INTERVIEW WITH NANAON-SHA'S MASAYA MATSUURA

JANUARY 2013
VOLUME 20 NUMBER 01

gd

GAME DEVELOPER MAGAZINE

postmortem PAPO & YO

FRONT LINE AWARDS
BEST TOOLS OF 2012

UBM Tech

Ninja Theory and Capcom Reboot Devil May Cry *with Unreal Engine 3*

Dante is getting a makeover. The mercenary demon hunter is back in action in *DmC Devil May Cry*, a highly anticipated reboot of the franchise.

By working closely with Capcom, developer Ninja Theory stripped Dante to the bare essentials to incorporate the story of his origin into this new action adventure. The UK studio utilized Unreal Engine 3 (UE3) to bring this multi-world story to life.

Previously, Ninja Theory utilized UE3 technology to ship Namco Bandai's *Enslaved: Odyssey to the West*.

"We've worked with the Unreal Engine for a long time now, and we found that the engine would enable us to achieve what we wanted with *DmC*, whilst at the same time allowing us to get involved in making the game straight away," said Dominic Matthews of Ninja Theory. "A new engine would have meant at least six months' worth of learning before we could get started with the actual development."

Matthews continued, "We've learned what we can do with the engine and how we can modify it to our needs. Everything that we learned during *Enslaved* development we've been able to take into our work on *DmC*."

Alex Jones, producer at Capcom, said UE3 allowed the team to create larger open areas than they'd done before, moving away from corridors, and gave them more flexibility in opening up the gameplay.

"Unreal Engine scripting is very flexible, so some of our game events are a little more involved than in previous *Devil May Cry* games," said Jones.

Matthews said Ninja Theory used pretty much all of the engine's features at one point or another during the game's development.

"One really useful aspect for us was Unreal Matinee, as this allowed us to easily set up complex cutscenes and really get into the minutiae of achieving the filmic look that we're after," said Matthews. "On a related front, the material system gives us complete control over the look and feel of every surface – early on we made the decision to allow our artists to create their own materials and this really helped us achieve the distinct atmosphere of the game."

Ninja Theory has been at the forefront of performance capture, having worked with actor and director Andy Serkis on *Heavenly Sword* and *Enslaved*. In *DmC*, Dante and the other characters are being brought to life using the studio's latest performance capture technology, which has excelled at pushing things forward over its past story-driven action games.

"We've developed an in-house facial motion capture solver that we used for all of the cinematic scenes in *DmC Devil May Cry*," said Matthews. "This allows us to deliver top quality results without having to pay external companies to wrangle content for us."

Performance capture plays an important role in bringing a Hollywood cinematic feeling to the game. Early on, Capcom told Ninja Theory to think of Dante and *DmC* as a contemporary movie, as one goal is to introduce a fresh take on *Devil May Cry* to a wider audience, while at the same time preserving and building on the DNA of the series.

"This idea of creating a Dante as if he were in a modern day movie has guided us through development," said Matthews. "We're very happy with where we've ended up and I hope that those new to the series and existing *Devil May Cry* fans will find a lot of fun in the game."

In order to improve the overall game experience, Ninja Theory built new technology on top of UE3 to customize lighting and shadowing and to create faster and more accurate cloth simulations and faster particle systems. They also worked with Epic Games and other UE3 teams on the Unreal Developer Network (UDN) throughout the process.

"UDN is great for quickly looking up documentation, but it's the community forums that are invaluable," said Matthews. "All UE3 licensed developers have access to the forums so there's a lot of experience to draw upon, which always helps when there is a tricky issue to solve."

"Having an engine capable of cross-platform development from the start provides a huge advantage," said Matthews. "The platform flexibility that UE3 demonstrates was a key factor in allowing us to concentrate on the game itself. Rather than having to designate one console as a lead platform we were able to treat them as equals. There are, of course, systems that work differently between platforms but, thanks to the engine, these are the exception rather than the rule."

By allowing UE3 to do a lot of the heavy lifting, Ninja Theory was able to focus its efforts on creating a fresh approach to a beloved video game protagonist. *DmC* pushes the gameplay forward by giving players more choices, while offering a cinematic experience that's sure to attract a wide audience.

Thanks to Ninja Theory for speaking with freelance reporter John Gaudiosi for this story.

UPCOMING EPIC ATTENDED EVENTS

D.I.C.E Summit
Las Vegas, NV
February 5-8, 2013

Cloud Gaming Europe
London, UK
February 21-22, 2013

Please email licensing@epicgames.com for appointments

gd

GAME DEVELOPER MAGAZINE

toc 001

CONTENTS_January 2013

VOLUME 20 NUMBER 01

Postmortem

Features

Departments

game developer magazine

Our New Year's Resolutions

GAME DEVELOPER'S SELF-IMPROVEMENT GOALS FOR 2013

Every year I make a few New Year's Resolutions; last year I resolved to wake up by 8 a.m. every day and start flossing my teeth. (Pearl, our publisher, will be more than happy to tell you how the first one worked out.) But I'm going to walk you through a few planned improvements we're making to *Game Developer* in 2013, in the hope that you, dear readers, will keep me a bit more accountable than my dentist does.

NEW LOOK, SAME BOOK First: Welcome to the new *Game Developer* magazine! You've already noticed the first big step we've taken: *Game Developer* has a new look! Magazine redesigns are usually a big, time-consuming endeavor that takes several months of mock-ups and approvals, but we didn't have time for that, so we just told Art Director Joseph Mitch to redesign the magazine and he said "Okay."

We were all sad to see the ripped-from-*Thrasher* grungy look go, but this new, clean look should be a little bit easier on your eyes when you're digging into our latest postmortem (especially if you're reading on a tablet). I'm excited to see what you all think about it, so tweet your feedback to me at @gamedevmag.

In addition to the new design, we've also upgraded our magazine's binding from the saddle-stitched binding to what's called perfect binding, which is mostly neat because now we get a spiffy book spine (handy for those of you collecting the print versions). It's the small things, you know?

IMPROVED DIGITAL VERSION Our digital edition has come a long way since its release about a year ago, and the iOS app in particular has seen a few major under-the-hood improvements; not only has the resolution been bumped up to make the magazine more readable (especially on Retina displays), but there is also a new just text-and-images reader mode (similar to Instapaper or Readability) that makes it possible to actually read the articles on a smartphone or other smaller-than-a-tablet device.

And yes, I say "smartphone or other device" instead of "iPhone" because our Android app is finally coming along! We're still trying to figure out our specific launch plans (and, in fact, it might be out by the time you read this), but rest assured that all of you who told me on Twitter that you wouldn't be resubscribing until we were on Android can rejoin the flock very soon (that means you, @SnakeEaterITA).

GAMADEVELOPER MAGASUTRA This year, we're going to try to play more nicely with our sister site Gamasutra. That means you'll start seeing the occasional Gamasutra feature show up in GD Mag (in our new section called Cross-Platform), and vice versa. We love doing print/digital magazines, but we want to make some of our excellent content freely available for a wider audience to read and discuss, and for Gamasutra, it'll be a neat opportunity to give some love to some of their longer-form articles that might get missed in your daily RSS sweep. (Fun fact: The Google description for Gamasutra calls it the "free online version of *Game Developer* magazine," so tease Gamasutra Editor-in-Chief Kris Graft about that when you see him at GDC 2013.)

2012 was a big year for us and we're looking forward to making 2013 even better. Hope you enjoy the magazine!

—Patrick Miller
@pattheflip

Corrections

- From the November 2012 issue's Power 50 feature: We incorrectly credited "Dan Miller" from The Blast Furnace for his work on PITFALL! for iOS; the correct name is Dan Roberts. Also, the photo published for Jon Mak from Queasy Games was the wrong Jon Mak. Sorry about that. (Fortunately, we ran a whole bunch of photos of the real Jon Mak in the SOUND SHAPES postmortem in December 2012, so you should have a pretty good idea of what he looks like by now.)
- From the December 2012 issue's Year in Review feature: The photo next to Mare Sheppard's entry was not actually Mare Sheppard. This is Mare Sheppard right here. We regret the error.

Mare Sheppard.

GAME DEVELOPER MAGAZINE
WWW.GDMAG.COM

UBM LLC.
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
gamedeveloper@halldata.com
www.gdmag.com/contactus

EDITORIAL

PUBLISHER
Simon Carless - *scarless@gdmag.com*
Pearl Verzosa - *pearl.verzosa@ubm.com*
EDITOR
Patrick Miller - *pmiller@gdmag.com*
EDITOR EMERITUS
Brandon Sheffield - *bsheffield@gdmag.com*
MANAGER, PRODUCTION
Dan Mallory - *dmallory@gdmag.com*
ART DIRECTOR
Joseph Mitch - *jmitch@gdmag.com*
CONTRIBUTING WRITERS
Chris Edwards, Deborah Chantson, Noel Llopis, Jon Beilin, Steve Theodore, Damion Schubert, Damian Kastbauer, David Edery, Alexandra Hall, Matthew Wasteland, Magnus Underland
ADVISORY BOARD
Mick West Independent
Brad Bulkley Microsoft
Clinton Keith Independent
Bijan Forutanpour Sony Online Entertainment
Mark DeLoura Independent
Carey Chico Independent
Mike Acton Insomniac

ADVERTISING SALES

VICE PRESIDENT, SALES
Aaron Murawski - *aaron.murawski@ubm.com*
t: 415.947.6227
MEDIA ACCOUNT MANAGER
Jennifer Sulik - *jennifer.sulik@ubm.com*
t: 415.947.6227
GLOBAL ACCOUNT MANAGER, RECRUITMENT
Gina Gross - *gina.gross@ubm.com*
t: 415.947.6241
GLOBAL ACCOUNT MANAGER, EDUCATION
Rafael Vallin - *rafael.vallin@ubm.com*
t: 415.947.6223

ADVERTISING PRODUCTION

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

REPRINTS

WRIGHT'S MEDIA
Jason Pampell - *jpampell@wrightsmedia.com*
t: 877-652-5295

AUDIENCE DEVELOPMENT

AUDIENCE DEVELOPMENT MANAGER
Nancy Grant e: *nancy.grant@ubm.com*
LIST RENTAL
Peter Candito
Specialist Marketing Services
t: 631-787-3008 x 3020
petercan@SMS-Inc.com
ubm.sms-inc.com

UBM Tech
WWW.UBM.COM

ACE ATTORNEY

NEW MEDIA RIGHTS GIVES GAME DEVS FREE LEGAL ADVICE

WHAT CAN A SMALL DEV STUDIO DO TO STAY ON THE RIGHT SIDE OF THE LAW WITHOUT BREAKING THE BANK? WE TALKED TO SHAUN SPALDING, ASSISTANT DIRECTOR OF LEGAL AID NONPROFIT NEW MEDIA RIGHTS (NEWMEDIARIGHTS.ORG), ABOUT STICKING UP FOR THE LITTLE GUY.

THE DARK MOD.

PATRICK MILLER: ***What is New Media Rights?***
SHAUN SPALDING: New Media Rights (NMR) is a nonprofit program of California Western School of Law that provides free legal assistance to independent creative people and average Internet users. If you are starting a creative project and need help preventing issues before they show up, or if you get a letter from someone using their legal weight to bully you, you can come to us.

The same type of assistance that costs $300 an hour from a private lawyer, we give away for free. We do this because the Internet has created an era where individuals with no money and a lot of talent can create work that can compete with large media companies, but those companies have huge legal departments, and the only resource we normal people have to figure out what we can and can't do legally is Google.

We also do educational and policy work; we create guides, videos, and speak in the community, and we do things like change regulations at the FCC and Copyright Office through regulatory comments. For example, earlier this year, we were one of the groups that helped ensure that jailbreaking your phone is still legal. We also told the FCC that the government shouldn't be able to shut down your phone service on public transportation.

PM: ***The Legal Assistance for Game Developers series of videos—what inspired them? Who uses them?***
SS: Our goal with our educational work like LAGD is to make sure that the people we help *don't* need lawyers. LAGD is just an extension of that. After a string of game developers came to us in February, we realized there was no general information we could point them to at all. We were writing all of this general information ourselves, *then* having to do the specific, technical work afterward, too. That's when we decided that we should release all the general information out there in a format that people would want to engage in.

Since we already had text, we could post it on our website like we do a lot of our other guides, but a lot of people don't want to just read something. Creative people are visual and auditory, so we're trying to get this information to people any way they want to accept it, and audio/video in addition to text seems like the way to go. Some of our topics include cloned games (what can and can't be cloned, and how to react if your game is cloned), using real people in your games, working with business partners, and making fan films.

PM: ***Got any success stories so far?***
SS: Since we're attorneys, our biggest public relations problem is that we're under confidentiality obligations. Of what we can talk about, we've assisted with games like THE DARK MOD to ensure they could legally release a standalone version of their mod. We've defended the work of feminist media critics like Anita Sarkeesian of Feminist Frequency, and Jonathan McIntosh, the creator of the well-known Buffy vs. Edward mash-up. We've fought and won against an organization that was improperly threatening the mobile game WORDSMITH.

PM: What should devs know about before starting their own studio?
SS: First off: Our three-part trademark episode of LAGD talks about the process of picking a name for your studio and game. Trademark issues, such as calling your game *Super Monopoly* when your game has nothing to do with the company that makes *Monopoly*, are by far the easiest way to get your game taken down and have a lot of trouble retooling your game later on.

The more general number two is: Always try to figure out what you don't know. Try to read up on a bit of everything when it comes to business or legal issues, so you can realize when you're out of your element and you need help. The best thing you can do is realize situations where there could be a problem that you may need to fix, because then you can either do enough research to fix it, or better anticipate what happens when things actually do go wrong.

PM: ***You mentioned cloning earlier. How can devs protect themselves from other devs cloning their games?***
SS: We're actually putting out a whole huge 10–20 page guide about cloned games, but that subject is way too huge to talk about in this interview with specific tips. Long story short: Cloning games is mostly legal, but in some situations it's not. Not to do too much self-promotion, but the best way for indies to protect themselves is to read our guide that we'll be publishing, and guides like ours, to inform themselves. —*Patrick Miller*

THE TWINE REVOLUTION

NEW CREATORS FIND THEIR VOICES THROUGH HYPERTEXT

In an era of tumultuous game industry change, one of the most notable shifts comes in the increasing visibility of players and developers, be they female, queer, or otherwise marked "different," who fall outside of the game industry's focus on the 18–34 male demographic. One such episode is playing out in the interactive fiction (IF) community, where a diverse group of "othered" creators is enthusiastically adopting a hypertext creation tool called Twine (gimcrackd.com/etc/src/).

Twine has existed since at least 2009, but it has only really risen to prominence in the last year or so. Chris Klimas, Twine's original creator, is as surprised as anyone. "I really thought Twine was a dead project as recently as a year ago," said Klimas. "It's been kind of astonishing to see that, first of all, people are using it, and secondly, the directions they're taking it in. I have to credit Anna Anthropy for advocating quite effectively for Twine. I think she's the single biggest reason why people are taking it up."

