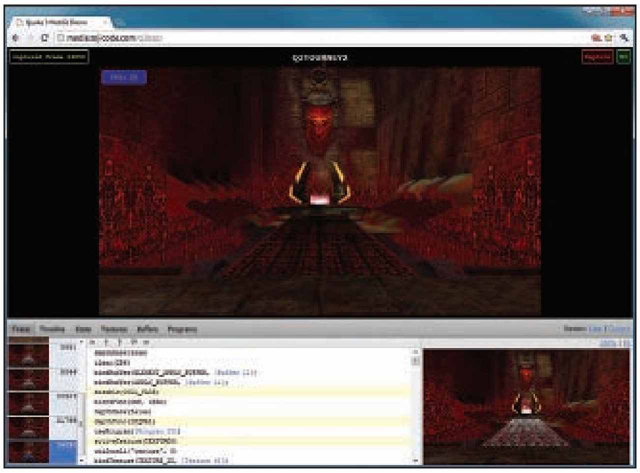


WebGL Sample Google Body explores 3D layers of the body.

Canvas 2D Sample of retro game ONSLAUGHT.



Canvas 2D can also be used for 3D rendering, as in TANKWORLD.



WebGL Inspector offers debugging and analysis of your WebGL application.

WebGL code looks similar to normal Open GL ES code.

```
matrix.makeIdentity();
matrix.rotate(currentAngle, 0,1,0);
gl.uniformMatrix4fv(uniformLocation, false, matrix.array);
gl.drawElements(gl.TRIANGLES, numIndices, gl.UNSIGNED_BYTE, 0);
```

Canvas 2D

Canvas 2D saw a spike in popularity in 2010, due to its simplicity and broad availability. Many Flash-like games have been created with this immediate-mode API, which handles images, vector shapes, and text.

Sprite-based games are common. Performance ranges from dozens of sprites on low-end smart phones, to hundreds on software-rendering desktops, to thousands on GPU-accelerated desktop browsers.

GPU acceleration is new in browsers, though. Expect particulars to change over time. Also, Canvas 2D is an immediate-mode API, meaning all data for each draw call must be specified every time. There will be performance limits that retained APIs would alleviate due to the difficulty of caching primitives.

The Canvas 2D API supports several operations, such as rectangles, circle arcs, lines, Bézier curves, text, and images; fills and strokes; effects such as gradients, shadows, line styles, and pattern fills; and transformation stacks. The features are an interesting mix, owing their heritage to Apple's Mac OS APIs. While

CANVAS 2D CODE SAMPLES

A simple example of loading an image of a cat and rendering it to a canvas:

```
<canvas id="e" width="200" height="100"></canvas>
<script>
  var context2d = document.getElementById("e").getContext("2d");
  var cat = new Image();
  cat.src = "images/cat.png";
  cat.onload = function() {
    context2d.drawImage(cat, 0, 0);
 };
</script>
```

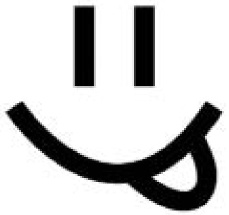
An example of the vector APIs (adapted from an Internet Explorer 9 sample):

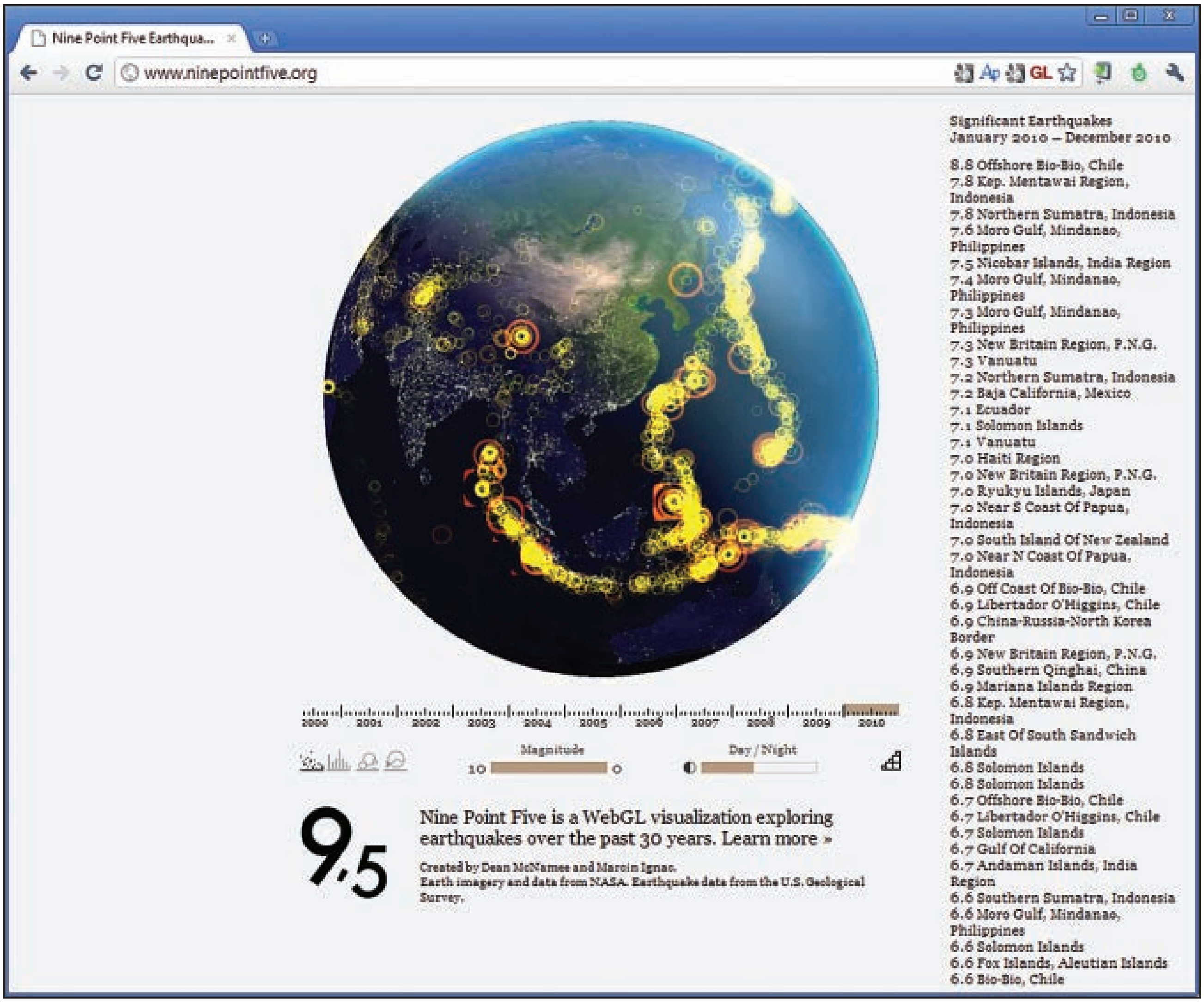
```
// Draw eyes
ctx.fillRect(160, 130, 20, 80);
ctx.fillRect(220, 130, 20, 80);

// Draw mouth
ctx.beginPath();
ctx.moveTo(100, 230); // Start smile
ctx.bezierCurveTo(100, 230, 200, 380, 300, 230);

ctx.moveTo(219, 298); // Start tongue
ctx.bezierCurveTo(278, 351, 315, 315, 277, 258);

ctx.lineWidth = 20;
ctx.stroke();
```





Significant Earthquakes
January 2010 – December 2010

8.8 Offshore Bio-Bio, Chile
7.8 Kep. Mentawai Region, Indonesia
7.8 Northern Sumatra, Indonesia
7.6 Moro Gulf, Mindanao, Philippines
7.5 Nicobar Islands, India Region
7.4 Moro Gulf, Mindanao, Philippines
7.3 Moro Gulf, Mindanao, Philippines
7.3 New Britain Region, P.N.G.
7.3 Vanuatu
7.2 Northern Sumatra, Indonesia
7.2 Baja California, Mexico
7.1 Ecuador
7.1 Solomon Islands
7.1 Vanuatu
7.0 Haiti Region
7.0 New Britain Region, P.N.G.
7.0 Ryukyu Islands, Japan
7.0 Near S Coast Of Papua, Indonesia
7.0 South Island Of New Zealand
7.0 Near N Coast Of Papua, Indonesia
6.9 Off Coast Of Bio-Bio, Chile
6.9 Libertador O'Higgins, Chile
6.9 China-Russia-North Korea Border
6.9 New Britain Region, P.N.G.
6.9 Southern Qinghai, China
6.9 Mariana Islands Region
6.8 Kep. Mentawai Region, Indonesia
6.8 East Of South Sandwich Islands
6.8 Solomon Islands
6.8 Solomon Islands
6.7 Offshore Bio-Bio, Chile
6.7 Libertador O'Higgins, Chile
6.7 Solomon Islands
6.7 Gulf Of California
6.7 Andaman Islands, India Region
6.6 Southern Sumatra, Indonesia
6.6 Moro Gulf, Mindanao, Philippines
6.6 Solomon Islands
6.6 Fox Islands, Aleutian Islands
6.6 Bio-Bio, Chile

Nine Point Five is a WebGL visualization exploring earthquakes over the past 30 years. Learn more »

Created by Dean McNamee and Marcin Ignac.
Earth imagery and data from NASA. Earthquake data from the U.S. Geological Survey.

WebGL Sample Nine Point Five, showing rich 3D visualization nested in classic 2D HTML.

it's mostly immediate mode with state stored on the context, gradients and patterns are objects. `drawFocusRing()` is for visual accessibility. `isPointInPath()` requires re-submitting a path, a structure that can only be defined in the context state. Compositing is specified globally with Porter-Duff operations (film full frame compositing), but most implementations only affect pixels modified by a draw operation similar to GPU APIs using src-dst ops. That said, most of this can be ignored and you can use the subset most useful to you.

SCALABLE VECTOR GRAPHICS

» SVG is a retained mode, declarative technology that has a longer history than Canvas 2D. While currently less popular, it has many good features.

Being declarative, there is an asset format in XML that tools such as Illustrator and Inkscape can export to. HTML5 standardizes the ability to place SVG inline with HTML, which is nice, though larger files are typically loaded as separate resources.

Being retained has pros and cons. Implementations can be more efficient by caching much of the data, and rendering the set of data can be faster in native code. There is also a bonus of mouse picking being implemented for you. However, the API can be more work when you're heavily manipulating objects frequently, and can be computationally expensive too. Modifying SVG DOM elements also causes the browser to refresh page layout unless you use `suspendRedraw`/`unsuspendRedraw`.

SVG is a larger and complicated specification compared to Canvas 2D, and implementations have different levels of support for e.g. filter effects and other features (Gaussian blurs, color matrices, displacement maps, and that sort of thing).

Declarative code is fairly straightforward, as shown below.

```
<!DOCTYPE html><html><body>
  <svg id="mySVG">
    <circle id="circle0"
            cx="100" cy="75" r="50"
            fill="grey"
            stroke="black"
            stroke-width="5"
            onmousedown="alert(´clicked´);"\>
    <text x="60" y="155">Hello World</text>
  </svg>
```

Procedural code is similar to manipulating HTML DOM:

```
<script>
    var circle = document.createElementNS(
"http://www.w3.org/2000/svg", "circle");
    circle.setAttribute('cx', 90);
    circle.setAttribute('cy', 90);
    circle.setAttribute('r', 30);
    circle.setAttribute('onmousedown',      "alert('no, me!');");
    document.getElementById("svg0").appendChild(circle);
  </script>
```

WINDOW.REQUESTANIMATIONFRAME

» We've covered several drawing technologies, but let's make sure you're playing friendly. Users can open a lot of windows and tabs with plenty of content. If everyone is drawing as much and as fast as they can, things won't pan out well. The old method of drawing in a webpage is to set a timer and blindly draw away.

Trying to draw at 60 frames per second looked like this:

```
setInterval(drawFunction, 1000/60) ;
```

What if the GPU is bogged down from other workloads? What if you're in a background tab? Who cares!?