Porpentine, a queer woman who recently hosted a Twine game jam, is another strong advocate for the software. "Twine is accessible on a level unseen since HyperCard, scales with existing languages (HTML/CSS/JavaScript), and has great visual feedback," she said. "Twine is one of the few applications I can think of that actually feels good to a non-programmer artist—like a table full of notecards that can be connected."

Twine's similarity to HyperCard is no accident. "I wanted Twine to be like HyperCard," said Klimas. "The concept I had in mind was that you would be able to get pretty far without writing any code, but you could also dip your toes into that water when you were ready. I wanted to make it for writers, not programmers."

Anthropy, the outspoken developer and critic whose book *Rise of the Videogame Zinesters* is a sort of DIY game-creation manifesto, is as close as anyone to the epicenter of the Twine explosion. She started using Twine in 2011 to create games like the kinky, adult-themed ENCYCLOPEDIA FUCKME, and more recently published a simple how-to for Twine game authoring (auntiepixelante.com/twine/). She sees Twine as an ideal tool to allow a wider variety of people to make games.

"The mainstream sort of game communities are really hostile to a lot of what they perceive as the other, to women, to queer people, to trans people, and so [the outsiders] don't generally have access to the same means of creation," she told *Game Developer*. "They're dissuaded from learning to program, to going to engineering school or whatever. They don't traditionally have the means to develop those kinds of skills."

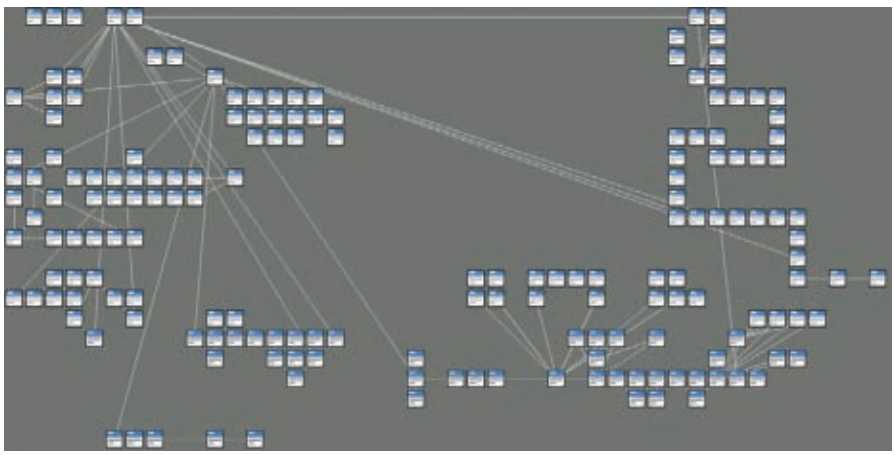
Twine's ease of use helps to mitigate that systemic disadvantage. "There's almost no programming involved, and that's a really traditional technological barrier to making games for a lot of people," said Anthropy. "If you can write you have almost all the skills you need to make a Twine game. And a lot of people can write." Anthropy further cited an hour-long Twine game jam she held in Toronto, where the diverse crowd of first-time game makers included a number of senior citizens.

On her blog, Anthropy posted a list of notable Twine games that've sprung up during the recent renaissance. "I'm not cherry-picking much at all," she said. "[There's] a really noticeable majority of women, queer, gender-nonconforming authors. I mean, it's basically an inversion of most game-making communities in the mainstream, which is really interesting. It looks genuinely different."

"Genuinely different" is a good way to describe both rat chaos and howling dogs, two key games from the new scene. J Chastain's rat chaos (monsterkillers.com/games/2012-07-18%20Rat%20Chaos/ratchaos.html) begins under the pretenses of a typical text adventure, but its stereotypical IF commands become increasingly nonsensical as the narrative transmutes into a soul-baring, fourth-wall-breaking monologue from the author. Porpentine's howling dogs (aliendovecote.com/uploads/twine/howling%20dogs.html), meanwhile, uses beautifully evocative language to weave a loose tale that may or may not be about lost love, VR, and day-to-day life in captivity. ZORK these games are not.

Other sidelined, would-be games creators are noticing. "[Releasing RAT CHAOS] was like raising a small flag," said J Chastain. "I think it's helped talk to some people. Wherever you are, if you're queer and feeling isolated and invisible, you can get on the web and start putting flags up and demonstrating that you exist."

—Alexandra Hall



A Twine project.

PUBLISHER Rad Dragon
DEVELOPER Rad Dragon
RELEASE DATE November 8th, 2012
DEVELOPMENT TIME 3.5 months
DEVELOPMENT BUDGET $17,996.24's worth of rent, ramen, and resources
 # OF LINES OF CODE IN THE GAME 14,638
A FUN FACT SHOVE PRO was originally prototyped for a USC game jam where the theme was "shadows." The judges called the game's use of theme "tangential bullshit."

Shove Pro

http://raddragon.com/shove-pro

MIKE SENNOTT AND TEDDY DIEFENBACH COMPRISE RAD DRAGON, A NEW INDIE STUDIO WHICH RECENTLY RELEASED SHOVE PRO FOR IOS. THE USC GRADS ARE NOW WORKING TO FURTHER DEVELOP THEIR RESPECTIVE THESIS GAMES, QUICKSILVER: INFINITE STORY AND THE MOONLIGHTERS. I CHECKED IN WITH THEM TO HEAR THEIR THOUGHTS ON USC'S GAME PROGRAM AND OF DEV LIFE AFTER GRADUATION.

Alexandra Hall: Has the game education landscape changed a lot since you originally shopped around for a school?
MIKE SENNOTT: Games education has certainly grown in popularity, so there are programs in more places, and you probably won't get as many weird looks for spending money on a games degree. It is unambiguously awesome that more people are getting the chance to develop their skills and create something they believe in, but as a result it's going to get increasingly difficult for a student to stand out in the field.

AH: ***Are you satisfied with how USC's program prepared you for "real-world" development?***
MS: Definitely. USC's grad program is very focused on getting out there and making stuff, so we learned practical development skills such as coding, iterative design, leadership, and scoping down. But maybe the most helpful thing we got out of the program was the culture of getting stuff done. We could see our classmates producing shockingly good work in very little time, and we knew that we had to do our best instead of psyching ourselves out. I think carrying that attitude with us is one of Rad Dragon's main advantages. At big companies there's a lot of waiting on other people or pushing for approval, but as indie devs we have no excuses not to get things done. I think USC has prepared us to take full advantage of that potential for drive and agility.
TEDDY DIEFENBACH: Mike and I were both fortunate to work in the industry before we joined the Interactive Media Division for our MFAs, so we had that perspective going in. When we decided to form Rad Dragon instead of taking jobs after school, we didn't see any reason why we couldn't carry the culture Mike mentioned into our company. While we now have to be cognizant of business planning and strategy, it's very important to us that we leave room for the experimentation and fast pace that the IMD cultivated for us.

AH: ***Both of your thesis projects feature interactive story elements. Was that easier to get off the ground in academia?***
TD: Before we'd had time to prototype and develop our thesis games, it would have been difficult to convince a big company to invest in time to experiment with QUICKSILVER'S procedural cartoon storytelling, and it would be hard to prove with just a PowerPoint and some concept art that people would connect like they have with THE MOONLIGHTERS, a heist game about washed-up 1950s crooners. USC's MFA program insisted that we set out to contribute something valuable and original to the game industry. But the freedom of academia and indie development is a double edged-sword. No one stops us from trying these things, but we have tighter resources. For our academic theses and our games at Rad Dragon, we've had to be game designers, coders, writers, producers, and pitch men. We can do whatever we want, so long as we find a way to make it happen.
MS: At USC, I was able to spend much of a year working on QUICKSILVER, scrap it, learn from it, and spend my thesis making a much better-designed iteration. Only in academia, where the emphasis is on experimentation and learning, can you take that sort of risk without fearing failure. In school, if your grand game idea doesn't turn out like you'd planned, you won't go bankrupt like an indie might. You can write a paper on why it didn't work, and the interactive field can still profit from your time.

AH: ***Given your trajectory, might a big studio situation cramp your style?***
TD: It's never really been about Big versus Indie to us. Our friends at L.A. triple-As like Naughty Dog and Insomniac blow our minds with their work! There are just some game ideas that we have a burning desire to create together, and the only way we could was to take the financial risk ourselves and form Rad Dragon. We love the freedom to do what we do—build games that entertain, but incorporate exciting original concepts in gameplay and narrative design. With any luck, we'll be able to continue forward with our more ambitious ideas and grow Rad Dragon into a studio that will be making games for all eternity. *—Alexandra Hall*

Shaw'll Ready For This?

CARYL SHAW HEADS TO KIXEYE

BY THE END OF 2012, IT SEEMED LIKE EXPERIENCED DEVELOPERS AND EXECUTIVES ALIKE WERE LEAVING SOCIAL GAMES JUST AS FAST AS THEY HAD BEEN JUMPING IN AT THE BEGINNING OF THE YEAR WHICH IS WHY WE AT GAME DEVELOPER WERE INTRIGUED TO HEAR THAT FORMER MAXIS/EA/NGMOCO PRODUCER CARYL SHAW WAS HEADING OVER TO KIXEYE FOR AN EXECUTIVE PRODUCER GIG. SHE TELLS US WHY SOCIAL IS STILL THE PLACE TO BE.

PATRICK MILLER: ***You've been in the biz for a while, first at EA, then jumping into free-to-play with ngmoco before that was an industrywide trend (well, in the U.S., anyway). What prompted the hop to KIXEYE?***
CARYL SHAW: KIXEYE is starting to move further into mobile, beginning with the launch of BACKYARD MONSTERS: UNLEASHED, and I just saw a huge opportunity here. Developing midcore games for mobile is a door that is just now opening, and given KIXEYE's dominance with synchronous PvP games on Facebook, I think this is a great moment to make the shift and bring our class of social games to mobile. I was looking for a company that had a really strong focus on quality where I could contribute at a fundamental level—and help drive the strategy. Plus I knew a few people here and jumped at the chance to work with them again.

PM: ***Be honest—it was the recruitment video on YouTube, wasn't it? Or those meme posters on BART?***
CS: Heh. I actually loved the video. Most people who know me at all will tell you I'm not a terribly shy flower, and it seemed like a company where my vibrant and creative personality would fit in well.

PM: ***Toward the end of 2012, we saw something of a downturn in the social games biz, most notably punctuated by Zynga's layoffs and exec departures. Why get into social games now?***
CS: I believe social is here to stay, but I think it has to be done in a way that feels meaningful for the gamer. My goal is to provide players ways to compete and collaborate with each other through games. Social interactions in games aren't going away, but will definitely need to evolve. Whether you are playing with people in your social graph (people you know) or in your interest graph (people who enjoy the same types of games that you do) opens up new ways to promote collaboration and competition. I think true competition has been missing from social games, and I can't wait to see where that goes in the next few months.

PM: ***You've been out of the traditional console/PC packaged-product side of the biz for a while. Do you miss it?***
CS: Watching the videos of the game footage for the new SIMCITY makes me super homesick for my time at Maxis. I would have loved to contribute to that game, but overall I would not trade my career shift to mobile even for that. I'm not sure I could spend five years working on one title again, but I love those guys for doing it. They are making a Maxis game I'll get to play having never had one bug assigned to me! But frankly, the fast-paced, server-based mobile world appeals more to my ADD side—there's always something shiny and new to explore and make the most of, plus I'm learning new stuff all the time.

PM: ***What's the next big game dev market trend that we (and everyone else) should be paying attention to?***
CS: I think the customers to watch are those that just started playing games in the last year or two—the casual gamers who don't think of themselves as gamers. In the not-so-distant future, those folks are going to start having higher expectations from their game experiences. They're going to mature as gamers and consumers, and it's our job to creatively address those new expectations and make truly innovative games that continue to foster their newly found love of gaming.

—Patrick Miller

Who Went Where

- The Secret World's creative director, **RAGNAR TORNQUIST**, has parted ways with Funcom to create a new independent studio following the premium MMO's rocky launch. Thanks to a revenue-share deal with Funcom, Tornquist's acquired the rights to create a sequel to his adventure game classic, THE LONGEST JOURNEY. Tornquist will also remain in an advisory role for The Secret World.
- **SIM DIETRICH** has joined free-to-play mobile MMO developer Machine Zone. Dietrich comes from OnLive, where he managed the platform, UI, engineering and game performance teams, and is credited with the invention of several patented technologies during his previous engagements with Intel and Nvidia.
- **NEIL YOUNG AND BOB STEVENSON**, two of the original founders of mobile games company Ngmoco, have left the company to move on to other opportunities. Founded in 2008, the company was bought out by Japanese social games giant DeNA for $400 million in 2010. Young and Stevenson are said to be "moving on to our next adventure."
- Mobile games company PlayFirst has hired **PAUL CHEN** as vice president of business development. Chen comes by way of Papaya Mobile and Nvidia, where he was product manager for the graphics company's Tegra line of mobile processors.

New Studios

- **MICROSOFT'S OPENED A NEW STUDIO** in Vancouver that will focus on creating triple-A games for the Xbox 360: Black Tusk Studios. Drawing a comparison to HALO, studio manager Mike Crump says that the studio is "working on Microsoft's next big entertainment franchise ... We are building something from the ground up."
- Perhaps influenced by local financial incentives, **TAKE-TWO IS MOVING** its 2K West QA Testing studio from Northridge, California to downtown Las Vegas. The new location will likely support upwards of 150 employees.
- Since being laid off from Reverge Labs, the development team behind SKULLGIRLS has formed a new studio, **LAB ZERO GAMES**. Still working with publisher Autumn Games, Lab Zero will continue supporting its hardcore 2D fighter.

THE 15TH ANNUAL *GAME DEVELOPER*

FRONT LINE AWARDS 2012

For 15 years, we at *Game Developer* have continued to honor the best development tools in the business with our Front Line Awards. Here's how it works: We started with an open nomination process where readers and at-large community members could nominate their favorite tools. From there, the *Game Developer* and Gamasutra editors, with a little help from our respective advisory boards, sorted through those nominations to come up with a list of finalists that we put to a community vote to determine which tools come out on top.

This year's crop of Front Line Award winners includes a few repeat winners and a few new members to the club, which is not so surprising, considering how our industry somehow manages to simultaneously change and yet stay the same. Havok Physics, Pro Tools, and Unreal Engine 3 had repeat wins this year, while Luxology's modo 601 managed to unseat Autodesk 3ds Max for best Art tool, and Bugzilla finally won a much-deserved nod in the Programming category. Also, we nixed the Networking category in favor of a Best Free Tool category; as more and more game developers start working in small and scrappy indie studios, it becomes ever more important for devs to make sure they can work with tools that don't break their (nonexistent) budgets.