The user does, and a better way is being released by browsers as this article is written. Here's how to let the browser call you as fast as it can render everything on the page.

```
function drawFunction() {
    //... Do Drawing Work, then ask to be called again:
    window.requestAnimationFrame(drawFunction);
}
// kick off first frame:
window.requestAnimationFrame(drawFunction);
```

If you're in a background tab, the browser can intelligently know not to call you. If you'd prefer a lower frame rate (because users prefer consistent FPS over unsteady but occasionally higher FPS) you can early out of your draw call by checking elapsed time.

Worried about some browsers not supporting this functionality yet? Make your app future compatible with this code from Paul Irish.

```
window.requestAnimFrame = (function(){
  return   window.requestAnimationFrame ||
           window.webkitRequestAnimationFrame ||
           window.mozRequestAnimationFrame ||
           window.oRequestAnimationFrame ||
           window.msRequestAnimationFrame             ||
           function(callbackFunc, element){
                window.setTimeout(callbackFunc, 1000 / 60);
  };})();
```

CASCADING STYLE SHEETS 3D TRANSFORMS

» There's another method to get 3D in your web page: CSS3 3D Transforms. I bring this up because it might be useful for a game's heads-up display (HUD) or other user interface (UI).

With HTML, it's easy to mix many components together and layer them on top of each other. Why build out a 2D HUD or UI and implement it in WebGL when you can just use HTML? Traditional games not in a browser have announced using HTML engines instead of implementing their own UI. If your game is in the browser, it's an obvious choice.

CSS3 offers 3D transformations on any web content. This can be declarative via CSS files, or controlled procedurally from your JavaScript. Items can be animated as well. Here's a simple example of applying a perspective tilt on a web page.

```
<!DOCTYPE html><html><body>
<div style="-webkit-perspective: 400;">
<iframe
    src="http://www.gdconf.com/" width = 1024 height = 768
    style="-webkit-transform: rotate3d(1,0,0, 15deg)">
</iframe>
</div>
</body></html>
```

That's a simple example, but with each element able to receive an arbitrary transform, more complicated effects are possible.

The transformed content preserves its HTML behavior. Mouse clicks, text selections, buttons, and input fields all still work just as in 2D.

AUDIO

» Audio in HTML5 can be inserted with an HTML tag, or controlled with JavaScript. Here's a simple example of loading a sound effect, detecting when it's fully loaded, and playing it later.

```
var audio = new Audio();
audio.addEventListener("canplaythrough", function () { audio.loaded =
true; });
audio.src = "treasure.ogg";
// (later...) Hey, I found treasure!:
if (audio.loaded) audio.play();
```

There are still some rough edges across all the implementations, however. Some browsers still have some latency issues, and some bugs remain. Additionally, on iOS devices version 4 and up only one sample can be played at a time.

Also, no one codec is supported across all browsers. You must pick two formats to serve out of MP3, Vorbis, and WAV. Multiple sources can be specified, or you can detect compatibility with code like this:

```
(new Audio()).canPlayType("audio/ogg"));
```

Libraries such as SoundManager 2 help by detecting compatibility and implementing fallbacks.

The future is definitely interesting, though. Mozilla has an experimental Audio Data API which provides audio sample buffer read & write access. They have demonstrations of DSP effects and visualizations that process the audio with Discreet and Fast Fourier transforms (DFT and FFT, respectively) (see image on page 39).

There is also the proposed Web Audio API, a more holistic API focused on high-performance implementation of features by the browsers, controllable via JavaScript. It provides for modular node graphs, allowing many applications to be described. Node capabilities include gain, delay, panning, convolution, FFT, 3D spatialization, and JavaScript processing.

VIDEO

» Video is now available as a first-class element in HTML—just add <video>. But it's also available for your Canvas 2D and WebGL games. The process is quite straightforward—what follows is an example of using video as a texture in WebGL.

```
 videoElement.play();
  videoElement.addEventListener("timeupdate", function () {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE,
videoElement);
    gl.generateMipmap(gl.TEXTURE_2D);
  }
```

Unfortunately, there is no single codec supported by all browsers. There's an evolving level of support in browsers for H.264, WebM, and Theora. Additional installations offer support via plug-in or OS component for missing H.264 and WebM support on some browsers (Chrome, Firefox, IE9). But these suffer from plug-in install friction. How many users will be lost if you expect them to install more software? The solution for the foreseeable future is to host video in multiple formats, most likely H.264 and WebM.

WEBSOCKETS

» Networked communication isn't new for web browsers, but it has always been a pull model. Techniques exist to simulate a server push model, but they have not had low latency or high-quality connections. They've been sufficient for instant-messaging-type communication, but not rich multiplayer games.

WebSockets enable low latency, full duplex, persistent connections. They upgrade from an HTTP handshake. Today, implementations support UTF8 data (everything serialized to a string), but binary is coming. And it's very simple, as below.

```
var socket = new WebSocket("ws://server.com");
socket.onopen = function(event) { socket.send("Hello Server"); }
socket.onmessage = function(event) { alert("Server says: " + event.
data);
```

At the time of this writing, Firefox and Opera are putting their support behind development flags. The specification is still being stabilized, which will potentially require servers to update for protocol adjustments.

There are also interesting topics still at the discussion level and not yet part of a specification, such as peer to peer connections and unreliable transport (vs. TCP).

WebSockets will be important, though, for bringing about much richer and more intimate multiplayer game experiences, as opposed to the fairly asynchronous or laggy experience of previous multiplayer web games.

I have two library recommendations as well: Socket.IO is a higher-level library, which includes transport fallbacks onto older techniques (Ajax, Flash, and the like) and more utility. Then there's web-socket-js, a shim implementation using Flash.

NODE.JS

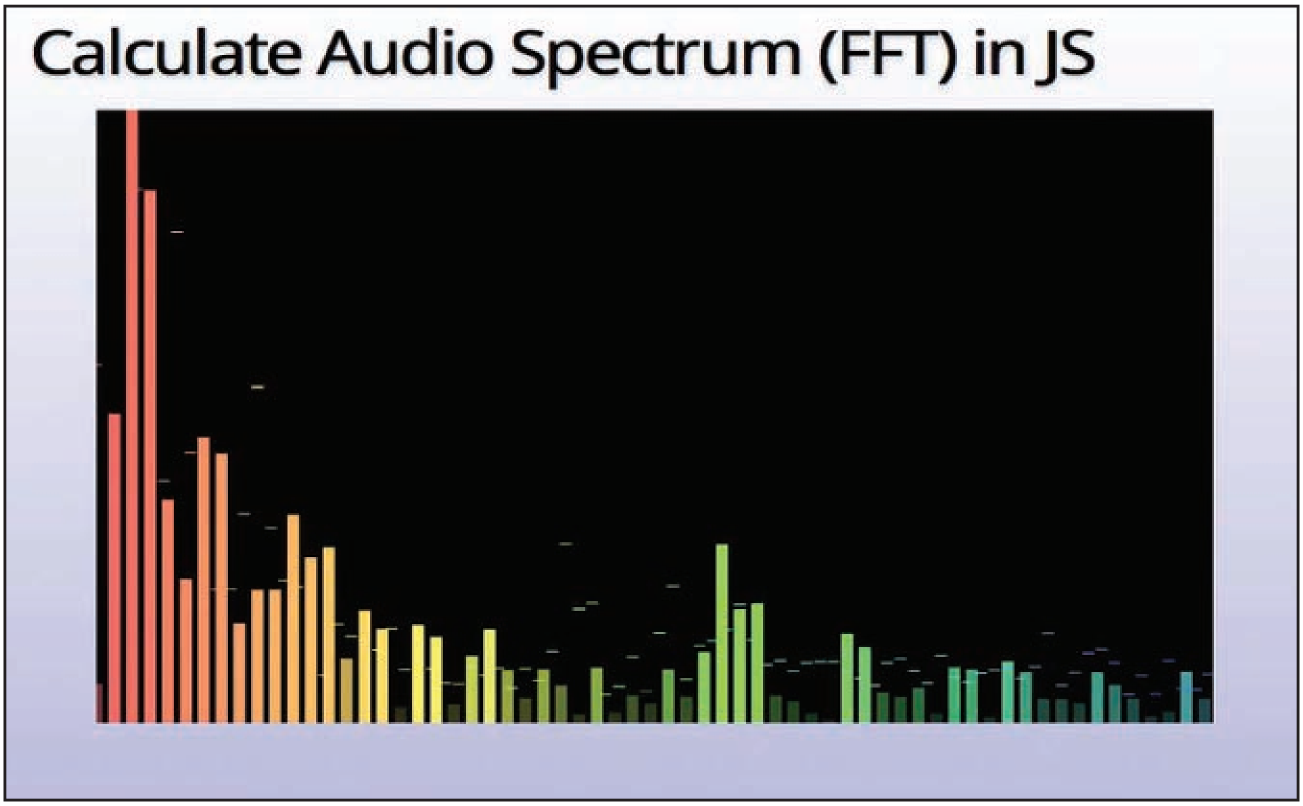
» On the topic of networking, I should mention the server technology Node.js. It runs JavaScript via the V8 engine (the one used by Chrome), which means that you can use the same language in your client and server code.

There are many modules available to add features or bridge to other technologies, such as databases, authentication and crypto, server-side graphics, routing, and so forth. WebSockets can be supported by several Node.JS modules, including Socket.IO, which I mentioned for client-side as well.

Visual debugging is also available, using the Eclipse V8 debugging tool, or the node-inspector tool, for example. The latter uses the same WebKit Web Inspector user interface you're likely to use client-side.

GEOLOCATION

» Player location has interesting applications for games. It's easy to think of this when developing a mobile game, but it's relevant for desktop games too. The browser API to query location is simple:



Mozilla demonstrates DSP effects and FFT visualizations in JavaScript with its Audio Data API.

```
navigator.geolocation.getCurrentPosition(
    function (location) {
        doSomething(location.coords.latitude,
                      location.coords.longitude);  });
```

To protect users' privacy, browsers will only provide a location after prompting the user for permission.

DEVICEORIENTATION & DEVICEMOTIONEVENT

>> Though not as widely supported as location, orientation and motion events are increasingly available on phones, tablets, and some laptops. The orientation event enables you to orient your UI based on how a device is being held (landscape vs portrait). Using motion events, you can react to the linear and rotational acceleration of the device, meaning you can detect whether the user is tilting or shaking the device.

WEB WORKERS

>> Workers offer multithreading for your JavaScript. Without them, all page structure updates and JavaScript executions are performed serially. They're simple to use.

```
// File main.js (main thread):
var worker = new Worker(´doWork.js´);
worker.addEventListener(´message´,
       function(e) { console.log(´Worker said: ´, e.data); });

worker.postMessage(´Hello World´); // Send data to our worker.

// File doWork.js (the worker):
self.addEventListener(´message´, function(e) {
       self.postMessage(e.data + "? " + e.data + "!"); });

// Output "Worker said: Hello World? Hello World!"
```

Communication between workers is handled with messages. There are no shared objects, so all data passes through messages. This offers a simple programming model, but there is a fairly high cost to move data between workers. For high compute-to-data transfer ratios, workers can be a big win. They also offload the main thread, which is a bottleneck for page rendering.

A shared worker is a special worker that will not be created more than once, even if multiple instances of a web page are opened. With this, you can coordinate between tabs and detect whether a user has accidentally opened tabs more than once.

A SERIES OF CLIENT-SIDE STORAGE OPTIONS

>> There are several methods to save state and resources on the client. In the next few sections, I'll go into some detail on the most relevant. Since these can be confusing, here's a summary:

- **Web Storage** Widely available, easy to use key value pairs, but with limits.
- **WebSQL** A database API which has stalled due to the intractability of capturing SQL in a specification. It's supported in Chrome, Opera, Safari, iOS and Android. But as it's dying, I won't discuss further.
- **Indexed Database** The solution to the WebSQL problem, though few implementations available.
- **File API** Directories & System: Application-managed file system, which will cache your big resources.
- **Application Cache** Enables your game to run offline by controlling how the browser caches resources.