Congratulations to the winners! Here's to another year of great games—and the tools to build them with.

—Patrick Miller

HALL OF FAME

Unity 3D

UNITY TECHNOLOGIES UNITY3D.COM

In the early part of the decade, the game industry planted the seeds of change. High-quality development tools, historically hidden behind high-cost barriers, were being opened up to the masses for free or cheap. Along with this democratization of tools came an increased focus on speed and ease of development. Fast-forward to 2012: Tools across the board are easier to obtain and use, which has enabled the indie sector's rise to prominence, and one of the most important tools to come out of this era is Unity. Out of this storm of experimentation, Unity bubbled to the top, offering an unprecedented mix of power, value, and ease of use that has put new faces in the game development scene while simultaneously offering established developers a compelling tool.

Speed of development is a key factor in making games, and this is arguably Unity's strongest suit. Coding for Unity is simply a nice experience; it has a clean, component-based architecture that is both powerful and flexible. Since it's built on Mono, those who've had run-ins with .NET will be partly familiar with it, while its JavaScript support eases the transition for web developers. The code side of things meshes well with the Unity Editor, which provides many of the commonly used development tools (level editing, asset importing, animation tool, particle designer, and so on).

Unity also has a lot of power in the non-coding tools, and very few barriers to getting your hands dirty when you need coding support; the whole package plays nicely together. This is a powerful combination that allows you to fluidly move between using code-based solutions and tool-based solutions without struggling against Unity. The editor itself can be modified through code, allowing you to tailor your own tools, which is a boon if you need to provide extra support for non-coders that integrates with the rest of Unity. The broad deployment options that Unity provides are also a big time saver if you're interested in cross-platform releases, or if you need to standardize engineers with disparate skills onto one toolset.

Unity's potential for speedy development makes it particularly useful for small studios and indies, especially because it offers a free basic license and reasonable pricing for its pro and platform licenses. I worked at a small iOS independent studio on a game called GREM LEGENDS, and one of the engineers didn't know native code—but we needed all hands on deck. We were all pretty well-versed in C#, had a little bit of Unity experience, and didn't have time to build a bunch of tools from scratch, so we chose Unity. With Unity we got the game done on a tight schedule, which would have been tough if we had taken a different production path.

Once I got comfortable using Unity, I started using it as my standard tool for game jams as well. I used it at the "What Would Molydeux?" jam, and was able to finish the game coding solo while still going home and getting some sleep each night.

We would be remiss if we didn't acknowledge that part of what makes Unity special is the community of Unity devs. There are a *lot* of people using Unity, which means it's easy to share knowledge with and support other devs. The Unity team also provides the community with the Asset Store, which allows developers to distribute plug-ins that they've created. Beyond providing a means for developers to sell their tools, it's a centralized repository of tools, making hunting for what you need not quite as painful.

Unity is still a generalized engine, which means compromises that keep it from being a cutting-edge tool, but for many developers the tradeoff is well worth it. Plus, it abstracts away much of the extremely technical work involved in reasonably efficient rendering, opening the gate of development to those who aren't Direct3D/OpenGL gurus. For studios operating on the bleeding edge, or who need different workflows than what Unity has to offer, it can still be valuable for quickly prototyping, iterating ideas, and tinkering.

Unity reflects the change that has been occurring for the past several years in the industry. It's approachable, both from a financial and technical perspective, and offers excellent results for the amount of effort you put into it. What's more, Unity thrives in the diverse ecosystem of devices that game developers need to address, and with its swift development speed, it can have a place in all manner of studios, from the four-person team at a game jam to the triple-A juggernauts. Unity has opened the doors of game development for many, and we wouldn't have some great games without it.

*** ***Elijah O'Rear is a software engineer at Midverse Studios.***

ART

modo 601

LUXOLOGY
LUXOLOGY.COM/MODO

3D artists in game development often need to use several different software packages to create assets for a modern console game. A video game artist has to adapt to a pipeline of tools and applications that have different user interfaces, workflows, and file formats. As a result, today's game artists are asked to create next-generation art using last-generation tools that are more like a collection of separate ideas rather than a complementary ecosystem. With the release of modo 601, Luxology has created a compelling modeling and rendering tool that seamlessly incorporates easy-to-use character rigging, texturing, and shaders—which is strangely unique among 3D software packages.

I first encountered modo while looking for ways to improve the quality and speed of character production on GOLDEN AXE in 2007—I picked it up to edit the UVs on my character models more easily than I could with comparable applications at that time. I gradually found myself using modo more often for other tasks, such as modeling and texture rendering and cleanup, as many of the tools were the same ones I had used to edit UVs.

Fast-forward to spring 2012 with the rather timely launch of modo 601. With its implementation of Non-Photoreal Rendering (NPR) shaders and character rigging tools, I am able to realize the promise of creating an entire game character from beginning to end, starting from concept art, to real-time content, to a polished high-resolution model and renders that fully represent what I had imagined when I started on a character idea or design.

Typically, a game character requires two models: a real-time mesh, and a more detailed version of the same character that is used as a source for texture maps to be applied to the in-game model. The high-resolution model is often generated in external sculpting applications, and is mostly used only for texture or re-topology reference—if someone needs a detailed model for a rendered FMV or printed promotional

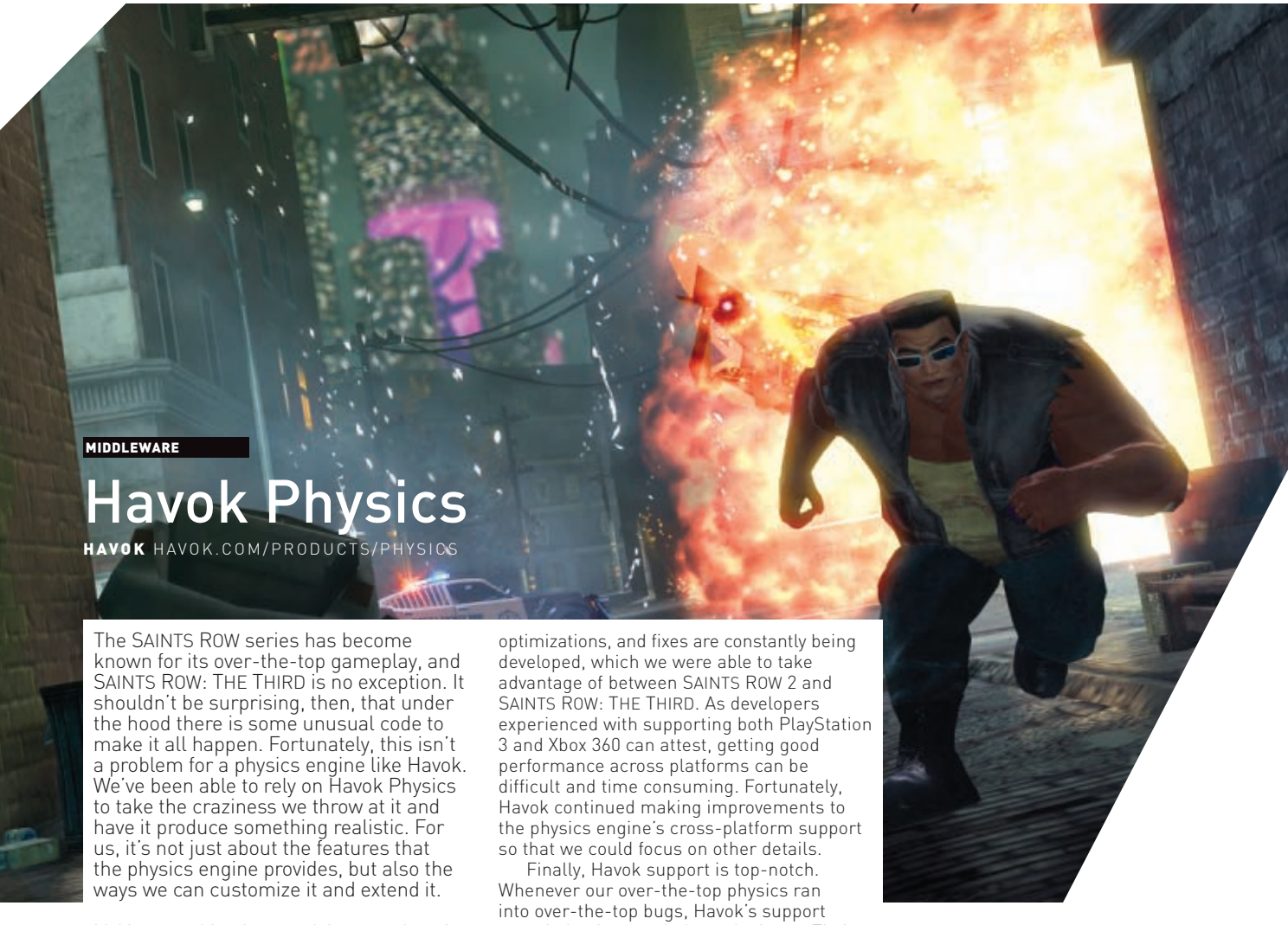
images, the artist will either make a new model or just make do by rendering in-game assets and touching them up in Photoshop. The end result looks different (sometimes dramatically so) from the actual in-game character.

For SLY COOPER THIEVES IN TIME, our goal was to make the in-game assets faithfully represent the original signature style of the classic SLY COOPER characters, while at the same time updating their look for a modern console game. With modo, I was able to create both the in-game and high-res models within a single scene by using its sculpting tools to convert the low-res model to a highly detailed Sub-D model. Since I used modo for the high-res model, I could apply the same textures created for the in-game model, quickly add a skeleton, and then use the PoseTool to position the characters into various poses that show off their personality or provide compositional narrative.

The last step was to render using a combination of cel shaders, material presets, and textures to get a rendered, slightly stylized, painterly look that matches to the overall style of the game. Since I had been reusing the same high-res models used to create the in-game geometry, the rendered images accurately and consistently represent the game characters and their subtle style and form.

Thanks to modo 601's new shader and posing tools, we were able to rapidly provide high-quality images to Sony for their marketing efforts; indeed even our quick lighting and pose tests were sometimes mistaken as final art from time to time.

**** John Hayes is a senior character artist at Sanzaru Games.*



MIDDLEWARE

Havok Physics

HAVOK HAVOK.COM/PRODUCTS/PHYSICS

The SAINTS ROW series has become known for its over-the-top gameplay, and SAINTS ROW: THE THIRD is no exception. It shouldn't be surprising, then, that under the hood there is some unusual code to make it all happen. Fortunately, this isn't a problem for a physics engine like Havok. We've been able to rely on Havok Physics to take the craziness we throw at it and have it produce something realistic. For us, it's not just about the features that the physics engine provides, but also the ways we can customize it and extend it.

Making a world as large and dense as that of SAINTS ROW: THE THIRD inside the memory limits of current consoles is challenging. Although Havok provides multiple options for terrain collision representation, we chose to implement our own by extending one of the provided options with our own storage scheme custom tailored to our usage needs. The result was a fast and very memory-efficient representation of our terrain collision. Havok provided a solid foundation upon which we could then expand.

Vehicles are very important in SAINTS ROW games, so we are constantly striving to provide better vehicle physics. Once again, we used the Havok vehicle physics kit as a foundation and then customized and built on top of it. We were able to implement our own drifting physics by changing the way vehicle friction is simulated; make our own tank turret physics by combining Havok constraints and motors with our vehicle weapon system; and build all kinds of watercraft, airplanes, helicopters, and hybrid vertical takeoff or land (VTOL) vehicles by building on top of Havok.

Even with everything it already does, Havok Physics is always improving. New features, optimizations, and fixes are constantly being developed, which we were able to take advantage of between SAINTS ROW 2 and SAINTS ROW: THE THIRD. As developers experienced with supporting both PlayStation 3 and Xbox 360 can attest, getting good performance across platforms can be difficult and time consuming. Fortunately, Havok continued making improvements to the physics engine's cross-platform support so that we could focus on other details.

Finally, Havok support is top-notch. Whenever our over-the-top physics ran into over-the-top bugs, Havok's support team helped us track down the issue. Their knowledgeable team and excellent support tools, such as the Havok Visual Debugger, made a big difference when we were dealing with difficult bugs. Beyond just helping when asked, they also contacted us to inform us of their future plans for the engine and ask for feedback on what to improve for future versions. On occasion, we even had them on-site for some hands-on attention toward the end of a project.

We've been using Havok Physics for several years, and it just seems to be getting better and better. More and more of the cross-platform differences are handled behind the scenes so that we don't have to worry about them like we used to. Interesting new features are being added, so that we can expand upon them to make crazy new games. When those games run into equally crazy problems, it's comforting to know Havok support is there for us. SAINTS ROW has really benefited from its use of Havok Physics, and we look forward to using them in the future.

*** *Shawn Lindberg is a senior programmer at Volition Inc.*

AUDIO

Avid Pro Tools

AVID TECHNOLOGY
AVID.COM/US/PRODUCTS/FAMILY/PRO-TOOLS

When it comes to digital audio, Pro Tools is considered the standard digital audio workstation (DAW). While Pro Tools was a must-have in every major recording studio around the world by the mid-1990s, it took a little longer to find acceptance among game studios due to its high price tag. But these days just about every publisher I'm aware of uses Pro Tools, as well as just about every major game-centric audio studio.

The main edge Pro Tools has over just about every one of its competitors is reliability. When you are faced with linking Pro Tools to your game engine to process hundreds of thousands of lines of dialogue in a complex chain of integration, you can't afford to lose data during a recording session or an export. For example, I have used Pro Tools in conjunction with both commercial (Gallery Software) and proprietary voiceover software that will automatically trim voice recordings, perform a spectrum analysis, and based on scripting, correctly process and name VO takes. Many DAWs tend to stumble, crash, and hang the system from time to time. If your Pro Tools system is set up properly from the get-go, the chances of it crashing are almost zero—and when you're recording expensive star voice talent, rock-solid performance is a must.