WEB STORAGE

This is strikingly easy to use for small amounts of data:

```
localStorage["PlayerName"] = name;
```

You can store up to 5MB of string data via this simple key/value API. The Web Storage specification includes Local Storage and Session Storage, which differ only in that session storage clears the data after a browsing session is closed.

Being widely supported and simple, it's hard to not use this API for any simple small game. It's advantageous over cookies, as those are sent to the server on every request.

There are downsides, however. It is a blocking, serial API. Also, it is not transactional, which exposes race conditions if multiple tabs are open with your game in them. That can likely be worked around by coordinating via a shared worker, but it's ugly. There is also nothing but an exception when you run out of your 5MB storage space. Support from all major desktop and mobile browsers is hard to overlook, though.

INDEXED DATABASE

>> Web Storage offers only the most basic key/value interface. The Indexed Database API has several notable features. Indexed records allow efficient search for records, and enable higher-level query languages to be built. Asynchronous calls provide better responsiveness by not blocking, and enable transactions. Transactions remove issues of mixed writing and reading operations from multiple instances of a page. In-order traversal and multiple values per key are also available.

This is clearly an important API moving forward, but the implementations are only ready for developer R&D at this point.

FILE API: DIRECTORIES & SYSTEM

>> I expect many games to have significantly more content worth caching as compared with general web applications, especially WebGL games. An ideal scenario is to be able to prefetch assets needed for a game's later stages while running earlier content.

The File API D&S provides for an application-managed file system. A server can provide archives of many data files, which the client code may then unpack and manage. Loading of models, textures, and other arbitrary binary data can be made efficient, even as game asset patches are released. The specification and implementations are at a point where developers can start R&D with them.

On a related note, there is also the "File API," which is different but useful. It allows web apps to ask the user for a local file, and then read and manipulate it. For example, a game could allow custom logos to be selected.

OFFLINE APPLICATIONS & APPLICATION CACHE

>> Games are some of the best applications to have when you've lost your internet connection. Thankfully, HTML5 gives you a way to make your game available offline via an Application Cache manifest file. In this simple example, some key resources are specified to be cached for offline use.

```
CACHE MANIFEST
      index.html
      stylesheet.css
      images/logo.png
      scripts/main.js
```

In script, the state of connectivity can be detected with an event, or by checking `window.navigator.onLine`. The state of the cache can also be queried and request an update from code.

More control is possible with the manifest file as well. Resources that require the user to be online can be listed, and fallbacks can be specified for server resources when the user is offline. That means instead of just caching a resource, an alternative resource can be provided when offline.

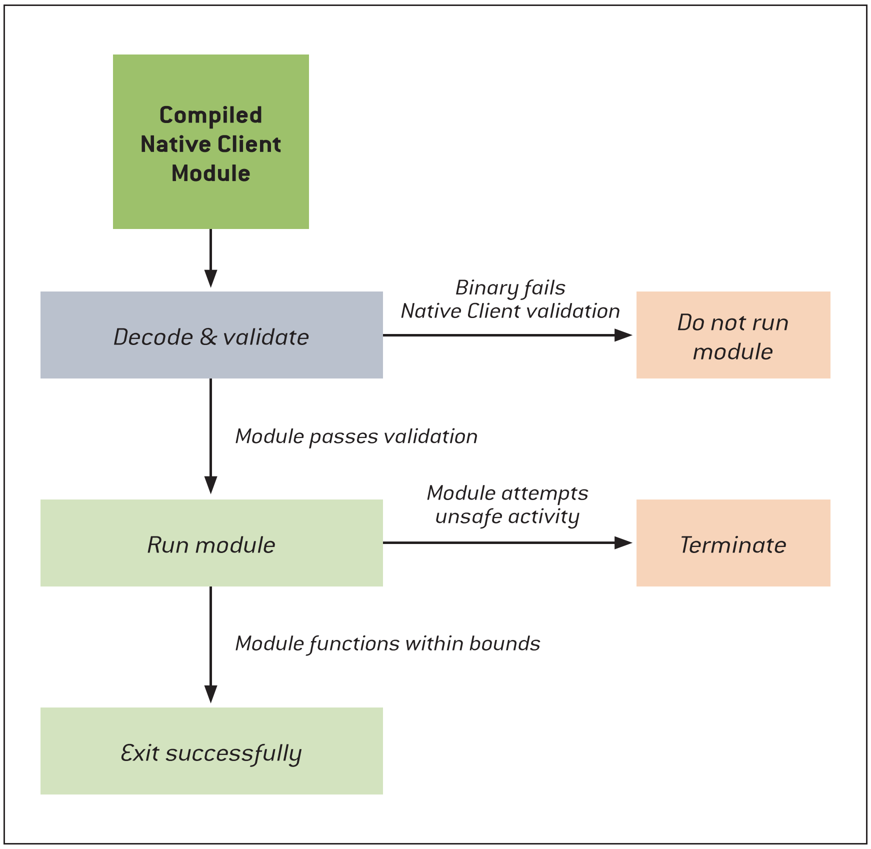
NATIVE CLIENT

>> All right C/C++ programmers, how did you make it this far? Sure, you're impressed that JavaScript can run FFT music visualizers and Box2D scenes, but you want "real" code running the browser. Is all this blah blah blah about JavaScript just not convincing you?

Well, fine. You can have C or C++, compile it down to machine code, and run it on any web page without even mentioning it to the user. You can run it as securely as you run JavaScript, and it'll run nearly as efficiently as a stand-alone application.

The key ingredient is that Native Client (NaCl) performs a static validation of your compiled code as it is loaded into memory and set for execution. It can do this by defining a subset of instructions and call patterns that can be efficiently validated. You generate this particular flavor of assembly by using a modified compiler, which the NaCl SDK provides.

Standard operating system libraries aren't available, for security reasons. Instead, a plug-in interface is provided that has a very POSIX feel to it—not too much work if you've already ported to systems such as the PlayStation 3. Low level 2D, OpenGL ES 2, audio, and HTML (like networking and storage APIs) are available as well.



Compiled Native Client Module
Decode & validate
Binary fails Native Client validation
Do not run module
Module passes validation
Run module
Module attempts unsafe activity
Terminate
Module functions within bounds
Exit successfully

Native Client sandboxes C++ code, allowing it to run as freely as JavaScript on web pages, with the same level of security. Previous methods of running C++ code, that is to say plug-ins, are not secure, and are the primary source of computer viruses and malware.

Unity games are the highest profile demos running to date. The engine has been demonstrated as ported to NaCl, which includes running C# code on Mono under the hood. So, if you love Unity but have a hard time getting users to install the player plug-in, look forward to a future where you can run the game plug-in free. Quake and the open source FPS Nexuiz have also been shown. That includes running C# code on Mono under the hood. Quake and the open source FPS Nexuiz have also been demonstrated.

Google Chrome is the only browser currently supporting NaCl, though it is an open source project, and available for other browsers to adopt. It is also still being stabilized—it's suitable for R&D, but not quite ready for final games.

WRAPPING THINGS UP

>> So, we just plowed through graphics, audio, video, networking, location, orientation, multithreading, storage, offline, and C/C++ code. That's a lot of tech coming online, all directly supported by the browser.

Popular technologies will be supported well by the different browsers, and the more you bang on them the more they'll fix issues. In this space, standards and competing implementations are held accountable by early R&D on your part, and public benchmarks.

Then there's the span across desktop and mobile. Many HTML5 technologies have appeared rapidly on mobile devices. Perhaps that's because most mobile browsers are based on WebKit, the foundation for Chrome, Safari, and others. The benefit is that cross-device development is that much easier, though not painless.

Distribution and monetization are interesting topics, too. On mobile devices, users have engaged well with Apple's App Store and Android Market. You should know that Chrome launched the Chrome Web Store, with similar benefits. Check it out as another option for getting your game in front of people.

VINCENT SCHEIB *works on Chrome like it's a game engine. Earlier he worked on Gamebryo and a few console games. Find him online with your favorite search engine, or the cheat code http://scheib.net.*

R E F E R E N C E S

A FEW OF THE BEST SITES AVAILABLE FOR INFORMATION ABOUT BROWSER TECH:

www.html5rocks.com
http://caniuse.com
(This site in particular has a lot of useful compatibility charts, which are frequently updated.)
http://diveintohtml5.org
http://developer.mozilla.org
http://gamedev.stackexchange.com

Game Developers Conference™ Europe

August 15–17, 2011 | Cologne Congress-Centrum Ost | Cologne, Germany

Visit www.gdceurope.com for more information.

Supported by

European Games Developer Federation   BIU   gamescom

UBM TechWeb

STACKING OVERFLOW

THE MUSICAL MAGIC OF DOUBLE FINE'S LATEST

ILLUSTRATION BY KELSEY KRAUS

Few developers are as identified with unbridled creativity as Tim Schafer's Double Fine Productions. Whether running through a summer camp for psychics or diving devil horns first into the world of an Iron Maiden album jacket, players have come to expect the unexpected from Schafer and Double Fine.

In February, Double Fine's art director Lee Petty gave players the PSN/XBLA title STACKING, which put them into the wooden shoes of Charlie, a micro matryoshka doll on a quest to rescue his family from a cabal of Industrial Revolution robber barons. Stacking is a smaller, download-only title, but is still jammed full of content. The game's audio balances its strengths between two pillars: a musical score that defines the game's setting, and a rich voice system that manages to express diverse characterization. All this is done without a single word of intelligible recorded dialogue. To see how Double Fine's audio team tackled the task of crafting detailed sound in a world inspired by silent films, I spoke with lead sound designer Brian Min and sound designer Camden Stoddard.

MATRYOSHKA MURMURS

>> As a pastiche of silent films, the world of STACKING sits at the confluence of Victorian Era luxury and Industrial Era grime, and is populated by a diverse cast of animated Russian stacking dolls. "When I first saw Lee's pitch," said Min, "it hit me that this world really needed a unique voice." Gameplay in STACKING is centered around solving puzzles by stacking Charlie inside of other dolls. Each doll has its own special ability, and abilities run the gamut from floating to flirting to farting. One of STACKING's stand-out features is its use of audio icons in a system by which the unique doll abilities each have immediately recognizable audio feedback.

"If there is one thing we needed to get right, it was the dolls' abilities," said Min. "In the advertisement world, they called it 'sound logos.'" These sound logos function as the game's main source of characterization, since STACKING has no recorded dialogue. Stoddard explained, "We felt that the sound for these abilities had to be concise yet solidly illustrate each doll's ability. If a sound was grating or boring, [there was a greater] chance that the player would avoid using that ability."

These sound logos, though, function within an interactive and reactive world. The diversity for the system is rooted in the notion that each sound logo "required a unique set of reaction effects," said Min. "The abilities don't work nearly as well without other dolls reacting to them. If you smoke a pipe, for example, all the other dolls around you start coughing. If you burp, the dolls run away in disgust. We have every kind of permutation for male, female, kids, and so forth, for every type of doll ability. We also have a dizzying array of custom doll chatter for every locale, and even various game states. Dolls will grumble during the workers' strike, but when the strike is over, they are nice and cheery."

According to Stoddard, a dynamic walla system was the second piece of the voice characterization puzzle. "We knew there would be no dialogue," he said, "but when we first started seeing the Train Station mock-ups, we realized that train stations have a ton of background chatter—lots of people on their way somewhere, reading things, conversing—but you're never fully aware of what they are saying. Without that chatter, it just felt empty. We would have scripts with scenes, but no text, just themes: 'This is the train car full of really rich, evil industrialists. They are doing under-the-table, shady deals, as well as commenting on the rather good buffet dinner.'"

MATRYOSHKA MUSIC

>> "When you think about it," said Stoddard, "most silent films were not silent. When presented to the public, there was piano accompaniment, and it was actually a very interactive audio experience. The pianist had to really convey story and emotion through the voice of the piano. The piano became a very illustrative and emotive character." For STACKING, it became very important for the team to include this same "invisible character" through the use of a mixture of licensed and original classical underscore.

"I wanted STACKING to have all live acoustic instruments and players," explained Min. "Ninety percent of STACKING's score was either licensed or re-edited versions that I constructed in Pro Tools. I didn't have stems, alt takes, or different mic perspectives, so the first step was a convolution reverb and EQ pass to make sure all the recordings sounded like they were done in the same room." In order to help sell the time period while making the music editing tasks easier, Min specifically focused on a number of solo piano pieces by Chopin. The rest of the licensed score uses music by Brahms, Tchaikovsky, Mozart, and Vivaldi, meaning that the game's original music—composed by both Min and Peter McConnell—was faced with the daunting task of creating original cues that fit seamlessly amongst music by some of history's greatest composers.

"Our soundtrack influenced the set pieces, abilities, and puzzles that would be in-game," Min concluded. "Even the dolls were inspired by our piano score. STACKING will always have a special place in my heart as the project that saw the realization of true collaboration between art, design, and audio."

JESSE HARLIN *has been composing music for games since 1999. He is currently the staff composer for LucasArts. You can email him at* ***jharlin@gdmag.com.***

THE END OF GAMES?

OR, WILL FREE-TO-PLAY SWALLOW THE INDUSTRY?

IN A GDC SPEECH FROM MARCH 2010, NGMOCO'S founder Neil Young described the advent of free-to-play gaming in the West as "the most significant shift and opportunity for [game developers] since the birth of this business." Since then, more and more game developers have been making this transition.

In June, Turbine announced that its profitable, subscription-based MMO LORD OF THE RINGS ONLINE would adopt a free-to-play model, based on the success of a similar change with their MMO DUNGEON AND DRAGONS ONLINE, which increased the game's revenues fivefold. In November, EA announced BATTLEFIELD PLAY4FREE, a downloadable, free-to-play shooter built on the BATTLEFIELD 2 engine, meant to improve on the success of the similar WW2-based BATTLEFIELD HEROES. In February of this year, Riot Games, developer of the popular free-to-play strategy arena-combat game LEAGUE OF LEGENDS, was purchased by Chinese games behemoth Tencent for $400 million, signifying the massive revenue potential of the format.

Indeed, few major franchises are not being considered as raw material for a transition to free-to-play; the revenue potential is simply too large to ignore. In March of this year, at the Bank of America Merrill Lynch 2011 Consumer Conference, Activision CFO Thomas Tippl stated that, shockingly, STARCRAFT 2 was not worth the effort from a financial perspective. Although the game sold very well, grossing over $250 million dollars, the lack of any ongoing revenue streams coupled with the high development costs meant that the final return on investment was simply not high enough. Ultimately, publicly traded companies, like Activision, must invest their money in projects with the highest potential profit margins and, increasingly, free-to-play games are dwarfing single-purchase games in that regard.

Neil Young's own ngmoco provides an interesting example of how this shift can change a company's priorities. The smartphone developer's first major hit was ROLANDO for the iPhone, providing the young startup with its first major revenue stream. However, the company soon discovered that, although they could make modest profits on single-purchase mobile games, their most profitable releases were free-to-play games built with in-app purchases. Their kingdom-builder WE RULE became the highest-grossing "free" game on iOS devices within a month of release.

Understanding that free-to-play games would be the only way to scale their business, ngmoco cancelled single-purchase games, including the guaranteed moneymaker ROLANDO 3, in favor of games which fit the freemium model. In Young's words, "If we can't make the game free-to-play, we're not going to release it." The strategy worked, as his young company was purchased by the Japanese social gaming giant DeNA in late 2010 for an impressive $400 million.

A NEW DESIGN

» Today's game designers will increasingly hear similar mandates from their own management teams. However, shifting to free-to-play design is not a straightforward process. Indeed, designers of single-purchase games have a much easier job, as they only have to focus on one thing: making the player's experience as much fun as possible. In contrast, designers of free-to-play games must make the game engaging enough to attract and retain players while also holding back enough of the experience to drive microtransactions.

The energy model is a proven mechanic that maintains this balance for many free-to-play games, including my own Facebook-based RPG DRAGON AGE LEGENDS. Under this model, certain player actions, such as starting a battle in LEGENDS, consume a set amount of energy. Once a player uses up all of her available energy, these actions are unavailable until the energy regenerates, commonly at the rate of one point every five minutes.

Thus, with a full energy bar, the player can typically fight four or five battles before getting stuck. At that point, they can decide to either let the energy recharge naturally, which might take two hours for an empty bar, or purchase an instant refill with real money. While they still have energy remaining, free players have access to the full game experience —the battles in LEGENDS do not work differently for paying players—but they have to deal with some impatience after they hit the energy gate.

In single-purchase games, designers rarely build a game mechanic that intentionally tests the player's patience; in fact, that is a hallmark of bad game design. Thus, free-to-play games upend many of the assumptions that designers bring to the table from traditional single-purchase design. Indeed, this break is creating a great deal of anxiety within the game design community, as many developers feel that their original motive for making games—to bring players as much fun as possible—is now in danger. Noted independent designer/programmer Chris Hecker recently voiced his own concerns in a post online:

"The problem I have with free-to-play is business types rarely talk about what you're giving up by going to that model. Microtransactions warp game designs, not necessarily for the better or for the worse, but they certainly make the designs different. If the profitability of microtransactions makes it so most companies go toward this model with their big-budget titles, then that is a shame and a loss, because there are lots of designs that are interesting and important to the art form to explore, but that don't lend themselves to microtransactions and free-to-play. I hope single-purchase, "complete experience" games don't go away; not because I'm old and curmudgeonly—although I am—but because there is an entire subspace of game design there that still needs to be explored."

A NEW HOPE

» LEAGUE OF LEGENDS, one of last year's most successful strategy games, provides an interesting case study which ultimately shows that Hecker's primary concern—that the design space explored by single-purchase games could be lost—is valid. The game is commonly held up as the best example of free-to-play design done right. Not only has LEAGUE OF LEGENDS been both wildly popular and commercially successful (see the $400 million purchase earlier), the game has also garnered critical acclaim, sweeping the first-ever Game Developers Choice Online Awards. Most importantly, the game's business model has been accepted peacefully by the core gamer community, one which typically views microtransactions with suspicion.

Because LEAGUES OF LEGENDS is a highly competitive, team-vs-team arena-combat game, microtransactions which could give one side an advantage over another would be wholly unpalatable to a large portion of the Western audience. Instead, the game only hands out bonuses to players who have invested large amounts of time in battle.

The meta-economy employs a dual-currency model common to many free-to-play games, with a time currency (Influence Points/IP) which is earned through play, and a cash currency (Riot Points/RP) which is bought with real money. Items which can boost a player's abilities (Runes) can only be purchased with the time currency (IP), providing players a strong incentive to keep playing to earn more IP. Cosmetic items, which only change the player's appearance, can only be purchased with the cash currency (RP), which simply appeals to a player's pride or vanity.

The player can also buy a temporary boost with RP which increases the rate at which IP is earned. This microtransaction presents a time-vs-money question to the player: he can spend some money now to earn Runes faster, or he can simply play some more games to earn the extra amount of IP required.

Still, the most interesting microtransaction is the character unlocks. LEAGUE OF LEGENDS descends directly from the popular WARCRAFT 3 mod *Defense of the Ancients*, in which the player gives up control of an army of units for control of a single hero. The mod's depth came from the different combination of hero types—103 in the current version. LEAGUE OF LEGENDS has a similarly large stable of "champions"—72 as of March 2011.

These 72 champions are not all available at all times. Instead, a rotating selection of around 10 are available each week, which means that players have only limited control over which champion they can use. This cycle can greatly upset a player who has become quite good with a specific champion but who must now learn a new one. Some players enjoy the challenge of mastering a new set of skills and attributes, but many others prefer to keep winning with a champion that works for them.

Accordingly, LEAGUE OF LEGENDS gives players the option to unlock champions permanently, with either IP or RP. Like the energy mechanic common in social games, the character unlocks are charging players for their impatience. Can they wait weeks until their favorite champion is again available for free or days until they earn enough IP to buy the unlock, or will they just spend some real money right now to get back into the action with their favorite character? This model works well for both gamers, who are getting an incredible experience for free, and for the developer, who can rely on player impatience to generate revenue.

Nonetheless, single-purchase games would never be designed this way, with players limited to a small sub-set of possible characters each week. (Indeed, the single-purchase strategy game COMMAND AND CONQUER 4 was roundly criticized for forcing players to "earn" the right to build certain units over multiple play sessions.) Although LEAGUE OF LEGENDS could make the transition to a single-purchase game by simply unlocking the majority of champions immediately, not all single-purchase games could just as easily make the transition to free-to-play.

For example, STARCRAFT 2 is not so easy to imagine as a free-to-play game. The franchise is the very model of a tight, elegant ruleset, with no extraneous parts or redundant options to muddy the design. Even with the new units added for the sequel, Blizzard removed enough old elements to keep the unit count down near the original 12 per race. Indeed, few reviewers even felt the need to comment on the lack of a fourth race to differentiate the two versions.

Could STARCRAFT 2 follow the LEAGUE OF LEGENDS model? Perhaps Blizzard could allow everyone to play the Terrans for free but only offer the Zerg and the Protoss to paying players? This model would probably fail both for business reasons (not enough opportunity for repeatable purchases) and for design reasons (having 90% of the players forced to use Terrans would destroy the balance). Anything more aggressive—like actually selling extra siege tanks during battles—would violate the concept of a fair playing field, a core tenet for strategy games.

AN OLD LESSON

>> Thus, if Activision believes that making a game of STARCRAFT 2's scope is not worth the effort, will this type of tight, intricate design simply disappear? Although microtransactions are still relatively new in Western video games, they are not new for physical games in the West. The emergence of *Magic* and other collectible card games (CCGs) in the 1990s showed the power of microtransactions by encouraging players to make recurring purchases within the same game system over many years. The makers of CCGs have been dealing with this new world, which necessarily mixes business and game design, at least a decade before us.

However, the wild success of CCGs—*Magic* still regularly grosses over $200 million yearly, which dwarfs non-collectible card games—did not send single-purchase, "complete experience" physical games into extinction. Indeed, the card and board game industry is more diverse and innovative than ever before.

In fact, two of the most successful card games of the last few years have taken a mechanic directly from *Magic*, adapted it to fit the format of a single-purchase game, and found commercial and critical success. Dominion turned the meta-game of a CCG into a traditional card game by having players build and play a deck of cards during the game. 7 WONDERS turned drafting, a folk method for distributing CCG cards, into a game by conducting a single draft after each individual card play.

These examples of successful single-purchase games emerging from the shadow of *Magic* prove this format can still thrive in a world with microtransactions. These designers brought some of the gameplay of *Magic* to a new audience simply because not everyone is ready to buy only a small part of a card game, which will never be complete. Many gamers will never be ready for CCGs, just as many gamers will never be ready to spend money on a free-to-play game.

Thus, if every video game adopts microtransactions, many players will be left behind who are looking for a different type of experience. Microtransactions do warp the game design toward a model that supports recurring purchases. But this shift leaves a great deal of space behind as a vacuum ready to be filled by smaller publishers and developers who are looking for great opportunities. The profits from single-purchase games can easily justify the development costs for teams that take their budgets seriously, and these profits can only go up as more and more big publishers abandon this still fertile design space.

SOREN JOHNSON *is a designer/programmer at EA2D, working on web-based gaming with strategystation.com and DRAGON AGE LEGENDS. He was the lead designer of CIVILIZATION IV and the co-designer of CIVILIZATION III. Read more of this thoughts on game design at* ***www.designer-notes.com***

take control of your future
www.gamasutra.com
the art and business of making games
UBM
TechWeb

BUILD YOUR OWN BUZZ

OPEN DEVELOPMENT, INDIE PR, AND YOU

THE TRADITIONAL GAME DEVELOPMENT PR STRATEGY IS TO CRAFT YOUR GAME in a secret cave where no one can steal your ideas or get a premature glimpse of your work. Then, just before your masterpiece is complete, you scramble to make trailers and get some previews and reviews lined up, pray to your deity of choice, and launch the game.

This may sound somewhat reasonable, but for an indie developer, it's actually pretty dangerous. You can't assume that the world will discover and care about your game when it's ready. By putting PR off to the very end of your development cycle, you are making the success of your game highly dependent on forces outside of your control.