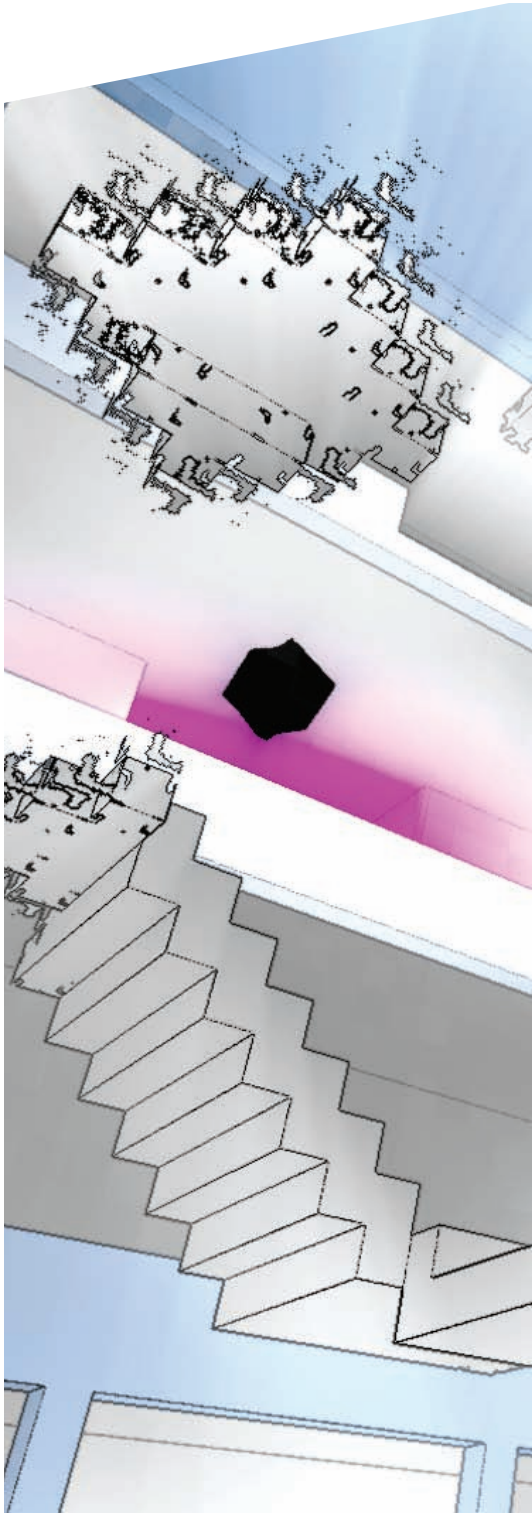
When it comes to dealing with recording a hundred orchestral players and foley sound sessions, where so many people are working that your hourly rate starts going through the roof, you need a DAW that doesn't let you down. Whether recording down the road or at Abbey Road, most likely you'll find a Pro Tools system connected that the engineers "just don't have to worry about." Unless you are recording in the field with a Sound Devices or Nagra system, Pro Tools will be there.

In addition, Pro Tools has made an effort to be reachable to the PC user as well as the budget studio, so if you're looking for something in the same family as the huge studios, opt for Pro Tools Express or Pro Tools SE with an Mbox interface. When you decide your wallet and your needs require the top end, it's a simple matter to upgrade to Pro Tools|HD.

Another thing Pro Tools has going for it are its RTAS and AAX plug-ins, which are widely regarded as the highest quality in the business from amp simulation to compression. There are also a great deal of virtual instruments available.

I'm waiting for the day Pro Tools will directly connect with game hardware and software to create one pipeline from asset creation to integration. Hopefully, that day isn't too far away.

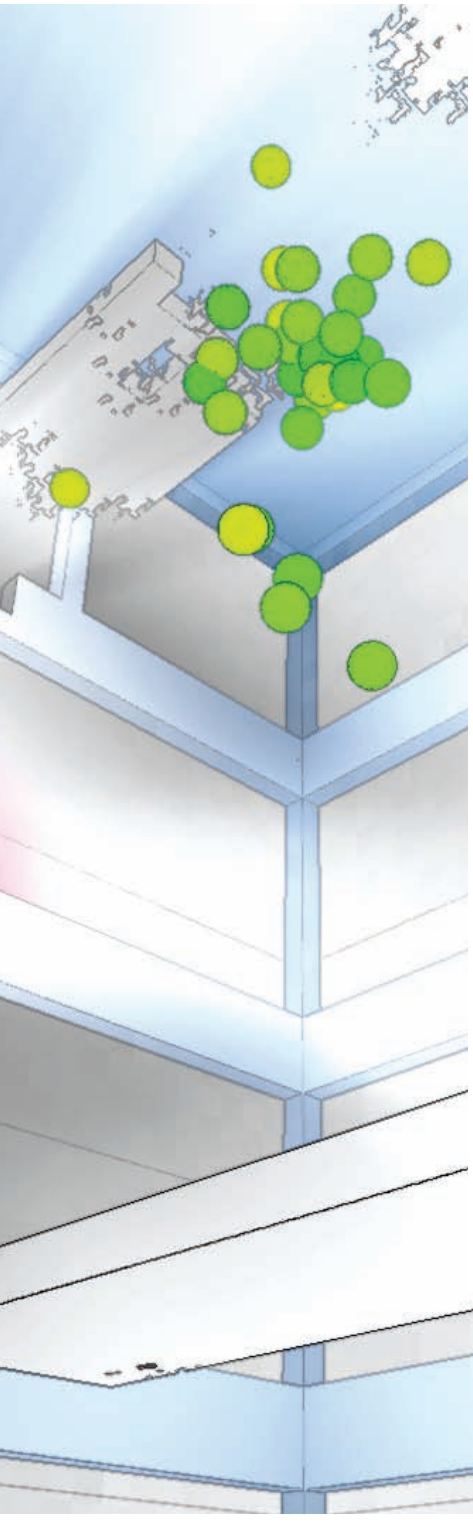
**** Alexander Brandon is the president of Funky Rustic, an audio outsourcing group, and vice president of the Game Audio Network Guild.*



ENGINE

Unreal Engine 3

EPIC GAMES UNREAL ENGINE.COM/UDK



Whether you're a student trying to get into game development, an indie developer aiming to make a splash in the PC or mobile markets, or a studio looking to develop the next big thing on consoles, one thing that should definitely be high on your radar is Unreal Engine 3.

Available under multiple licenses, starting with the most affordable Unreal Development Kit, you can create high-quality, standalone commercial titles, using the same tools that Epic and its licensees used to create games such as GEARS OF WAR, MASS EFFECT, BORDERLANDS, and INFINITY BLADE.

I was first introduced to Unreal 3 in 2007 through a university degree that included creating mods for ROBOBLITZ and UNREAL TOURNAMENT. When a few of us were hired by a large studio in 2008 to work on an unannounced Unreal Engine 3 game, the tools available to us as full licensees were essentially just a more up-to-date version of what we were already familiar with. During pre-production, design specs could be turned into fully fledged prototypes in a matter of days, due to Unreal's mature, efficient toolset.

When I later worked on other games at the company using their own proprietary tech, the tools were a mess. I learned to appreciate the enormous difference between tools that a programmer can use to get the job done, and tools that artists and designers need to use every day to effectively do their jobs.

Good tools can be the difference between a game that gets finished on time and under budget and one that gets stuck in development hell. After seeing the damage that bad tools could do to a project, when I began working on my own games again in early 2009, I went straight back to working with Unreal. With two decades of experience being at the front line of game technology, an expansive portfolio of highly successful triple-A games and a growing list of independent releases via the UDK, new updates every month, and good documentation, it's pretty clear that Epic knows what it's doing when it comes to developing an effective engine.

From a programming perspective, there are a couple of caveats to be aware of. The UDK will only give you access to a higher-level scripting language known as UnrealScript, not the lower-level native C++ code that runs the engine beneath it. UnrealScript is like a user-friendly combination of C++ and Java, with a focus on development simplicity and power over raw execution speed. Don't let that deter you, though.

Using nothing but UnrealScript and the Unreal Editor, I managed to create ANTICHAMBER, an award-winning non-Euclidean psychological exploration game, complete with custom physics, obscure spatial navigation rules, and a unique visual style made up of multiple rendering techniques, edge detection, and a custom inverse lighting system. I did all of the programming, design, and art myself, and even won the award for Technical Excellence at the 2012 Independent Games Festival, without ever having to go near any of the lower-level native C++ code.

Several successful independent titles have been developed using the Unreal Development Kit, such as THE BALL, SANCTUM, and HAWKEN. Other games, such as DUNGEON DEFENDERS, began development using the UDK and then upgraded their license when they made the transition to consoles. One particularly interesting case was QUBE, which was released on Steam at the start of 2012 by a team with no programming experience.

One of the most important aspects of developing games is knowing what your strengths and weaknesses are. Some people like having control over every single detail, but to me, there is nothing more daunting than staring at a blank slate, trying to work out where to start. By using Unreal and leaving the core foundational work in the hands of experts, I can just focus on working on the things I actually care about. I'm interested in exploring the boundaries of game design, but I hate reinventing the wheel.

**** Alexander Bruce is an independent developer and the creator of* ANTICHAMBER.

FREE TOOLS

Blender

THE BLENDER FOUNDATION (OPEN SOURCE)
BLENDER.ORG

Blender is a free, open-source 3D art suite with a combination of powerful modeling, unwrapping, and rendering tools that is growing in notoriety as its user community demonstrates that you can still make professional-quality movies and game art with free tools.

I first started working with Blender six years ago, while working toward my classical art degree, and felt an affinity with computer graphics that led me to spend the rest of my university years learning to use Softimage XSI and 3D Studio Max. Blender's accessibility remains one of its strong points and is helping artists around the globe to express themselves in new ways.

Our development studio (Nine Dots) was founded on a small budget, so we opted to use Blender instead of paying $5,000 per head for commercial 3D software. As the studio's founder started gathering a team, my previous experience with that software was a strong asset and I was hired. I've used Blender ever since, and new Nine Dots artists are taught to use it when they start out in our company.

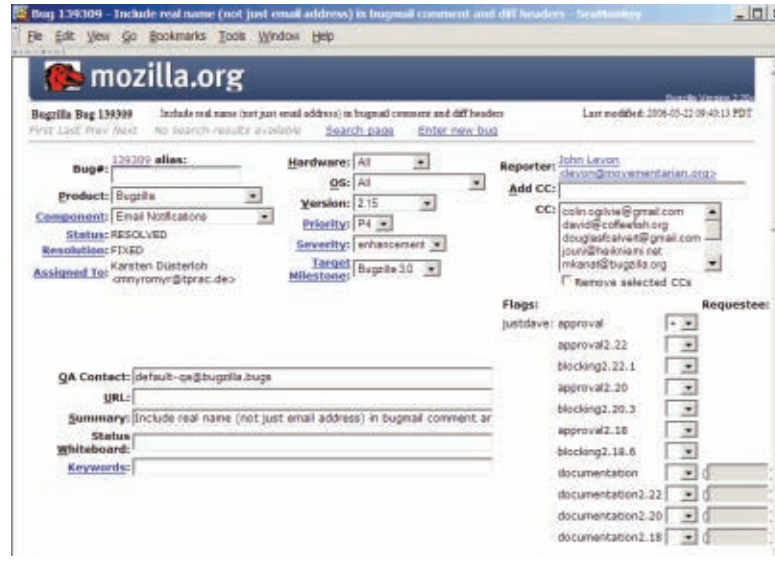
After a few weeks of adaptation—getting used to the interface and customizing our keyboard shortcuts—the 3D artists at Nine Dots were able to reach the same production speed they had on the software they were used to. In my experience, I have actually found Blender to be far faster than any of Autodesk's products in terms of modeling and unwrapping, at least for the low-poly models required for video games.

I also made good use of my graphics tablet in Blender while working on our first game, BRAND. Using the Poly Paint mode, I drew the lights and shadows on the 3D mesh itself before baking the result in the diffuse texture. That single feature I found while exploring the software has helped us obtain a painterly look for our game. Other features use the tablet, such as the newly implemented sculpt mode, which lets the artist manipulate the geometry in a manner similar to ZBrush.

After we released BRAND on Xbox 360 and PC, we started work on a more ambitious project with fully detailed characters. Once again, Blender has proven to be a great tool for creating an elegant facial geometry and for animating the bodies and faces of our human (and nonhuman!) characters. We also used Blender to edit some of our trailers and gameplay videos; since the video editing tool is integrated with the rendering software, the production pipeline is streamlined, saving us some precious time!

In short, Blender is an accessible and customizable tool that can be optimized to greatly increase production speed. For video game creation, I can speak from experience that Blender's modeling and unwrapping tools are on par with those of its costly competitors. I'm eager to see it grow as it refines its strengths and irons out its last kinks!

*** *Etienne Vanier is the lead artist at Nine Dots Studio.*



PROGRAMMING

Bugzilla

MOZILLA FOUNDATION (OPEN SOURCE)
BUGZILLA.ORG

Bugzilla isn't the sexiest product in the world—after all, it's bug-tracking software. But it's one of those required tools in our development arsenal, one of those tools that we all rely on during the most critical moments of the development process.

For me, Bugzilla was particularly convenient while I was working with a bilingual team in China. In this situation, we had a group of about 20 developers of all disciplines working on building a free-to-play online PC game. In our case, Bugzilla was efficient, quick, and easy to learn and use.

Bugzilla has some key features that I found essential in our development process. The ability to mark or update multiple bugs simultaneously was a workflow convenience I could not live without. I also appreciated the tracking reports, which gave me an up-to-date chart that showed how quickly we were closing bugs off our list, which users were behind in closing their issues, and how many bugs of each priority category were remaining. This was useful for feature tracking as well, because we could earmark requests with their own category, which could then serve to roll up new features for the team to implement.

Bugzilla's tracking features are complemented by a whole host of other features that contribute to its utility and usability. Its email notifications allowed me to immediately react to updates and communicate build-testing results back to the team within shorter timeframes. Its automatic duplicate detection feature (a form of autofill) lets the user find potential bug duplications before they are committed to the database. Time tracking allows the team to set deadlines and test against user time estimate accuracy.

Perhaps the most useful part about Bugzilla, though, is that it's a free, open-source product. This cannot be understated. In an industry that is now party to an enormous host of smaller developers working with lower budgets to make their products, Bugzilla is a welcome respite to the incoming tool costs associated with new cloud subscription models that have become the norm for many development toolsets. Because of my positive experience working with Bugzilla, I now automatically look toward open source solutions to our tool problems before I consider the licensed per-seat solutions that are currently on the market.

To that end, I salute the Bugzilla team, as well as everyone else who is offering their time to make the product even better, and hope that we in the game industry do our part to keep it alive well into the future.

**** Carey Chico is a game industry veteran and a member of the* Game Developer *advisory board.*

GDC 13
GAME DEVELOPERS CONFERENCE®
MARCH 25-29, 2013 MOSCONE CENTER SAN FRANCISCO, CA

GDC 2013 DEBUTS SCHEDULE BUILDER, HIGHLIGHTS NEW ROUNDTABLES

With planning for March's GDC 2013 in San Francisco now underway, the show's organizers have debuted more than 20 sessions via the initial Schedule Builder, with hundreds more due over the next few weeks.

As always, prospective GDC speakers are working closely with the advisory board members, who are individually mentoring talks through multiple submissions stages and into the official program. As part of this process, GDC 2013's very first announced talks are now available online via its official Schedule Builder, with highlights to be further explored soon, including a "poster session" on Metacritic scoring, plus talks from Double Fine, Frictional Games, and Microsoft.