THE PHILOSOPHY OF OPEN DEVELOPMENT

Instead of making excuses to close yourself off from the outside world, you should be doing whatever you can to reach out to it. At Wolfire Games, we have been using what we call "open development" for our upcoming ninja-rabbit fighting game, OVERGROWTH. From day one, we have been publicly showing off our development process, and this has allowed us to make noise, make friends, and build a community around our game—even before we had any gameplay!

Making Noise. Many developers assume that they don't have anything interesting to share with the world before their game is finished. But how many people out there have the courage to make a living by creating video games? By even attempting to make a game, you already have a lot of interesting things to talk about. You can share your thoughts about game design, creating art assets, building the tech for your engine, or even about other games that have inspired you.

It's hard to predict what will and won't be hot. So the sooner you start, the more things you can try, and the more likely you are to discover something that the outside world is really interested in.

Making Friends. With a traditional mindset, you might view your peers as competitors. You might think that blogging about someone else's game would dilute your brand and risk cannibalizing your sales. But that is not how the Internet works—it's not a zero-sum game. Cross-pollination is mutually beneficial. One of the most successful Wolfire blog posts was David Rosen's "design tour" video of WORLD OF GOO, in which he explained in great detail why it was such a great game. Not only was it a fun way to reach out to the awesome guys at 2DBoy, it also ended up bringing positive press to both of our studios.

Building A Community. At the core of every community is communication, which is a two-way street. It's nice to think that you can just beam out information to the internet and everyone will care, but it's also important to listen to feedback. If you share the progress of your game along the way and continue to make people happy, chances are you're on the right track.

STRATEGIES THAT ENHANCE OPEN DEVELOPMENT

Mod Support. Even before we started putting the ninja-rabbit combat into OVERGROWTH, fans had fun making their own custom levels and cities in our map editor. You'll find yourself surprised with the creative things fans come up with that you didn't even think about, if you make your tools available.

Supporting Mac and Linux. The personal computer space is the last place an independent developer can sell games directly to fans. So if you want to maximize your independence, you owe it to yourself to reach out to Mac and Linux users.

We have found that these communities tend to be noisy and very happy that we've made the effort to reach out to them. The sales data for Wolfire's original ninja-rabbit fighter, LUGARU, and more recently the sales data for the two Humble Indie Bundles, suggest that supporting Mac and Linux as an indie can actually double your revenue.

Early Preorders. If you're serious about making your game, you should allow fans to offer you early support. Early preorders give you something tangible to direct your PR efforts toward, and having more financial resources early in development increases your runway and allows you to stay indie.

Preorders work best if you can offer your fans something in return. Wolfire's approach has been to offer weekly alpha builds to anyone who has preordered.

THE MECHANICS OF OPEN DEVELOPMENT

Once you understand the appeal of open development, you still have to figure out how to implement it. Here are some tools we've found useful for expanding outreach.

Onsite Tools. The two must-haves are a blog and a forum. Your blog is one of the best tools for sending out information to the world, and the forum is the easiest place for your community to grow. These may be ghost towns when you first implement them, but don't be discouraged—they all start out that way. The sooner you have your blog and forum in place, the sooner you can begin accumulating visitors. When Wolfire first started work on OVERGROWTH, we were getting about 500 daily visitors to our blog. Two years later, our smallest posts get 5–10K visits.

Embedded live chat on your site can also be helpful. Olark is a great solution for real-time customer support. Wolfire even maintains a public IRC channel where people can hang out to chat with our team.

Finally, don't forget this all requires building a scalable, robust web site. Prepare for success so that the day your blog post goes viral and you're getting thousands of page views, everyone can still access your content. Nothing hurts more than doing everything right and having your big moment, only to watch your site crash. Google App Engine and Amazon Cloud Front have been integral in allowing Wolfire to maintain a scalable site.

Social Media. Social media is very useful for increasing the world's exposure to the fun activity on your site. As with your blog and forums, it's a good idea to stake a claim early so that you can grow your web presence over time. Traditionally, the Wolfire Blog has been the center of our PR efforts, and we've mainly used Facebook, YouTube, Twitter, and ModDB to echo our news.

What we've seen recently is that there is something inherently viral about videos. In general, people are more interested in watching videos about game development than they are in reading lengthy posts with static pictures. In the wake of David and Aubrey's weekly development update videos, Wolfire's YouTube channel has taken the lead as the largest and fastest-growing method of communication for us.

THE BIG PICTURE

If you think about it, what's more interesting to look at: a static finished piece of concept art or a time-lapse video showing the creation of the finished art from the first few strokes to the finishing lighting and shading? If you can perceive the appeal of the time-lapse, I think it's easy to see the appeal of open development. Instead of putting PR off to the last minute, you can turn your development process itself into a form of PR.

JOHN GRAHAM *is a Grower of Beards, the co-founder of Humble-Bumble, and founding-member of Wolfire Games*

LEGAL AGE

PHOTOSHOP TURNS 21. IT'S TIME TO GROW UP.

WE LIKE TO THINK WE'RE A CUTTING-EDGE BUSINESS THAT DEFINES NEW technological frontiers, which makes it kind of jarring to realize that one of our key tools is now old enough to drink.

Photoshop may not literally be the mother of all bitmap editors, but it's definitely older than many of the people reading this article. Released in 1990, Photoshop turned 21 in February, a pretty remarkable milestone for a piece of software in a business where almost everything is obsolete in less time than it takes to ship an installment of GRAN TURISMO.

Remarkably, the core of the Photoshop UI is still recognizable 21 years after the product first shipped. (In the interest of full disclosure: as a pre-1.0 beta tester in 1989, I complained that a valuable icon slot was given up to the "Hand" tool for a job that could be done with a hotkey or scrollbars ... Apparently I was wrong, since it's still there. See Figure 1.) Of course, there have been some very important improvements since then, particularly the introduction of layers in Photoshop 4 (1996), vector tools in version 6 (2000), and Smart Objects in CS2. On the whole, though, a Photoshop artist from the mid-1990s could be productive with the modern version of Photoshop in a day, and a modern artist would find a 10- or even 15-year-old version somewhat spartan, but still usable (see Figure 2).

So, take a moment to thank the Knoll brothers, Thomas and John, who built an impressively solid foundation for the product all those years ago. (You might also want to check out the two articles mentioned in Resources for a little more history of how the bitmap behemoth was born.) Having given some respect where it's due, let's return to a more natural, game-artist-friendly posture. In other words, it's back to the usual kvetching.

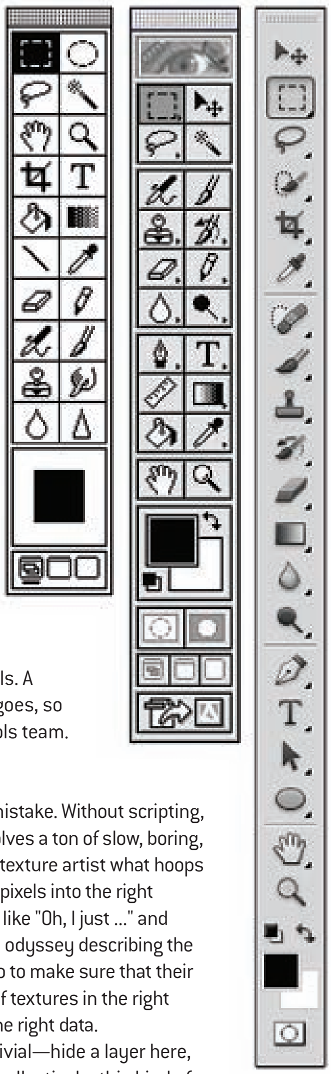
OFF THE SCRIPT

Photoshop does a great job pumping out pixels. The constant churn of game development, though, is continually imposing new needs and demands on us, and the steady, majestic pace of Photoshop's evolution doesn't always keep up with our problems.

Pretty much every studio supports Max, Mel, or Python scripts to smooth artists' workflows and clean up their data to suit the needs of a game. Although scripting was created mostly to simplify workflows, it has evolved into the glue that holds our pipelines together. Scripts aren't just for reducing button-clicks anymore; we use them to do everything from checking files out of source control, to submitting jobs to render farms, to formatting data for export to the game. Scripts are a vital way to cut down on the drudgery and clerical work that gets in the way of making art.

Strangely, though, few teams devote even a fraction of their scripting efforts to supporting texture artists and Photoshop users. Part of this, of course, is that the technical side of texturing is comparatively simple. We have to maintain squads of tech artists to support our 3D artists because the whole business is still embarrassingly clunky. In the texturing business, by contrast, things are rarely quite bad enough to stimulate a serious investment in tools. A bitmap is just a bitmap, the thinking goes, so it's not worth any energy from the tools team.

Figure 1: The evolution of the Photoshop toolbar, from 1.07 to 5.5 to CS5. That Hand tool is still around.



SLACKERS

This is a common idea, but it's also a mistake. Without scripting, the texturing side of art inevitably involves a ton of slow, boring, error-prone manual steps. If you ask a texture artist what hoops they have to jump through to get their pixels into the right places, they'll typically say something like "Oh, I just ..." and then launch into a three-minute verbal odyssey describing the myriad little operations they have to do to make sure that their working file produces the correct set of textures in the right formats, with the right names, using the right data.

Individually, these steps are all trivial—hide a layer here, save with a special name there—but collectively, this kind of song and dance slows down iteration enormously. It's also an easy way to create bugs when files are misnamed, the wrong layers are included, or some complex packing operation needed by a shader is botched. The fact that these steps are stupidly simple doesn't stop them from being an unnecessary and dangerous annoyance.

This is a perfect example of the kind of thing that 3D artists expect to be handled by scripts. We all know that 3D scenes are full of information that serves different purposes. Some of that information is for the game, but a lot of it is there for the convenience of the working artist, whether it's scale reference, template objects, or variant copies that help us visualize different alternatives. We use scripts and exporters to make sure we can filter out these private elements when sending data to the game while keeping them handy for our workflow.

INTEGRATING PHOTOSHOP WITH MAX AND MAYA

If having to write batch files just to do simple commands seems too limiting, JavaScript does allow you to write extension DLLs that can do more complex tasks using C or C++. Unfortunately, that's hardly "scripting" at that point.

A slightly less daunting form of rocket science is to get into Photoshop's automation system using COM. COM is a standard automation technology that older Windows applications use to talk to each other, and you can use Python, C#, or even MaxScript to communicate with Photoshop from the outside.

Not only does this give you all the features of those languages, it also means you can talk to Photoshop directly from inside Maya with Python or inside Max using C# or Maxscript. Volition's Adam Pletcher has a good introduction to Python-based Photoshop scripting on his blog at http://techarttiki.blogspot.com/2008/08/Photoshop-scripting-with-python.html, and there's a great Maxscript introduction at http://lonerobot.net/?p=374. Most of the documentation for the standard JavaScript-based Photoshop tools translates very naturally to any of these options. The only real drawback is the difficulty of communicating from Photoshop back to Max or Maya. In COM terminology, Photoshop acts as a "server" while Max and Maya are the "clients," meaning the communications originate in the 3D tools rather than in Photoshop.

The typical Photoshop pipeline, on the other hand, offers no support for this kind of organization and filtering. Moreover, turning layers on and off is the least of our problems. As shaders get more sophisticated, graphics engineers are constantly looking for ways to pack more data into bitmaps, which means they must do things like decree that artists pack extra information into the top 4 bits of an alpha channel, or create a texture where the red channel is a per-pixel subsurface scattering coefficient, or employ other equally esoteric methods of stuffing more and more data into the same limited amount of GPU memory. It's hard enough for artists to understand many of these requirements at all. Making them pack the textures by hand (or worse, re-pack them when the shader changes) is like adding insult to injury.

TIME TO GRADUATE

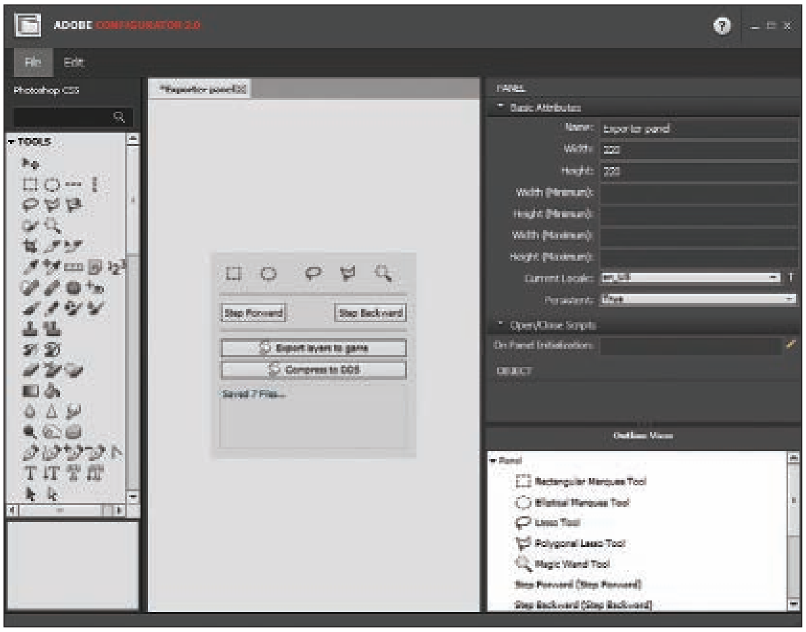
For all these reasons, then, it's high time that studios start looking into extending the same kind of tech-art support to Photoshop users that Max and Maya artists have enjoyed for the last decade. Fortunately, Photoshop does support scripting. Since Photoshop CS debuted in 2003, Photoshop has supported scripting using JavaScript, AppleScript, and VBScript. Although the Photoshop scripting environment will feel constraining to a TA who is used to the power of Python or MaxScript, there's still a lot of power there to streamline artists' workflows and make for cleaner, more reliable data.

Although Photoshop supports three different languages, the most common choice among scripters is JavaScript. As a bonus, it's also the language of Adobe's ExtendScript toolkit, allowing you to write scripts for Illustrator and other Adobe products as like Illustrator (something that may come in especially handy for vector-art oriented developers). JavaScript is both a blessing and a curse. There are more JavaScript tutorials and references available than you can ever use in a lifetime, but they're usually focused on web-specific problems, or bogged down in HTML parsing.

JavaScript looks like a cross between MaxScript and Mel; it has Mel's C-style syntax (with curly braces and semicolons galore) but behaves more like MaxScript. It's a typeless language, meaning you can stick any kind of value into any variable. It's also object oriented, so you can write clean code with nested properties and functions — a huge boon for readability, as you can see in this snippet:

```
// set the current document to 256 x 256 pixels
// where ´app´ is the Photoshop application
app.preferences.rulerUnits = Units.PIXELS;
app.activeDocument.resizeImage (256, 256);
//create and assign variables for document settings
```

Figure 2: Photoshop turns 21 this year. It's still remarkably familiar, as this 1990 screenshot shows (photo credit: www.flickr.com/photos/jcapaldi)



Name: Exporter panel
Width: 220
Height: 220
Width (Minimum):
Height (Minimum):
Width (Maximum):
Height (Maximum):
Current Locale: en_US
Persistent: true
On Panel Initialization:

Figure 3: Adobe Configurator allows you to create custom palettes and dialogs for Photoshop, including scripts and even Flash content.

Another nice feature, shared by both MaxScript and Python, is that JavaScript allows you to put functions themselves (and not just their results) right into variables. This is an elegant mechanism for handling program flow.

ENTRY LEVEL

On the whole, Photoshop scripting is not too intimidating. Unfortunately for the novice scripter, the JavaScript language comes with a few quirks. The most noticeable one is that it's too easy to make your variables into globals, which can lead to you shooting yourself in the foot by innocently reusing a variable name. There are a number of lesser gotchas as well: Much like MaxScript, JavaScript supports multiple alternative ways of specifying things like the beginning and ending of a code block, which can lead to code that's hard to read if you don't develop a style and stick with it aggressively.

It's a bit tricky to find good guidance on the web, since the overwhelming majority of JavaScript info on the internet addresses web developers rather than general-purpose scripting. Douglas Crockford's book *JavaScript: The Good Parts* is a fast way for experienced scripters to spot and avoid the worst pitfalls. Novices, however, will probably find it a bit intimidating. A newbie might want to start with something like David McFarland's *JavaScript: The Missing Manual*, which assumes less programming knowledge.

However you get your scripting chops on, you'll quickly find that Photoshop scripting is a mixed bag. It's possible to do very complex operations—if you really want to, you can paint individual pixels by scripting—but the overall experience is a bit like Maxscript was 10 years ago. You can feel the fact that you're basically automating a lot of UI actions, rather than "programming" Photoshop directly, a fact that's reinforced strongly by the way you can see the UI update for every command you send along. If you're familiar with Photoshop's "Actions" macro system, then you know the feeling well. It should be stressed that, unlike actions, Photoshop scripts are capable of analyzing and reacting to complex data. Nevertheless, that's a small price to pay for a goodly amount of image processing power and workflow cleanup.

RESUME BUILDING

Once you've got some scripts running, you might start wanting to put together some UI for your users. A neat new tool that even a lot of experienced Photoshoppers don't know about debuted in CS4. The "Configurator" utility (available for free download at http://labs.adobe.com/downloads/configurator.html) allows you to create custom toolbars and dialogs, known as "Panels" (see Figure 3). Panels can host buttons for standard Photoshop tools, actions (as you'd usually see in the Actions palette), and most importantly, scripts. This allows you to create custom toolboxes or "shelves" in the familiar Max / Maya mode. While the panels have to be installed directly into your Photoshop plugins directory, which is a hassle for maintenance and versioning, this is a powerful tool for helping streamline Photoshop operations in a user-friendly way.

You can even go beyond that, if you've got a lot of Flash experience. Under the hood, panels are actually Flash SWF files, so they can theoretically contain animations, video, and all sorts of other interactive effects. However, getting beyond the basic button-and-field level of UI requires some serious Flash chops, both for creating the panels and also for communicating between Flash's ActionScript and Photoshop's JavaScript. If you've got experienced Flash developers in-house, this could be a very powerful tool. On the other hand, it may be a bit much for TAs with a background in MaxScript or Python rather than web development.

The biggest weakness of Photoshop scripting is the fact that JavaScript, as a web-oriented language, has few of the tools that Python or MaxScript offer for interacting with the world outside of the application. JavaScript is designed to be a safe client-side web tool that can't, for example, wipe your hard drive at the behest of a malicious web page. This means that vanilla JavaScript can't create or delete folders on your hard drive either, even if you want it to. Likewise, calling external programs (for example, to check a file out of source control or update a game database) is hard to achieve without some clunky workarounds, like saving DOS commands to disk and executing them as a batch file. So, if you do have heavy-duty pipeline needs that can't be met without more horsepower, you should check out the sidebar on integrating Photoshop with Max and Maya. In any case, JavaScripts do provide a lightweight way to take control of your Photoshop workflows and automate boring, repetitive tasks.

THERE IS NOTHING IMPOSSIBLE TO HIM WHO WILL TRY

History relates that Alexander the Great, having smashed the Persian Empire and united the known world before his 33rd birthday, wept because he had no more worlds to conquer. If you're a technically-inclined artist and you've been wondering where your next adventure lies, perhaps you'll find it in that most familiar of all art tools—the one that probably pre-dates your career in games.

STEVE THEODORE *has been pushing pixels for more than a dozen years. His credits include* MECH COMMANDER, HALF-LIFE, TEAM FORTRESS, COUNTER-STRIKE, *and* HALO 3. *He's been a modeler, animator, and technical artist, as well as a frequent speaker at industry conferences. He's currently the technical art director at Seattle's Undead Labs.*

R E F E R E N C E S

HISTORY OF PHOTOSHOP:

www.storyphoto.com/multimedia_photoshop.html
www.webdesignerdepot.com/2010/02/20-years-of-adobe-photoshop/

Hired someone interesting? Let us know at editors@gdmag.com!

dashing to social

EX-BLIZZARD ARTIST MICHAEL DASHOW JOINS KABAM

Michael Dashow worked for seven years at Blizzard, on DIABLO II and III. Several years ago he got an early start in the social/casual industries, working with hi5 Networks, and meez.com. Now he's taken a job as art director for Kabam, and we caught up with him to see how his fantasy art background works in the social space.

BRANDON SHEFFIELD: You have a painterly style, but that sort of talent is not often showcased in social games, though it seems like Kabam is finding ways. How do you reconcile ability versus the often lower artistic demands of the medium?
MICHAEL DASHOW: Social games are in many ways a better place to showcase the abilities of 2D artists than console or PC games: 2D artists get to realize beautiful final art that will actually be seen by our users instead of having their concept art recreated by 3D artists. While many console and PC games strive for a high degree of realism, our medium lets us explore a wide range of visual styles, whether it's painterly, stylized, cartoony, or thoroughly realistic. Rather than make all Kabam games match a singular company aesthetic, I feel that it's really important for each game team to find its own voice. Art leads at Kabam have a lot of latitude to explore what they feel works best for a given game. Producing so many titles as original IP gives us a very open canvas to work on!

BS: I've heard your work will be visible in DIABLO III. How do the art challenges differ from the Blizzard team to the Kabam team?
MD: The tools and engines vary from game to game—DIABLO III is being created with a 3D engine while we are currently using 2D – but the approach to making art for Kabam games is overall pretty similar. Many of our strategy games feature a pulled-back camera view which lets you see the buildings and resource-gathering structures you're constructing, which is pretty much the same view that we used on DIABLO or in other Blizzard strategy games like STARCRAFT. Any of these games require the same sort of artistry and attention to form and silhouette to ensure clarity and attractiveness at that size.

The biggest difference between PC titles and social games is the speed to market. You can spend years on a single title—to your point, I last worked at Blizzard five years ago and they're still working on DIABLO III—but in social gaming you have a much shorter time to get the initial game out. We need short development times because this part of the industry is moving so quickly. In social games we get to keep adding content to a game after it releases, so in that respect, they're much more like working on an MMO.

BS: This is a large question, but what to you represents the "next generation" of social games?
MD: [For us this means] focusing on games with more depth and more authentic gameplay than a lot of what we've seen in the marketplace thus far. We call our new games Massively Multiplayer Social Games (MMSG) because they are both deeper and more social than the vast array of "casual games."

We started by focusing on a different audience—not the casual crowd but core gamers like ourselves—and are building our games to really appeal to this audience. Kabam MMSGs are more like "traditional" strategy games and RPGs, but we've also added a whole social layer which adds connectivity and interaction with friends and other users. This way, we're building deeper, more immersive games that feature synchronous play with and against other players in real time, greater social interaction via alliances, and more engaging, longer-lasting play sessions (as opposed to the brief gameplay of many casual games).

These are still early days for the social gaming industry, and there's lots of room for innovation. Going forward, I think the next advancements will be in the fidelity level of social games. Better art, increasing use of 3D, and even more animation and sound ... we've seen the same advancement in the games industry in the past on PCs and consoles, and now we're seeing it happen again with online and mobile gaming. There's every reason to believe that this trend toward better looking, more in-depth, and more fun games will play out in social gaming as well. As a long-time industry vet, it's exciting to be starting over with this new segment of gaming, and it's really rewarding to be part of a studio that's helping push growth in social gaming forward.

new studios

¤ German company Idea Fabrik quietly purchased the HeroEngine development platform and technology from Simutronics late in 2010, and now the company has founded a new game development studio in northern Virginia called Second Star Interactive.

¤ While Bethesda Softworks sibling ZeniMax Online Studios has yet to reveal any of its MMORPGs, it is already making preparations for future launches with its new customer support facility in Galway, Ireland.

¤ RATCHET & CLANK and RESISTANCE developer Insomniac Games has revealed that it is looking to enter the web and mobile social gaming space with its newly formed division Insomniac Click.

who went where

¤ After 20 years with UK-based WORMS and ALIEN BREED developer Team17, studio head and co-founder Martyn Brown is leaving the company to become an independent game consultant.

¤ Nintendo of America announced it has hired Cynthia Gordon, a public relations veteran with over 20 years of experience in consumer brands, as its new vice president of corporate affairs.