Alongside the Schedule Builder, GDC organizers have also added a pair of roundtable talks focusing on managing art teams and incorporating designers into your production processes. Unlike GDC's standard lectures and panels, these roundtable sessions allow attendees to sit down with the host to engage in face-to-face discussions, share their personal experiences, and learn from their fellow developers.

First, Keith Self-Ballard, a art director at Blizzard Entertainment on the company's unannounced next-gen MMO, will host the "Art Director and Lead Artist Roundtable," which will provide an open forum for artists and art directors to exchange ideas regarding how studios should manage, lead, and direct their art teams. This session will address the most pressing issues facing today's game artists, and attendees will walk away with a better understanding of how to overcome these challenges.

In the second new roundtable, storied developer and University of Advancing Technology professor David Wessman (TIE FIGHTER, THE CHRONICLES OF RIDDICK: ESCAPE FROM BUTCHER BAY, SAINTS ROW) will help attendees look at how designers should interact with developers from other disciplines. His roundtable, dubbed "Whose Design Is It Anyway? Game Designers and Development Teams," will help designers and other developers understand each other's roles, improve cross-studio communication, and create a more collaborative atmosphere across their entire studio.

Both of the above sessions will take place as part of GDC 2013's Main Conference, which is open to All Access and Main Conference pass holders. Discounted Early Bird registration is now open on the show's official website, and GDC 2013 itself will take place March 25-29 at the Moscone Center in San Francisco.

Be sure to keep an eye out for even more updates on GDC 2013 in the coming weeks, as the show's organizers have plenty more talks to announce for the upcoming event. For all the latest information on the Game Developers Conference, visit www.gdconf.com, or subscribe to regular updates via Facebook, Twitter, or RSS. (GDC is owned and operated by UBM TechWeb, as is *Game Developer*.)

BECOME A BETTER WRITER, PROGRAMMER WITH GDC 2013'S SPECIALIZED TUTORIALS

No one can make a truly great game without first understanding the basics of development, which is why the next Game Developers Conference in San Francisco will feature a number of robust tutorials to help attendees sharpen their essential skills.

GDC 2013's organizers have debuted a pair of tutorials to help developers create better narrative content, and understand the mathematic principles behind modern game programming.

In the first tutorial, LucasArts lead narrative designer Evan Skolnick will return to GDC to host "Game Writing Fundamentals in a Day." During this session, he'll provide a comprehensive primer for game writers, covering the basics of good story structure, character development, and dialogue writing. Of course, developers in other disciplines will benefit from this tutorial as well, as they will learn how to bridge the gap between game writers and the rest of the development team.

The second new tutorial is another GDC favorite aimed at helping developers understand the foundation of modern game programming. The session, titled "Math for Games Programmers," will bring together a host of programming experts to cover everything from basic mathematic principles to the complex topics every programmer should master.

Confirmed speakers for this session include Jim Van Verth (Insomniac Games), Manny Ko (Imaginations Technologies), Gino van den Bergen (Dtecta), Stan Melax (Intel), Squirrel Eiserloh (The Guildhall at SMU), Robin Green (Microsoft), and Graham Rhodes (Applied Research Assoc., Inc.).

PHOTOS COURTESY OF GAME DEVELOPERS CONFERENCE

Foundations of Digital Games

fdg 2013

www.fdg2013.org

Chania Crete Greece

14 - 17 May

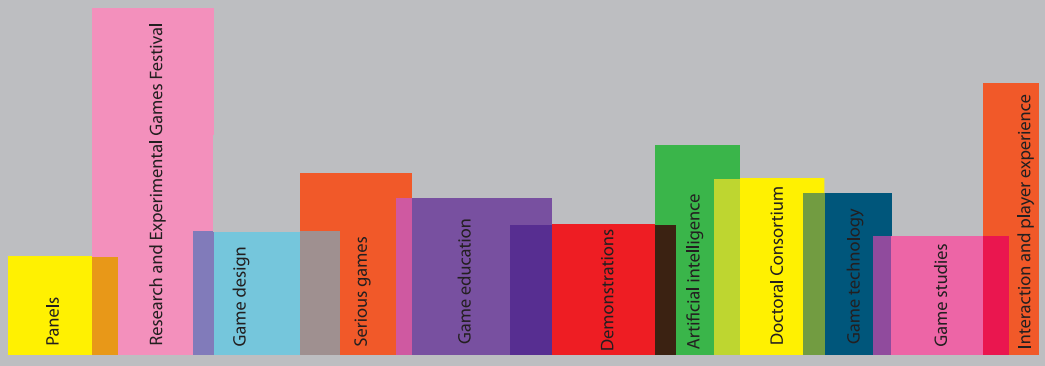
Take a step back from the trenches of development and explore topics that a looming crunch time often precludes. At the conference Foundations of Digital Games conference, for the 8th year in a row leading researchers and developers from around the world will be presenting their latest work.

Topics range from:
- adaptive games - procedural content generation - player psychology - game analytics
- player experience - game culture - games for change - game philosophy

Contribute to this forum focused at advancing the study of digital games both as cultural and technological artifact.

Join us in Chania, Greece for FDG 2013!

Tracks



Microsoft Research

Microsoft Studios

unity

FDG 2013 is a conference of the Society for the Advancement of the Science of Digital Games

BRING YOUR TOOLS TO BROWSERS

USING A WEB-BASED ARCHITECTURE FOR DEVELOPMENT TOOLS CAN SLASH ITERATION TIME AND SPEED UP CROSS-PLATFORM DEVELOPMENT. HERE'S HOW INSOMNIAC GAMES' CORE TEAM DID IT.

By Chris Edwards

A couple of years ago Insomniac began work on its first cross-platform console title, FUSE, which would be shipping on PS3 and Xbox 360. Insomniac's Core (engine and tools) team began rewriting its engine to support both consoles, with a focus on improving iteration time during the game production phase. Making the engine also run on PC became part of the rewrite, because it would allow us to cut iteration time by integrating our developer tools more tightly with the engine. So we took this task as an opportunity to rethink our tools architecture—and ended up moving our development tools into the web browser. As our engine director liked to point out, the World Wide Web was here to stay, and we should stop ignoring it.

WORLD WIDE WHAT? Developing a browser-based toolset was a new challenge for Insomniac's Core team. We had limited experience with browser technology and we faced a lot of unknown factors, so we settled on a number of constraints to mitigate the risks we were taking.

First, we decided to support only a single browser: Google Chrome. Most of us used Chrome anyway, and it seemed well supported, with a lot of debugging and diagnostic tools. Our focus had to be on developing an engine and tools for shipping FUSE, so we couldn't become bogged down in the subtle differences among browsers.

Next, we decided that the client and server code would run on the same machine. We knew that we would be dealing with large dataset, and we were concerned about the latency involved in moving that information from a remote server to a local client. This also enabled us to develop the server with a standard use case of only a single connected Insomniac. Although nothing we did precluded having multiple Insomniacs connected to the same server, we weren't going to focus on multi-user support at this stage.

The other notable constraint was that asset formats for FUSE were already being developed, and they were individual files on disk, managed by revision control. The tools would attempt to keep this paradigm unchanged.

INTRO TO LUNASERVER The tools UI runs in a web browser, and is mostly written in JavaScript and HTML. We have separate editors for working with levels, materials, animations, and visual scripts, just to name a few. Our engine is also embedded in the web browser as a plug-in, so the various editors can use it to visualize the 3D assets. The editors tend to be loosely coupled with the engine plug-in.

The web pages and engine plug-in are the clients that make up the UI used by Insomniacs. The clients communicate with a web server via HTTP to make changes to data, and they poll the web server to discover when they need to update their own views of the data. For this article I will mainly focus on the design and development of the web server. You can see a general overview of the different components of our toolset in **Figure 1**.

The backend for the tools is a C++ web server built on top of Mongoose **(Reference 3),** which we call LunaServer (we love our moon references at Insomniac). LunaServer provides access to files on disk, such as tools UI, scripts, source assets, and built (platform-dependent) targets.

We built LunaServer to provide a RESTful interface to its resources, though we are a little loose in our implementation of REST. REpresentational State Transfer (REST) is an architecture for defining web services **(Reference 1).** Its principles include a client-server architecture where the client manages state.

LunaServer does not directly monitor clients, though it does keep a running log of the changes that are made by clients (called a changelog). It also provides a scratch area where loosely coupled UI elements can share information to work together (called session data). Admittedly, this session data is a deviation from REST's design, but it does allow the server to provide some useful features.

LunaServer spawns a number of threads on startup, and those threads sleep until client requests start arriving. When requests arrive, the receiving thread often handles them immediately. Some requests need to be handled in a nonconcurrent fashion, and these are placed into queues that belong to various worker threads. The thread that received the request then sleeps until the worker thread handles the request, and then sends the results back to the client.

With LunaServer, clients can edit assets; access source asset files, built asset files (with dependency tracking for knowing when to build assets), and the revision control system; and they can automatically convert data when the source formats change. It also supports a unified undo/redo system.

WORKING WITH ASSETS ON THE WEB Source assets are individual JSON (JavaScript Object Notation) files on disk. For those unfamiliar with JSON, it's a rather commonly used text format that describes objects in terms of key/value pairs **(Reference 4)**. The JSON asset files contain various settings that are utilized in the game. Most of these

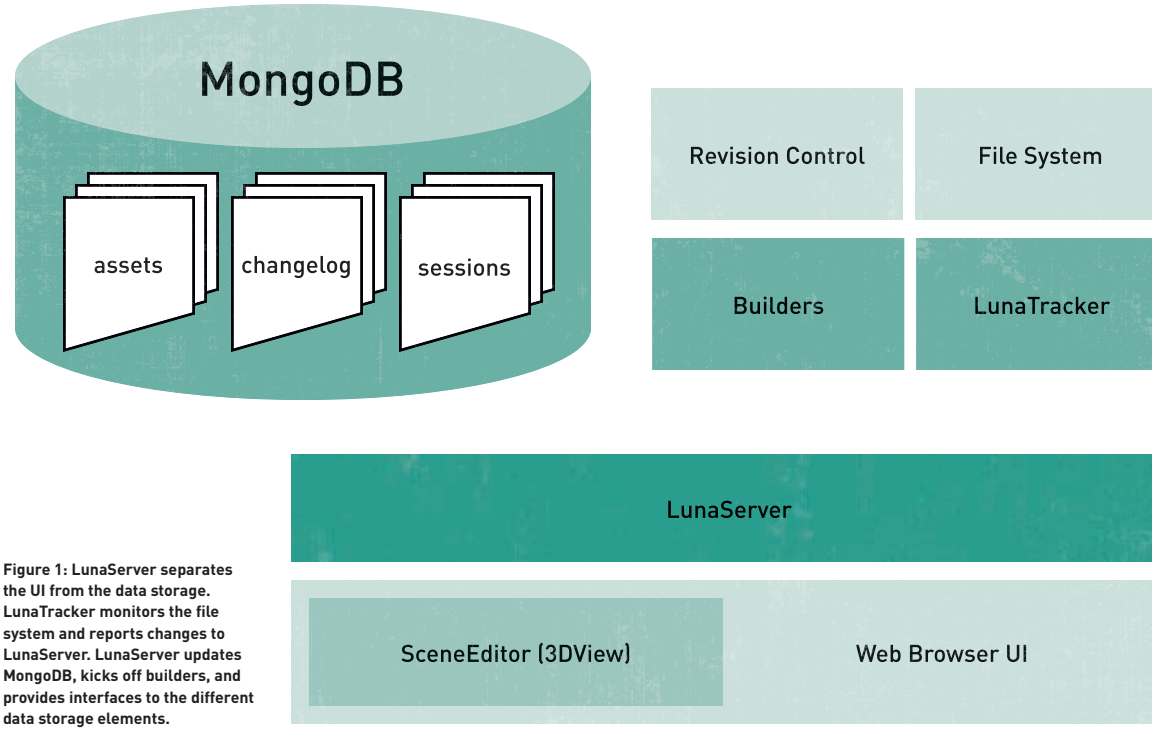


Figure 1: LunaServer separates the UI from the data storage. LunaTracker monitors the file system and reports changes to LunaServer. LunaServer updates MongoDB, kicks off builders, and provides interfaces to the different data storage elements.

settings are the properties that need to be tweaked during development of the game.

The asset files do not contain binary data, though they will have references to other files that store this type of information (such as geometry or textures). When an Insomniac wants to edit an asset, they check the text and binary files out of revision control to make edits locally. Once all the edits are completed and saved, the asset is submitted to the revision control system and available to the rest of the production team.

The file system is monitored by a separate application called LunaTracker. Changes to the file system are reported to LunaServer, which tracks all the dependencies among the files in a database (see **Figure 2**). As files change on disk, builders are automatically kicked off to convert source data into platform-dependent formats that are utilized in the game. LunaServer also accumulates all of the JSON asset files into a database.

We chose MongoDB **(Reference 5)** as our backing database. MongoDB is a NoSQL database that arranges BSON documents (Binary JSON documents, which are easily created from JSON) into collections. If you come from an SQL background, you can think of documents as rows and collections as tables. Because all of our asset data was already in JSON format, this was a simple transition for us to make. MongoDB has a robust language for querying and manipulating the data that it stores, all via JSON.

Insomniac's data is organized into collections by asset type. All the levels are in one collection, all the materials are in another, and so on. A single asset file on disk becomes a document (row) inside of the appropriate collection in MongoDB. These are the "live asset" collections and contain the most up-to-date changes to each asset.

When clients manipulate assets, they directly make those changes to the appropriate asset document in the database (see **Figure 3**). Those changes can then be saved to the backing JSON file and put under revision control. Also, assets can be edited outside of the tools (in a text editor, for instance, or by pulling a newer version from revision control). LunaServer will be notified of these changes by LunaTracker, and LunaServer will update the database.

To summarize, asset data is persistent between runs of the tools, and is eventually built into formats used by the game. Changes to an asset need to be reflected in all clients that are showing that asset. However, there is another class of data necessary for designing an editor, and we refer to that as session data.

Session data is shared among all the UI elements that make up a single editor. It includes things such as the current selection, which assets are currently open, and other preferences specific to that editor's view. Editors create sessions when they are opened, and destroy the sessions when they exit. When LunaServer is restarted, it purges any outstanding session data (to account for clients that crash without shutting down properly).

DESIGNING URLS FOR LUNASERVER When we first created LunaServer, it had a lot of very specialized URLs, and all communication was carried out via the POST method of HTTP. However, as the server has matured, it has become more and more generic. We have moved toward supporting fewer URLs, but making use of more of the available HTTP methods **(Reference 2)**. The combination of the URL and the request method type tell LunaServer how to act (see **Figure 4**).