¤ Following his departure as CEO from Atari's Cryptic Studios, John Needham has taken the chief executive position at another online gaming firm, Gazillion Entertainment.

¤ PC game designer Doug Church, whose credits include titles in the ULTIMA, THIEF, and SYSTEM SHOCK series, has taken a position with Seattle-based HALF-LIFE and PORTAL developer Valve.

GDC 2011 CONFIRMS RECORD ATTENDANCE, GDC 2012 DATES ANNOUNCED

HIGHLIGHTS AND 2012 DATES REVEALED

\\\ Organizers of the Game Developers Conference announced that the 25th edition of the event hosted a record 19,000 game industry professionals at San Francisco's Moscone Convention Center.

The weeklong anniversary event offered more than 450 lectures, panels, summits, tutorials, and roundtable discussions across a full five days of content, with GDC references trending on Twitter in San Francisco. It also saw unprecedented media coverage from outlets like the Los Angeles Times, the Associated Press, USA Today, and beyond, with GDC sister site Gamasutra including full coverage on its site.

Lecture highlights from Monday and Tuesday's GDC: includes more than 15 tutorials and summits with Rovio's Peter Vesterbacka discussing the ANGRY BIRDS phenomenon, Zynga's Mark Skaggs on going from FARMVILLE to CITYVILLE, game designer and author Jane McGonigal on "gamefulness," and SUPER MEAT BOY's creators on their rough route to success.

GDC 2011 also played host to the 13th Annual Independent Games Festival and the 11th Annual Game Developers Choice Awards. Swedish independent developer Mojang's acclaimed 3D world-building sandbox title, MINECRAFT, won the Seumas McNally Grand Prize and Audience Award during the IGF, as well as three awards at the GDCAs, becoming the first title ever to win awards in both ceremonies in the same year.

Some other highlights from this year's GDC included an all-star line-up of "classic postmortems," discussing the making of some of the most seminal video games of all time. Eric Chahi's talk on OUT OF THIS WORLD/ANOTHER WORLD received a standing ovation for its inspirational story. Other postmortems included Mark Cerny on MARBLE MADNESS, John Romero and Tom Hall discussing DOOM, and Will Wright on RAID ON BUNGELING BAY, featuring bonus "Russian Space Minute" diversions.

Other major talking points at the show were a keynote from Satoru Iwata, President of Nintendo, titled "Video Games Turn 25: A Historical Perspective and Vision for the Future," a social game-centric Rant, and the Game Design Challenge won by Jason Rohrer with an innovative MINECRAFT mod.

GDC 2011

Standout talks from STARCRAFT II's Dustin Browder, LucasArts' Clint Hocking, and key lectures from GDC veterans Chris Crawford on "days of yore" and Brian Moriarty on Roger Ebert and "sublime art" were also among the most buzzed-about talks at the show this year.

Finally, a packed GDC Expo floor included multiple demo units for Nintendo's 3DS handheld, major showcases for game tools companies, spectacular 3D stereoscopic gaming showcases, busy Career Pavilion and Business Center areas, as well as the always popular IGF Pavilion.

"The week of GDC truly embodied the passion and spirit of the video game community," said event director Meggan Scavio. "From seasoned game veterans to aspiring game professionals in areas spanning social and online games through major console titles and beyond, we are honored to continue to serve the industry—and hope to see you all next year."

Following the success of the show, Game Developers Conference organizers have announced that GDC 2012 will return to the Moscone Convention Center in San Francisco from Monday, March 5 to Friday, March 9, 2012, with a call for lecture submissions to open this summer.

GDC EUROPE 2011 DEBUTS NEW SUMMITS

TEAM PLANS TO FOCUS ON EMERGING MARKETS

\\\ Organizers of GDC Europe 2011 have revealed a host of focused Summits for this August's conference in Cologne, Germany.

Taking place Monday through Wednesday, August 15–17, 2011, at the Cologne Congress-Centrum Ost, GDC Europe 2011 will again provide the essential pan-European perspective of game development and business trends happening throughout the continent today.

Now in its third iteration and taking place alongside the European-leading Gamescom trade fair, GDC Europe will continue to serve the European game industry by gathering the world's leading speakers in areas specific to current game development across platforms and development disciplines.

GDC Europe 2011 plans to expand the breadth of its conference by offering four Summits to be held concurrently with the main conference.

Reflective of the growing development and activity within the continent, GDC Europe will introduce the Social Games Summit, Smartphone & Tablet Games Summit, Independent Games Summit, and Community Management Summit to its roster this year.

"It is our goal to support growth of the European games community by providing a venue for collaboration and business opportunity, as well as a valuable learning experience featuring leading industry tools and techniques," said Frank Sliwka, event director of GDC Europe.

"This year, the addition of Summits will provide strategic, dedicated content focused on emerging and innovation-driven topics in the industry. We are pleased GDC Europe is returning to Cologne, and are proud to continue providing the forum for innovation and networking for the European game development community."

In 2011, in conjunction with its new conference content lineup, GDC Europe will offer even more business and networking opportunities for attendees, with a developer-focused Expo Floor area, a host of day and evening networking events, and the return of the GDC Europe Business Lounge within Gamescom itself.

OCTODAD

www.octodadgame.com

IGF STUDENT SHOWCASE FINALIST OCTODAD PUTS PLAYERS IN CONTROL OF A CLUMSY INVERTEBRATE MASQUERADING AS THE FATHER OF AN UNASSUMING SUBURBAN FAMILY. WE SPOKE WITH THE GAME'S LARGE TEAM FROM DEPAUL UNIVERSITY TO FIND OUT HOW THIS SURREAL ADVENTURE TITLE CAME TOGETHER.

Tom Curtis: The most immediately striking aspect of OCTODAD is its ridiculous premise. How did you settle on such a silly concept?

John Murphy (Executive Producer): We spent a few weeks pitching ideas and got to a point where we almost decided to go with one of a couple of "platformer with a physics-based twist" ideas. We did one more round of brainstorming in small groups to either flesh out earlier pitches or to come up with something totally new. Nick, our sound designer Seth, and I were becoming frustrated with our inability to make these platformer ideas interesting enough. We started throwing around joke ideas. I said, "How about a person driving a person with horribly complicated controls?" Then Nick responded, "What if it were an OCTOPUS driving a person?" We pulled up a YouTube video of 'Jurassic Park: Trespasser,' which we thought was unintentionally hilarious but also strangely compelling. So we brought this ridiculous idea of an octopus driving a person back to the team, and it eventually morphed into the player controlling an octopus with a human family.

TC: How was the decision made to emphasize unwieldy controls? Did the gameplay arise from the appealing premise?

Jake Anderson (Lead Designer): The concept and the game controls really went hand-in-tentacle with one another. Our goal was set for us when John gave his live demo during the pitch, flailing his arms and acting out how OCTODAD might stumble around. The comedy in the concept was what really drove the game forward. We started development with that target, and recognized that in order to create the slapstick destruction we wanted we could limit the control the player had over the character by giving them the ability to micromanage tentacles. Balancing the frustration and the accessibility became a task that we tweaked throughout development.

TC: Any favorite features that were left on the cutting room floor during development?

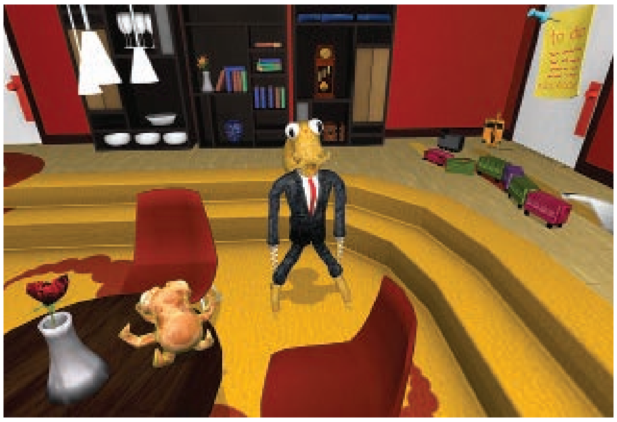
Kevin Zuhn (Project Lead, Writer): We had some ideas for more octopus-themed mechanics: For instance, OCTODAD's suspicion gauge was not only how much his family was suspecting him, but how much ink he had built up in his body due to nervousness. When the bar was high, he could either subtly dispose of his ink to reduce the gauge, or go crazy and spray ink all over the room. We also discussed having OCTODAD change his skin color to show his mood or to camouflage himself.

Nick Esparza (Lead Artist): One early idea was for OCTODAD to have a tape recorder under his suit, so that part of fitting in was to use a few pre-recorded phrases to converse with people. I think having to communicate with people while only using phrases like "Hello," "Thank you" and "It's four o'clock!" would have been hugely funny.

JA: Our early prototypes included banana peels and other "slippable objects" which would send OCTODAD flying and landing in a heap on the ground. We included this in our GDC Kinect demo, so I'm pretty excited to see these making their way back into the game.

TC: How long did it take to settle in on the final control method? Did you look at input methods other than the mouse?

KZ: We spent a few weeks prototyping different control schemes for OCTODAD's movement, and that was after coming up with an exhaustive list of possible ways to represent his uncontrollable body. We tried out using WASD and arrow keys to keep his body from falling over as you walked. Our very first tentacle prototype was built for the 360 controller, using the right trigger to lift and the joystick to navigate. At one point we were considering more than one mouse! We also recently implemented Kinect controls that track your arm movements.



JA: Even after developing the initial system, the control and input mechanics were not finalized. We were constantly trying to make improvements throughout production by adding features such as the assisted grab, or setting the body on a pivot to allow for greater reach distance.

TC: How did you ensure that the game's loose and unpredictable controls didn't frustrate players?

Majdi Badri (Designer): A lot of playtesting and level design. We learned early that players did not like to climb up stairs with OCTODAD's wobbly legs. This led us to put only one pair of stairs in the game, and save another for our "unique" final boss. We didn't want to keep the player in one mode for too long, either. You can only jam your tentacle around so much in a kitchen before you want to move on. We kind of developed a rhythm of foot, hand, foot. As I said before, playtesting was also very important. In some of OCTODAD's early iterations, we had him fall over if he stretched his legs out too far. Needless to say, players became very annoyed when OCTODAD tumbled over, making walking around even more difficult. So that was nixed and we brought in the banana peels, which were similar, but kept with the game's humor and made more sense to players.

TC: You have quite a big team for a student game. How did you all end up working together?

JM: A couple of DePaul professors interviewed about 50 students to pick out a team of 20 to create a game to win the IGF Student Showcase. It was a six-month extracurricular project starting in June. While we didn't decide to work together, Patrick Curry and Scott Roberts, our advisers, did an incredible job of picking a team full of people with complementary personalities, so it pretty much felt like we all had come together on our own.

—Tom Curtis

INVEST IN YOURSELF.
BECOME A MASTER OF DIGITAL MEDIA.

We offer a 20-month Master's program in entertainment technology and digital media that combines industry-facing curriculum, real-world projects and a four-month internship. The MDM degree is jointly awarded by four leading Canadian post-secondary institutions: The University of British Columbia, Simon Fraser University, Emily Carr University of Art + Design, and the British Columbia Institute of Technology.

Our students come from around the globe, and from diverse undergraduate and professional backgrounds – including media and fine arts, computing science, natural and social sciences, business, and engineering.