The URLs are categorized into static content that maps directly to a file on disk, and dynamic content that requires some amount of processing or interacting with a database. We identify dynamic content with a URL that starts with "api." Dynamic content includes asset, session, and changelog items, which are the main focus of this article. There are also URLs that allow raw database access (mainly used for queries to locate assets based upon their

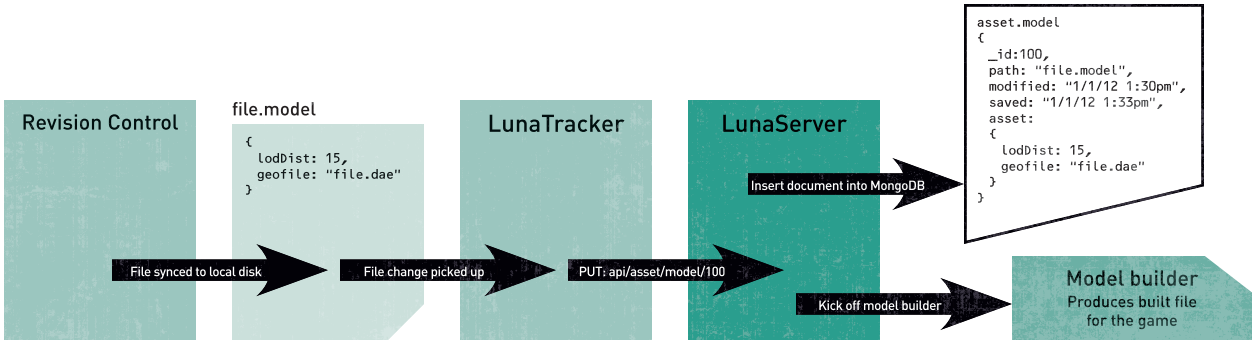


Figure 2: LunaTracker monitors the hard drive for changes and notifies LunaServer when files are updated. LunaServer keeps MongoDB in sync and kicks off builders.

file names or other attributes) and revision control access. See **Figure 1** for a table of URLs and operations.

In addition to dynamic data, LunaServer provides access to three main folders on the local disk, each designated by a different URL prefix. If the URL starts with "source," it refers to the root of our asset source tree; if it starts with "built," it refers to the directory populated by asset builder output; and if it begins with "user," it refers to the user preferences directory. Clients can read or write to any of these directories. Any other URLs are assumed to be part of the tools install (static HTML, JavaScript, images, etc. that make up the tools web pages themselves).

TEST CASE: OUR LEVEL EDITOR Let's take a closer look at Insomniac's level editor for an example of how the tools operate. When the level editor web page is opened, one of the first things it does is create a new session on the server. This is accomplished via a JavaScript class that manages communication with LunaServer.

This session is identified by a randomly generated ID, which is used to track the changes made by a particular editor so that LunaServer can provide undo/redo functionality to that session. The session ID is also a link between disparate pieces of UI that make up the same editor. For example, one of the main UI elements in the level editor is a 3D view of the level; the JavaScript code creates an instance of our engine plug-in and passes along the previously determined session ID, so that it is shared between the JavaScript 2D UI and the C++ 3D view.

The Insomniac can now open a level, which consists of searching for a level by name. There is a UI called the Vault that serves this purpose, and it just uses LunaServer's raw database access to query for levels by file path. Once you find the level you wish to edit in the Vault, JavaScript records that level asset into a list inside the session data and sends that change to LunaServer.

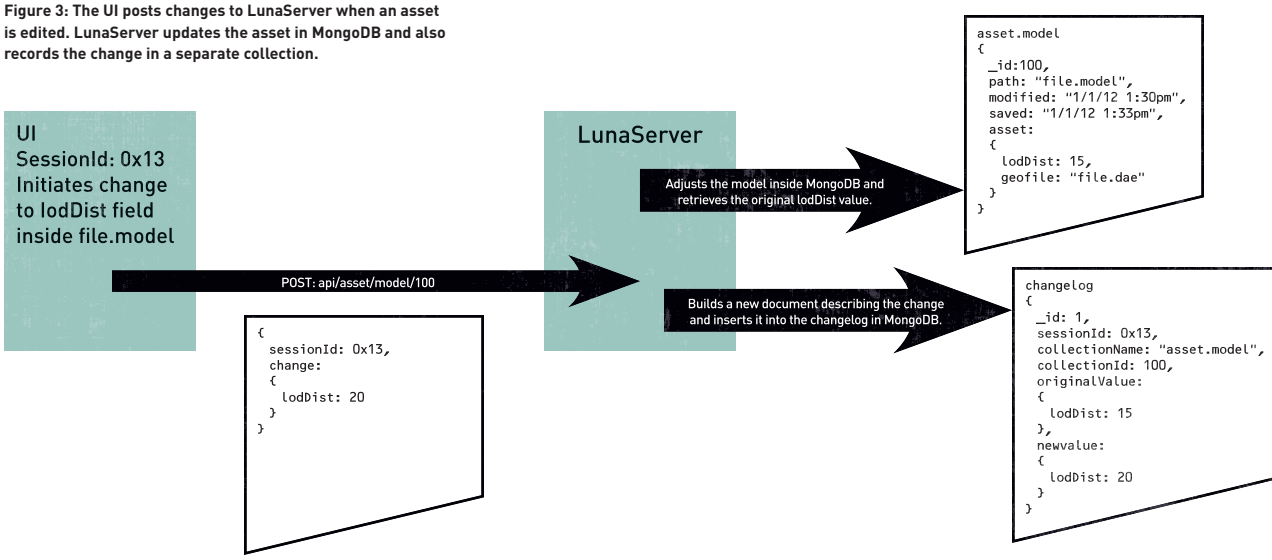
LunaServer updates the appropriate session document maintained in MongoDB by using MongoDB's findAndModify functionality. This operation allows you to edit fields in a document and fetch the previous values of those fields in a single, atomic call, so LunaServer now has the new value (in this case, the path to the level that was opened), *and* the previous value (there was no level file open in this session). These items are combined into a new document and added to the changelog collection. This is the basis for the change to propagate throughout the UI, and forms the building blocks of the undo system.

Meanwhile, the 3D view is busy polling LunaServer and will pick up that there was an entry added to the changelog. The 3D view will notice that the entry in the changelog matches the session ID that was passed to the view on launch, and so it will inspect the change and discover that there is now a level open, and it should be showing in the 3D view.

Both the 2D UI and the 3D view need to show information related to the currently loaded level, so they both will request the latest copy of the level from the server. The JavaScript code will use its copy of the level data to fill out an Outliner that shows a hierarchy of all the instances in the level, organized by name. The 3D view will walk over the list of instances in its copy of the level data and populate the 3D view with the visible elements of a level, including models, lights, and volumes.

The 3D view and 2D UI will continue polling the server for changes at about four times per second. They are on the lookout for any changes to their shared session data,

Figure 3: The UI posts changes to LunaServer when an asset is edited. LunaServer updates the asset in MongoDB and also records the change in a separate collection.



or the level asset that they each have a copy of. Any other changes can be ignored. As they handle or skip entries in the changelog, they track their position within the changelog using the ID of the last entry seen. This ID is sent to LunaServer the next time the changelog is polled, so the server only returns new changes that have occurred since the specified item. The UI is operational and responding to interactions by the Insomniac. Changes are reported to the server either as an update to the session (selection changes, for example), or as an update to the level asset (perhaps a model instance was moved to a new xyz location). All changes are marked with the session ID that originated the change, which is how LunaServer provides a unified undo system to all editors.

HOW THE UNDO/REDO WORKS Not all changes that are sent to LunaServer are required to be undoable—that's merely an option that the client can specify when they submit the change. If the client wants to make a particular change undoable, LunaServer will record the old value as well as the new value for any specified fields in the changelog. When an undo request is made, LunaServer searches the changelog for the most recent entry belonging to that session and uses the old field values to create a new change request.

This new change is executed much like a normal change: A findAndModify command updates the appropriate document and retrieves the previous values of the fields that were changed. This process constructs a new change document and inserts it in the changelog, referencing the same session that initiated the undo operation. Any editors that are open will pick up this event as they poll the changelog, which tells them what fields they need to update. Redo works in a similar fashion.

The undo/redo system also supports batching. This means that clients can issue a series of changes to session documents and one or more asset documents. Each change is recorded separately, but if they all share the same batch ID, they can be undone and redone as a single operation. If LunaServer attempts to undo (or redo) a particular operation, it will check to see if there are any other changes that match that item's batch ID. Any matches are also undone (or redone) automatically.

CHANGING DATA FORMATS It is worth mentioning that there is more than one way to fetch asset data from LunaServer: If the request comes via a URL that starts with "source," the asset file is fetched directly from disk and returned, and if the URL starts with "api/asset," the asset is fetched from MongoDB and returned. Usually these two documents match, but they can also diverge.

Over the course of development, data formats are constantly changing—sometimes because new features demand it, sometimes to improve performance, and sometimes just to satisfy programmer OCD. If you are just adding new fields to a data structure, you don't need to do any extra work—add the field and give it a reasonable default, and then you can start referencing the new field in the UI, builders, and game. If that field is missing from a particular asset, just assume the default. MongoDB does not need to know the document schemas, so there's nothing to update in the database.

However, we often find the need to rearrange existing data—for example, maybe we were storing angles in degrees and now we want to store radians. In this case, it is necessary to transform existing data so that it continues to work as you develop new UI, builders, and game code to consume the new format. Because this is a fairly common occurrence during development of a game, LunaServer provides a mechanism to transform data in this way.

Each of our asset types is assigned a version number, and we maintain a series of scripts that can convert from the previous version to the next. When we need to perform a large data rearrangement for a particular asset type, we increment the version number and write the conversion script. The version upgrades are performed in a lazy fashion—when an asset resource is requested, LunaServer checks that it is of the latest version before returning it. If it's not the latest version, LunaServer runs the converter and returns the updated asset.

LunaServer also writes the updated asset back into the database so that it doesn't need to be converted the next time that it is requested. The asset in the database no longer matches the saved version that is committed to revision control, but that typically doesn't matter. (I can hear the groans coming from the W3C on that one.) A GET request might actually make a change to the resource that is being requested, but the action is transparent to the client, and always the same, even if there are concurrent requests. The client and server code are packaged together for a tools release, so the client code always

Asset Operations

api/asset/{ASSET_TYPE}/{ASSET_ID}

PUT	Create/replace the specified asset
GET	Fetch an asset (*automatically converted to the latest format)
POST	Modify an asset (send partial changes to be applied)
DELETE	Remove an asset from the database

Session Operations

api/session/{SESSION_ID}

PUT	Create a new session
GET	Returns all data for a session
POST	Modify an existing session
DELETE	Remove a session

Undo/Redo Operations (session specific)

api/session/undo/{SESSION_ID}
api/session/redo/{SESSION_ID}

GET	Returns information on the undo/redo operations available for this session
POST	Undo or redo the most recent operation in the changelog for this session
DELETE	Prevent undo/redo operations beyond this point for this session

Changelog Operations

api/changelog

GET	Given a token, returns all changes since that token

Figure 4: LunaServer's URL schemes and corresponding operations.

expects to work against the latest asset definitions, and this mechanism makes sure that is the case. This automatic upgrade system means that it is not necessary to coordinate check-ins of all assets to run fixup scripts across the entire asset tree.

WORKING ON THE WEB Our web-based architecture has provided us with a number of benefits. For starters, anyone with sufficient motivation can build their own tools on top of LunaServer's infrastructure, and these tools can leverage the multitude of technologies and libraries available for working on the web. It's also easier to prototype new features, since refreshing a web page is a lot faster than recompiling source code.

Also, the complete decoupling of the client and server processes means that each one is insulated from crashes in the other. Because LunaServer writes changes into a database, it is relatively resilient to crashes, even if they are in the server itself. Restarting the server and the web page allows you to continue where you left off.

REFERENCES

1. "Architectural Styles and the Design of Network-based Software Architectures" - Roy Fielding - http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
2. Hypertext Transfer Protocol -- HTTP/1.1 - RFC 2616 Fielding, et al. - http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
3. Mongoose web server - http://code.google.com/p/mongoose/
4. Introducing JSON - http://www.json.org
5. MongoDB - http://www.mongodb.org/

Running the server and client on the same machine was very helpful in getting this architecture up and running, but if that machine shuts down improperly—specifically, if the database process shuts down improperly—the underlying MongoDB might get corrupted. That database can be rebuilt from the individual asset files on disk, but it can be a time-consuming process, particularly if everything needs to be rebuilt for the various platforms.

We also weren't quite prepared to develop on a platform that may change at any moment; every now and then we'll see an automatic update to Google Chrome that causes bugs in our tools. Most of the time, though, these just highlight deficiencies in our implementation that we should have addressed anyway.

For example, one recent update to Chrome started caching all of our interactions with LunaServer, which produced incorrect results in the editor and rapidly flooded hard drives with gigs of cached entries (including every poll of the changelog). As it turned out, LunaServer was not properly filling out the cache headers for requests to dynamic content, but it only became a problem after the new Chrome build started paying attention to those cache hints.

All in all, our new tools architecture has been successful in helping us build our first cross-platform console title. Automatic data upgrades and a unified undo system in particular have been extremely beneficial to Insomniacs as they worked on FUSE—and we know that we're just barely scratching the surface of what we can do with a web-based architecture.

***

Chris Edwards is a senior engine programmer at Insomniac Games, where he worked on FUSE as well as the RESISTANCE and RATCHET & CLANK franchises. He has been professionally developing artistic tools for over a decade.

REMIXING CLASSICS

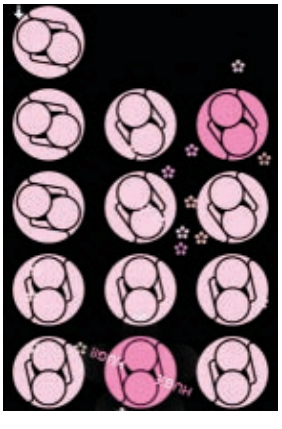
WE SPOKE WITH PARAPPA THE RAPPER CREATOR MASAYA MATSUURA ABOUT INDIES, THE FUTURE OF MUSIC GAMES, AND ADAPTING JAPAN'S MOBILE/SOCIAL SCENE

At the 2004 Game Developers Choice Awards, NanaOn-Sha founder and PARAPPA THE RAPPER creator Masaya Matsuura won the First Penguin award for his work integrating music and gameplay in PARAPPA, UMJAMMER LAMMY, and VIB-RIBBON. This award—now called the Pioneer award—was originally named for the first penguin in a colony who is brave enough to jump into water that might have predators. Now it has been 16 years since PARAPPA, and we thought we'd catch up with Matsuura and see what waters he's diving into next.

PATRICK MILLER: *We've seen some developers having trouble adapting to designing for mobile games (and touchscreens). How has that been going for you?*

MASAYA MATSUURA: We tried to play some of our existing successful titles with a touchscreen, and I was already satisfied with all of them; I couldn't think of anything new I wanted to do with them. So that was a problem for me, but it's changing now.

The important thing for me is that everyone playing a mobile game is chopping their game time. It's fragmented gameplay. Everyone wants to touch for a short time, then



do nothing, then touch. This kind of balance, watching, touching, talking, emailing, this kind of rolling mission should be designed into the game. It's very difficult.

PM: *Have you run into any particular challenges?*

MM: We're trying to figure out how we can be successful with a younger audience. Older smartphone users have credit cards, but many younger players won't have one. So right now, we have no way to appeal to younger audiences with smartphones, and we need to solve that problem.

The good thing [about mobile games] for the parents, is that kids are annoying all the time, and if the parents want the kids to concentrate and be calm, smartphone games are good for them. But is this a good way to provide the kids with a chance to play games? I'm very afraid that when kids grow up, the games they played on smartphones won't make for good memories. I think that communicating physically with their parents would be much more important experiences for the kids. No one denies this idea, but unfortunately, that kind of thing is not common sense right now.

PM: *Interesting. Have you seen any games on newer platforms that you think would make for good childhood memories?*

MM: What was that one game, the first one on Google Plus? It was a puzzle game, a very simple one but very sophisticated. It requires a brain. Calculating. That was a very good game for me, and I was addicted. The last time I was addicted to a simple graphics puzzle game was 20 years ago.

PM: *What kind of opportunities have you found in Japan's mobile/social market?*

MM: That's a very big issue for us right now. Of course, we've

"I'm very afraid that when kids grow up, the games they played on smartphones won't make for good memories."

been trying to make a social-based *something*, but social gaming in Japan has had some big trouble with the kompu-gacha schemes. I don't care so much about this kind of issue, because new things always have problems and we can't avoid this kind of thing. But as a game developer, I think it's very important to make the right tweaks for the market and the industry. Since we don't have a border between countries and territories with social games, I think we can tune this kind of era into a more sophisticated one.

PM: ***Many game devs in the U.S. look down on social games; is the perception among Japanese devs similar?***
MM: Yes.

PM: ***Do you think there's more potential there?***
MM: Yes, of course. Right now, the important thing is sucking money from people without any fun; this is an interesting tendency, but it's not the straightforward future, I think. It's a little bit of a bend into some unknown area. If that kind of direction is the center, I'm okay, but I think we have to find a different direction for social games.

We're trying to make something right now, and making a game that makes money and is fun is an ongoing challenge for us. And, you know, with the earthquake last year, it's changed the way we think of our economy, and the way we think of new products and trends. The past trends were always about expanding something, but now, our paradigm has shifted and we're not thinking about expanding any more.

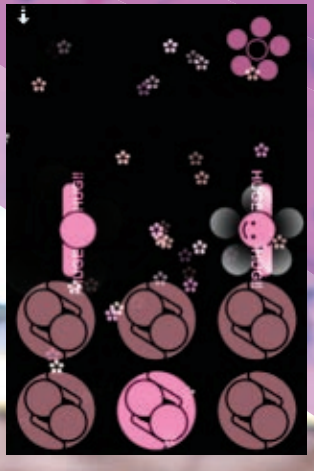
PM: ***In the past, you've talked about how you wanted music games to be more than just the GUITAR HERO/BEMANI-style rhythm game. Considering that genre has kind of imploded by now, where do you see music games going?***
MM: I saw one guy who did an interesting TED talk, named Eric Whitacre (http://ericwhitacre.com/). He was just a songwriter and composer, and what he did was he posted a video of himself conducting on YouTube with no sound. His fans posted recordings of their own singing while watching his conducting, and finally 2,000 people posted according to his conducting, and he decided to mix all the videos together to make a big choir that we had never seen before.

This kind of new experience, new interaction, new unsynchronized collaboration has big possibilities to make new music experiences. For example, singing is the best way to collaborate with each other, because everyone is good at singing *something* in karaoke. It's not a *game* right now, but I wouldn't want the game system to evaluate the music from the audience; we could instead use the YouTube viewer number as a kind of evaluation, or something like that.

PM: ***What have you been playing lately?***
MM: I haven't played many games recently. I'm playing our recent product; we released a 3DS download game for the kids—it's a violin lesson game. I thought that no one played 3DS download games, but the publisher said that the download number is increasing rapidly. It's very interesting. I'm looking forward to how we can have another chance to expand something like that.

What we did was very interesting; I always think that music games are not just about reproducing perfect playing. I really love to see the ugly playing. In this violin lesson case, we hired the actual violin player, and they played a perfect, attractive violin playing, and we recorded that, and they played the ugly playing too, so that if the player plays poorly, the ugly sound comes out.

PM: ***We had a postmortem from Q-Games on PIXELJUNK 4AM, and in it they talked about how they had a hard time balancing the musicians' desire to have the music sound good, with the designers' desire to build a proper game.***
MM: A very important point of the attractiveness of 4AM is that their focus is on a digital



type of music. This kind of music is always very abstract, and kind of hard to evaluate whether it's "good" or not... In our game's case, we would show a very explicit result, so all the players can detect how and where they'd played it poorly. This is very practical, and a very different direction.

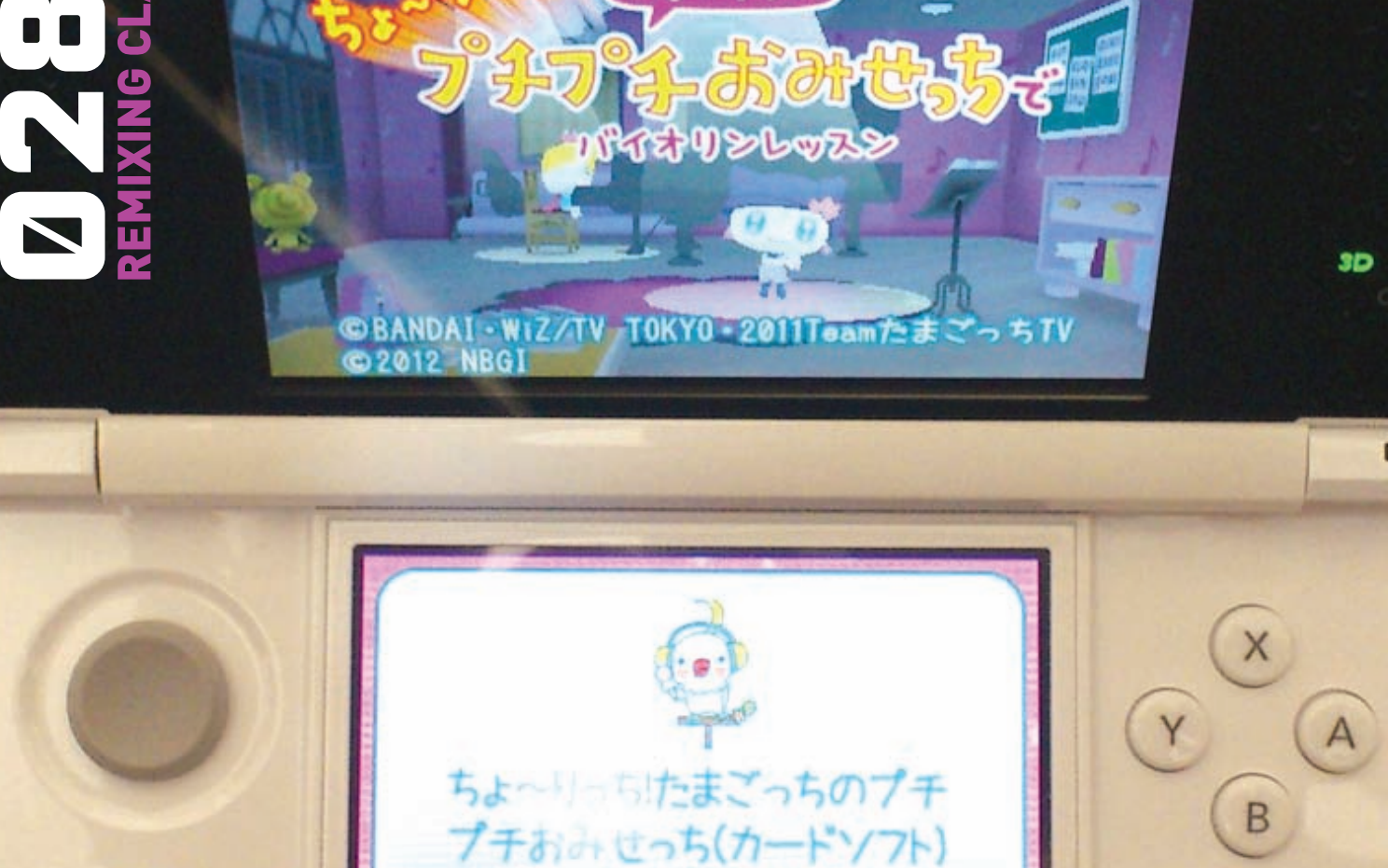
PM: ***Indies in Western markets are doing well on the PC, but in Japan, the PC games have always been kind of a highly specialized niche. Is the Japanese industry perception of the PC market changing?***
MM: I really hope that Windows 8 is a chance for us, and I'm keen to see the Microsoft Surface. I think that already many people dig the possibilities of a touchscreen device, but having a keyboard and tablet could be a completely different interaction. I am also keen to see how they support the camera; they already have the Kinect experience, and if they can minimize the technology and put it on the Surface, we'll have many possibilities as developers. Of course, Microsoft hasn't told me anything.

The fever for Apple products interests me; some parts, to me, are understandable, and some aren't. I've had a long history with Apple products. My first was an Apple II, so I know that Apple can be a dreamy company, but also a disappointment. The Windows environment doesn't have the same kind of thing. Its focus is too strict for me. With the Surface, Microsoft may have to change their strategy, which could be a chance for us. It's very risky, so far, for us to make PC games, because on the PC it's very hard to sell products directly to the customer. Already, we have many Internet-based, web-based game services that support Windows, so it's very competitive, and it's very hard to understand the business battlefield.

PM: ***You know, I didn't expect that the head of such a creative studio would speak so much about how technology informs his work. What's the next major tech advancement you're looking forward to?***
MM: That's a good question. Recently, I've been really interested in AI. Maybe AI has a big possibility to expand our industry to more attractive and interesting ends. I have a Facebook friend that posts his AI things and ideas on Facebook, and they look very interesting to me right now. I think it'll be the next big technology for games, but I'm not sure about how we could use it in practical ways for games yet.

Just a few years ago, I didn't have so much intention for AI; I thought all it did was handle enemies' activities in RPGs. But I think that was a misunderstanding.

The first time I made a music-based game, well, those games don't change. I really wanted to expand the music game possibilities to wider and more attractive areas, but that required AI tech. Music is a very strange



expression; it unifies and synchronizes human activities with almost nothing, and people synchronize their emotions and expressions to make something unified. I think that AI is needed to work with this kind of primitive activity.

PM: *Last time [GD Mag editor emeritus] Brandon Sheffield spoke with you, you were rather outspoken about how focused the game industry was on making violent games. What's your take on game violence now?*

MM: I changed my mind a little bit after my last talk with Brandon. He said "Mr. Matsuura, you said violent games should be decreasing in the market, but I still love them!" I understand that violence is irresistible, but from the Japanese stance in the game industry, I want to appeal to more, better ways to expand a game's expression with more interactions. Violent games are okay, but let's expand them.

I had a very good talk with one very well-known game developer in Japan, Masanobu Endo-san, well known as the programmer for XEVIOUS. More than 25 years ago, I was addicted to XEVIOUS, and I'd just sit there with a stack of 100-yen coins.

My biggest question with XEVIOUS was about the music. XEVIOUS started with (staccato) *"ja ja ja ja, ja ja ja ja"*—a very warlike sound that encourages the player to play the game. But when the game started, in the game, you hear a very strange, sensitive, ostinato sound, the *"pirororiro, rirororiro."* It kind of, hmm, sounds like the enemy is laughing, or a bug in the computer, or something.

I didn't understand why this kind of sound comes out, and I didn't understand whether the sound came from the enemy, or my plane, but I couldn't detect it, so I asked Endo-san directly, and he was surprised. "No one has ever asked this kind of question! We first programmed XEVIOUS without any music, but it was too scary, so we had to add some kind of music." So that was the reason to add the music for the game.

I was surprised, because I never felt that game looked scary. It's very smart, very practical, very algorithmic. But Endo-san's feeling was completely opposite from mine.

The feeling of beating someone else is very attractive to game players, but if we can produce games that leave other impressions, like music, or graphics, or something to express something higher... Well, I think that many gameplay developers don't care about this kind of thing.

PM: *So what'd you think when you found out that PARAPPA was going to be in PLAYSTATION ALL-STARS BATTLE ROYALE?*

MM: Well, in that case, I can do nothing. [Sony] decided to use PARAPPA for their game ideas, and I checked the gameplay, and for me, that kind of expression doesn't look so violent. It's not real characters, real graphics fighting each other. It doesn't look so violent. I don't know. It's a sense.

PM: *I'm under the impression that, historically, Japanese developers have been less likely to share game development technology and techniques with each other than American devs. Is this still the case?*

MM: It's getting changed. Around me, at least, many developers have been using Unity, especially independent small developers. With Unity, it's easier to share the experience with each other, so individual small developers can collaborate with each other more easily. My feeling is, the small developers have more chances to collaborate with each other, because there are so many opportunities and so many different platforms that every time we have to make a joint venture with other teams.

PM: *We're also seeing a lot of established devs in the U.S. start their own independent studios; is there a similar trend in Japan?*

MM: No, actually. This is a very important point; in Japan, the big trend comes out for social games, and the big two or three or four social game companies sucked up all the people—experienced people and new developers.

PM: *It has been 16 years since PARAPPA THE RAPPER came out. Do you think it has influenced younger devs to think differently about music and game mechanics?*

MM: Yesterday, I had a chance to chat with a Japanese music and game journalist, and she said she had many chances to interview the younger musicians, and she always asked them what games they played in childhood, and many of them answered with our games. I really want to see how new developers and younger people try to do new things. I really liked SOUND SHAPES; FEZ is also interesting.