Come work and play with us in Vancouver BC: Canada's video game capital.
For more information about the Masters of Digital Media (MDM) Program, go to mdm.gnwc.ca

CENTRE FOR DIGITAL MEDIA
Great Northern Way Campus

UBC SFU emily carr university of art + design BCIT

TURN YOUR PASSION FOR GAMING INTO A CAREER

Game Art
Bachelor's Degree Program
Campus & Online

Game Design
Master's Degree Program
Campus

Game Development
Bachelor's Degree Program
Campus

Game Design
Bachelor's Degree Program
Online

Campus Degrees

Master's
Entertainment Business
Game Design

Bachelor's
Computer Animation
Creative Writing for Entertainment
Digital Arts & Design
Entertainment Business
Film
Game Art
Game Development
Music Business
Recording Arts
Show Production
Sports Marketing & Media
Web Design & Development

Associate's
Graphic Design
Recording Engineering

Online Degrees

Master's
Creative Writing
Education Media Design & Technology
Entertainment Business
Internet Marketing
Media Design
New Media Journalism

Bachelor's
Computer Animation
Creative Writing for Entertainment
Digital Cinematography
Entertainment Business
Game Art
Game Design
Graphic Design
Internet Marketing
Mobile Development
Music Business
Music Production
Sports Marketing & Media
Web Design & Development

FULL SAIL UNIVERSITY

fullsail.edu

Winter Park, FL

800.226.7625 • 3300 University Boulevard

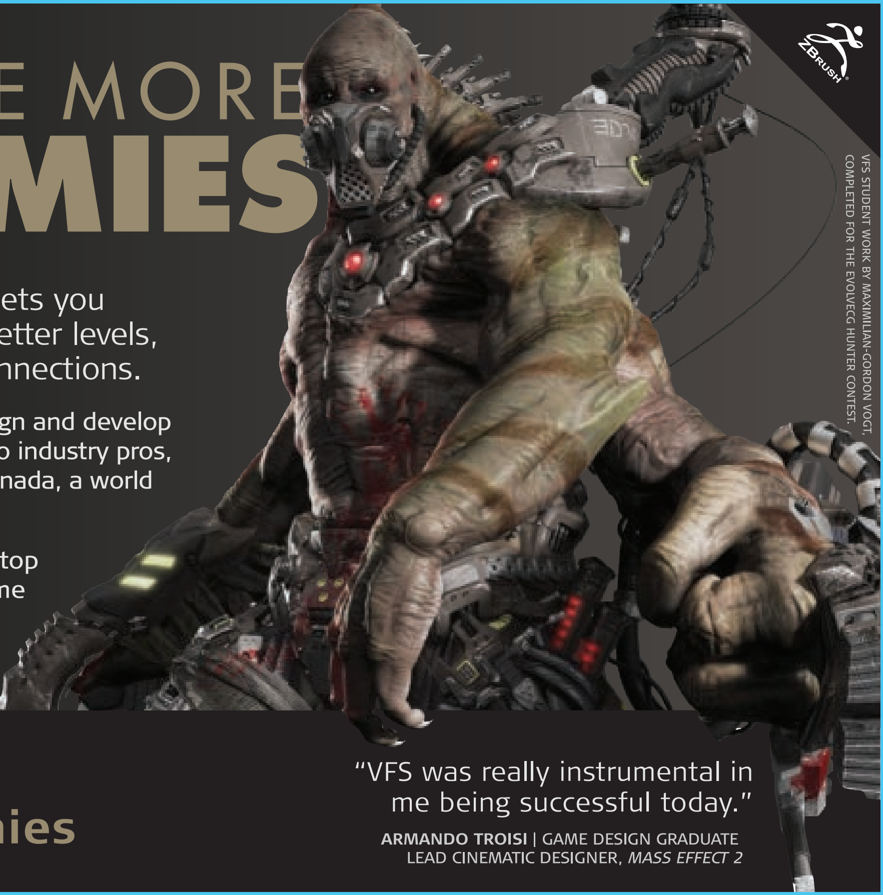
Financial aid available to those who qualify • Career development assistance
Accredited University, ACCSC

MAKE MORE ENEMIES

Game Design at VFS lets you make more enemies, better levels, and tighter industry connections.

In one intense year, you design and develop great games, present them to industry pros, and do it all in Vancouver, Canada, a world hub of game development.

The LA Times named VFS a top school most favored by game industry recruiters.


ZBrush
VFS STUDENT WORK BY MAXIMILIAN-GORDON VOGT, COMPLETED FOR THE EVOLVECG HUNTER CONTEST.

VFS

Find out more.
vfs.com/enemies

"VFS was really instrumental in me being successful today."

ARMANDO TROISI | GAME DESIGN GRADUATE
LEAD CINEMATIC DESIGNER, *MASS EFFECT 2*

GET EDUCATED

ADVERTISER INDEX

COMPANY NAME	PAGE	COMPANY NAME	PAGE
Devry University	13	Masters of Digital Media	54
Epic Games	14	NaturalMotion	C3
Full Sail Real World Education	54	Perforce Software	6
IDG World Expo	40	Rad Game Tools	C4
Geometrics LTD	11	Siggraph	29
Havok	C2	Sony Computer	34
iBeta Quality Assurance	17	TwoFour54	3
IDG World Expo	30	Vancouver Film School	55

gd Game Developer (ISSN 1073-922X) is published monthly by United Business Media LLC, 303 Second Street, Suite 900 South, South Tower, San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address. Canadian Registered for GST as United Business Media LLC, GST No. R13288078, Customer No. 2116057, Agreement No. 40011901. **Subscription Rates:** Subscription rate for the U.S. is $49.95 for twelve issues. Countries outside the U.S. must be prepaid in U.S. funds drawn on a U.S. bank or via credit card. Canada/Mexico: $69.95; all other countries: $99.95 (issues shipped via air delivery). Periodical postage paid at San Francisco, CA and additional mailing offices. **Postmaster:** Send address changes to Game Developer, P.O. Box 1274, Skokie, IL 60076-8274. **Customer Service:** For subscription orders and changes of address, call toll-free in the U.S. (800) 250-2429 or fax (847) 647-5972. All other countries call (1) (847) 647-5928 or fax (1) (847) 647-5972. Send payments to *gd Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. Call toll-free in the U.S./Canada (800) 444-4881 or fax (785) 838-7566. All other countries call (1) (785) 841-1631 or fax (1) (785) 841-2624. Please remember to indicate *gd Game Developer* on any correspondence. All content, copyright *gd Game Developer* magazine/United Business Media LLC, unless otherwise indicated. Don't steal any of it.

GAMIFIED!

GAMIFICATION COMES TO ARRESTED DEVELOPMENT

DEAR READERS,
Welcome to the new gamification column here in *Game Developer* magazine! Just by reading that last sentence, you've already earned a "Novice Reader" badge for your profile and 100 Magic Beenz™! Congratulations! You can spend your Magic Beenz™ inside the Magic Beenz Store™ to unlock different themes and fonts to customize your experience with this column!

Now that you're already on board, let's begin this column about the art and science of gamification! While you're reading, think about how many of your friends would enjoy reading this, too! Send them a note to check it out, and you'll earn 300 Magic Beenz™! Or, if they don't have a subscription to *Game Developer* magazine, sign them up and you'll earn 1,000 Magic Beenz™ while they earn a bonus 500 Magic Beenz™!

Notice how engrossed you are in this reading experience! In today's "attention economy," where sources of distraction are seemingly infinite, my ability to keep your attention focused on this page and reading this article is worth literally billions of dollars! Gamification takes the principles of game design and behavioral economics and combines them into concrete, actionable tactics that increase user engagement! You're now 25% through the column! Good work! You've earned the "Experienced Reader" badge for your profile!

[Hey, we just noticed your friends are also reading this column this very second, and they're all ahead of you! Don't let them beat you to finishing this column! Keep reading!]

The natural desire of all human beings is to be at play for 100% of the time. So why should we relegate our time playing only to specific periods? In the future, game mechanics will be tied to all important areas of life. We'll shed pounds and tone our abs by playing WORLD OF WARCRAFT! We'll encourage healthy debate on public policy through ANGRY BIRDS! Instead of being asked to pay your insurance bill, you'll be sent on a *quest* to pay your insurance bill!

Now you're almost 50% through the column! Just a little more and you'll get to the halfway point! When you do, you'll earn the "Veteran Reader" badge on your profile! You will also level up and unlock the last half of this column!

Ding! You did it! You can now read the rest of the column!

It's crazy, but there are some people who criticize "gamification." Now why would anybody want to do that? There's no Magic Beenz™ to be earned by making a sad face about the exciting world of gamification! Let's take a look at their criticisms anyway! We can do it just for fun! Fun is what gamification is all about!

"Gamification is actually just a new term for old sales techniques, such as loyalty programs."
[Press here to "dislike" this criticism of gamification!]

WRONG! Gamification is entirely new! There was no conception of the principles of gamification before Dr. Frédéric Gamific, the inventor of gamification, was struck with a vision after collapsing on his hedonic treadmill one evening! We all know this story! Don't you?

"Gamification fundamentally misinterprets why people do things."
[Press here to "dislike" this criticism of gamification!]

I've read about this one! It's got something to do with "intrinsic" versus "extrinsic" motivation, doesn't it? Well, what if I were to tell you that there were over TEN THOUSAND Magic Beenz™ in it for you to hit that "dislike" button right up there? That makes the issue go away real fast, doesn't it? Look at you go!

The beans! They tempt you!

ILLUSTRATION BY JACEK WOO

"Gamification is not real game design."
[Press here to "dislike" this criticism of gamification!]

Oh, now what? Did the International Congress of Game Design decide what's game design and what isn't? Well excuse me, but I'm increasing customer loyalty and boosting ROI here! If that isn't "game design," I don't know what is! Keep this up and I'm going to ban you!

And there we go! I hope that answers some of the critics of gamification! Honestly, though, I think those sad sacks are just jealous that they aren't having a mega-fun gamified experience like you are! Which reminds me, enable auto-Tweeting about your badges and you'll earn an additional 4,500 Magic Beenz™ per tweet! Or you could share it on your Facebook page! Your friends are all going to want to know about your accomplishments!

[Hey, we just noticed that all of your friends liked this column so much that they're already planning on reading next month's column when it comes out! You should probably join them and do the same! Reserve your copy of next month's *Game Developer* magazine right now and get an additional 50,000 Magic Beenz™!]

You're now 100% through the column! Excellent work! You've earned the "Amazing Person," "Genius Reader," and "Gamification Expert" badges for your profile! Here's 1,000,000 Magic Beenz™!

MATTHEW WASTELAND *writes about games and game development at his blog, Magical Wasteland (www.magicalwasteland.com). Email him at mwasteland@gdmag.com.*

IGNITION ENTERTAINMENT

USES MORPHEME

"The high quality, fluid performances seen in El Shaddai are delivered using Morpheme."

Sawaki Takeyasu, Director of El Shaddai at Tokyo's UTV Ignition studio

UTV IGNITION entertainment

El Shaddai

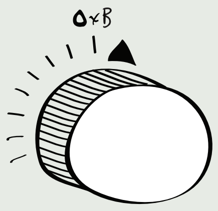
morpheme advanced animation system

naturalmotion

Image courtesy of Ignition Entertainment

www.naturalmotion.com

TURN UP THE

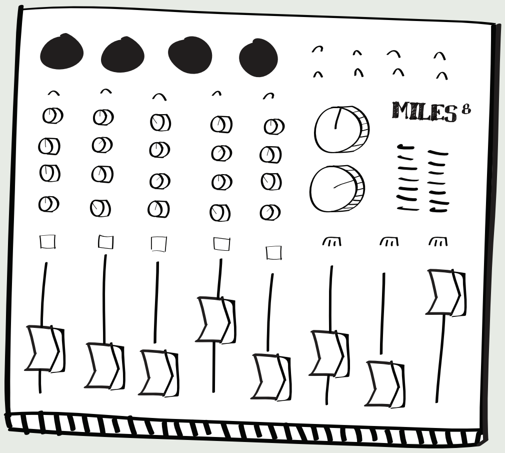


WICKED

AUDIO IN YOUR GAME WITH

Miles 8

BRAND NEW!



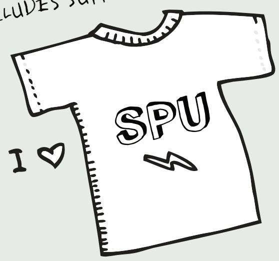
Our multichannel sound system features cool, new

high-level audio tools

+ the world's *FASTEST*

MP3 and Ogg **DECODERS.**

INCLUDES SUPPORT FOR THE PS3



USING MILES 8 NOT ONLY MAKES YOU WANT TO TURN IT UP, IT MAKES YOU rad!

RAD© GAME TOOLS

www.radgametools.com
(425) 893-4300