GROW IN THE GAME INDUSTRY

- Reference industry news and features
- Consult your digital counselor
- Play student games and join the forum

VISIT YOUR YEAR ROUND MENTOR
AT GAMECAREERGUIDE.COM

- Examine tutorials and exclusive features
- Check out the Annual Salary Survey
- Reference the premier Game School Directory

DOWNLOAD YOUR FREE DIGITAL COPY AT
GAMECAREERGUIDE.COM

- Learn from the pros
- Attend deep-dive sessions with Q&A
- Connect with your game making peers

VISIT GDCONF.COM FOR INFO ABOUT THE NEXT SEMINAR
AT GDC 2013

-- FOR PROFESSIONALS --

- Search for active junior to senior level jobs from 100+ leading game companies
- Upload your resume and get in front of direct hiring managers
- Organize your career research with your personalized Job Seeker dashboard

VISIT GAMASUTRA.COM/JOBS TO ADVANCE
YOUR GAME CAREER

UBM Tech

PAPO & YO

There's a little video floating around of that moment when PAPO & YO first went live for pre-orders on the PlayStation Network. Our creative director and co-founder Vander Caballero calmly placed his pre-order, and then walked into the TV room to announce it to the rest of the team. There were high-fives all around, and the excitement in the room was palpable—it was a brand-new sense of accomplishment for the project being one step closer to launch. The next day, our live-action cinematic launch trailer had over 50,000 hits, and PAPO & YO was trending worldwide on Twitter. Developing PAPO & YO has been quite the ride. Minority Media is a small band of triple-A developers—used to working in large teams with abundant resources—who went indie to create games, taking advantage of digital delivery systems. And while PAPO & YO got its fair share of media detractors, it also garnered a whole lot of deeply personal press coverage, with journalists and fans telling their own stories in response. It was these heartfelt and profound messages that told us we did something good.

WHAT WENT **RIGHT**

1 CHOOSING A STORY THAT MATTERED PAPO & YO is an emotional adventure filled with puzzles. You play as Quico, a young boy hiding in his closet from thunderous footsteps outside. Through his imagination, Quico escapes to a magical world of colorful favelas, with houses that can sprout legs and fly, and a robot friend who can act as a jetpack. Quico is soon introduced to Monster, who is both friend and foe. Monster can be helpful and kind—Quico can use his belly as a trampoline—but when Monster eats a frog, he turns into an uncontrollable fiery demon. It's an autobiographical metaphor of Vander's childhood with his alcoholic, abusive father in Colombia.

In many ways, being able to tell that one true personal story was a unifying factor throughout the whole development process. First, it's much easier to stay on course creatively if there's one vision, and in the end, it shaped the experience as if we were working on a film with a director, including the emotional highs and lows. Deferring to Vander helped gel the team and design, and helped filter creative ideas effectively, because he could say, "That's not what it was like in real life. My life."

For instance, Monster's design was changed late in the project because Vander found the old one too nice and likable. It did not evoke the same emotions he had when seeing his father, whom he describes as "distant and scary, but at the same time protective." Some fans were upset at the change when we first showed it, but ultimately Vander is the one who knows what it should feel like. A week after release, a fan tweeted, "It would be so sad, being afraid to hang out with your dad." When Vander read this, he went really quiet and eventually said, "It is." People understood.

Secondly, the team's personal buy-in was a big motivator. Vander was fighting to change the industry with more emotional stories by offering up his own story to start. Being a part of that meant being able to make a difference in creating something that was touching and entirely unique with an artistic depth, but since the game is autobiographical in nature, we encountered some unexpected challenges, like planning and implementing the right emotional curves for how we wanted players to feel, striving to evoke empathy through interaction. None of us had ever worked on a project this profound, and it was important for us to convey the story that Vander wanted to tell.

We found that PAPO & YO made a difference in people's lives, but we never imagined the quantity and kind of meaningful fan mail that we would receive, like children of alcoholics finding healing or gaining the confidence to talk about what happened to them. Or the single dad who wrote us a Facebook message to say that he doesn't want to appear like Monster when he loses his temper and yells at his five-year-old son, so he'll handle tough situations differently.

2 CHOOSING THE RIGHT PEOPLE In building a team from the ground up, it was important to bring on people that we could both trust and work with well, especially when it was going to be close quarters and long days. Aside from having greatly experienced people on the team, there was good chemistry and a great blend of personalities, so often, it did (and still does) feel like going to work with friends. Many of us had also worked together previously, so we were able to make design decisions that played to our strengths. For example, having an experienced animation programmer allowed us the opportunity to use some great character animation with a third-person camera. Vander has often said in interviews that having this kind of close-knit core developer team is like having a small functional family, especially when there are times when you see your colleagues more than your own family.

The experience of the team was crucial in delivering such an experimental game. The game design was by no means final when we started production, and we had a very organic approach to building it. We modified and redid puzzles several times in response to focus tests, and new game mechanics were introduced as the story evolved. For instance, throwing soccer balls back and forth with Monster was a very late addition to the game, when we realized people did not have enough occasions to interact with him in his friendly state, and were not getting as attached to him as we wanted them to.

We were constantly course-correcting to get the game to evoke the emotions we wanted it to, and the high level of experience of the team helped them prioritize tasks to deliver things on time even when everything was constantly changing around them. It was still hard, but being able to roll with it made the end product better. In the same vein, everyone felt a higher level of ownership and accountability for their part in the game, which consisted of multiple roles given the size of the team.

3) EXCELLENT AUDIO OUTSOURCING STUDIO We outsourced our music and audio work to composer/sound designer Brian D'Oliveira and his team at La Hacienda Creative, but it didn't actually feel like outsourcing. Instead, we were collaborating with them just like they were part of the core team, because they were just as dedicated to PAPO & YO as we were.

DEVELOPER Minority Media Inc.
PUBLISHER Minority Media Inc., Sony (Pub Fund)
RELEASE DATE August 14, 2012
PLATFORMS PlayStation
 # OF DEVELOPERS 12 in the core team and approximately 21 contractors
LENGTH OF DEVELOPMENT 18 months
BUDGET $1.5m
DEVELOPMENT TOOLS Unreal Engine 3, Visual Studio, 3ds Max


FACE + BODY COVERED W/ POWDERED ASHES AND BLACK CHARCOAL
SMEARED BY HAND
FOUND OBJECTS MADE INTO BRACELETS + NECKLACE. TWINES, THREAD, STRAPS, PLASTIC, METAL, ETC.
BURLAP SACK BAG
CHARCOAL

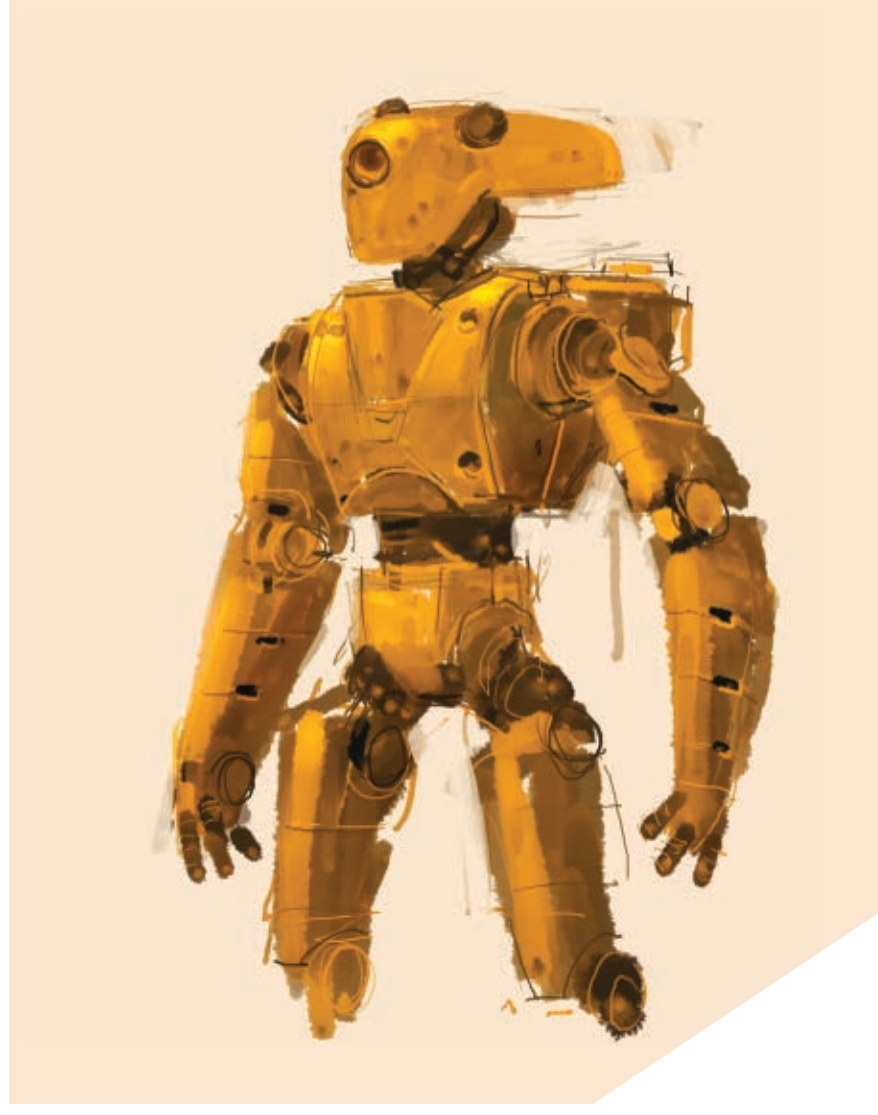
On our side, we had a passionate programmer (Frédéric Hamel) who loves audio, and on the La Hacienda side, they had a passionate composer. This passion was how we turned an otherwise run-of-the-mill outsourcing contract into a real collaboration. It was by chance that Vander and Brian happened to meet at a party, but somehow, things just clicked for Brian working on PAPO & YO, having grown up in Venezuela and experiencing a troubled childhood himself. For him, it also became an immensely personal project, and his devotion shows in his work.

La Hacienda normally provides music and sound services to film and TV studios, but they wanted to try their hand at working on a video game. Even though they were new to game music, they understood the opportunities offered by an interactive medium. Working with Audiokinetic's Wwise audio middleware allowed them to create a dynamic and ever-changing sound environment, which added a lot to the feel and authenticity of the game. Brian didn't want any canned sounds or sampled virtual instruments, so all of the game's music and audio is either captured from the field or recorded in the La Hacienda studio. Brian did an expedition to South America to record field ambiance, sometimes waking up in the middle of the night to capture things like frog sounds and monkeys howling.

In some levels of PAPO & YO, these types of sounds are woven into audio designs of 30-40 layers of music and 60-70 layers of sound effects, all differently triggered by various actions and looping dynamically. Brian plays about 80% of the instruments in the game, having learned about 15, with some sourced from all over the world. And on top of that, he handcrafted some himself. As programmer Antonio Maiorano said, "That guy is so talented, he could string floss on a piece of wood and play it."

To facilitate La Hacienda's first foray into video game sound design, our programmer Frédéric spent a great deal of time working with the audio and audio team, guiding them in how to make the most of their tools and providing them with the features they needed. Later on, we paired a junior audio engineer with a more senior one, which also helped immensely in streamlining processes. Ultimately, the whole soundtrack endeavor was driven by passion for making it the best that it could be.

Hearing each new piece of the soundtrack in-game was incredibly motivating for us, and made us see our work in a whole new way. Adding that dimension was an amazing morale booster and helped immensely in the last sprint toward the finish line.



4) UNREAL ENGINE 3 Unreal was the best option that was available to us, as most of the team had already shipped games with it, and knew its strengths and weaknesses very well. This helped get the team up and running in no time. We were setting up the company in November 2010 and in June 2011 presented a playable demo on PS3 at E3 that won six awards, including two from GameSpot for Best Adventure Game and Best Downloadable Game; Best Puzzle Game of E3 awards from IGN and *GamePro*; Gamer Theory Media Best's Downloadable Only Game and Bitmob's "This is Kind of Upsetting" Non-Award Award. We wanted to avoid building an engine from scratch, and UE3 was the right fit given the scope of the game and small team size.

Vander loves Unreal. "It's the best level editor I've ever used in my life," he said, "You're able to do so much, so crazy fast!" Unreal's editor was a great asset that allowed us to quickly start producing levels, art, and cinematics, and have them run on the PS3 without much effort. We were able to direct our efforts to creating unique gameplay mechanics instead of platform support and other technical details.

5) OPEN, INCLUSIVE CREATIVE PROCESS PAPO & YO was guided by a strong vision, and flavored by the personalities of the people who worked on it. Vander definitely had specific mechanics and intentions in mind when designing the game, but he left many of the particulars to be filled in by others on the team in the way they liked best. He would paint the mechanics with broad strokes, for instance, asking for a way to have fun with Monster by throwing a soccer ball back and forth with him. A programmer and an animator then experimented with various ways of getting that to happen and be fun, figuring out how it would be controlled, how Monster reacts, and how you can catch the ball in midair if you run and jump in a certain direction.

In general, we really enjoyed this creative freedom, which added a whole new value and depth to the design of the final product. Even though there were many ideas that didn't make it into the game, knowing that your suggestions would at least be considered made a big difference, especially for being so personally invested in the project.

Many key game elements, like the chalk outlines and upside-down hint boxes, came from the team, and in the end, each person can feel a greater sense of personal accomplishment toward the group effort.

WHAT WENT **WRONG**

1) SCOPE PAPO & YO was definitely an ambitious and challenging project for such a small team to make, especially as we set our own expectations quite high, coming from big studios. And while the budget for PAPO & YO was approximately a mere 3% of most triple-A titles, we were still aiming high, not only for the end product, but also for ourselves. The reality was that budget constraints couldn't afford us the freedoms that we were used to, like other people from alternate studio locations jumping on to help finish the game.

We initially planned for much more variety in settings, going from the favela to a tree village, an electric plant, and ancient ruins inside a mountain. As we progressed, we realized that this was more work than we could handle, and we had to consolidate the settings by focusing on the favela a lot more, differentiating scenes through lighting, weather, and changes in texture and color, without completely changing the background.

Even though the game has a wide variety of mechanics as it stands (new mechanics are introduced throughout the game in nearly every puzzle), we had plans for several more that we had to remove, like climbing on Monster's back to control his movement, or making Quico's robot Lula interact with electricity and water.

Keeping to the game's story was both a blessing and a curse. While it helped to tailor the creative direction, it also had a huge impact on iteration, level design, and quantity of assets because certain things just couldn't be cut. We did reduce the scope of the game compared to original plans, but we had a story to tell that we did not want to compromise, even if that meant having less time to polish certain areas.

2) BUGS Despite our best efforts at bug hunting and hiring a very talented QA company, we unfortunately shipped with a number of remaining bugs in our first release of the game, and while some of these have been addressed in a day-one patch, we're still in the process of fixing some others. We fixed all the known bugs in the game before releasing, so this caught us by surprise—the remaining ones were simply not discovered before release.

One issue with debugging a complex puzzle game is that many bug-prone situations will only arise with players who don't know the solution to the level. Our testers quickly discovered how to solve the puzzles, and while they made every effort to break the game by making the wrong decisions, it is hard to match the range of possible actions of a genuinely confused or lost player. On our new projects, we're holding frequent playtest sessions, leaving more time for polish, and securing a larger testing team.

Another issue is that our game has heavily scripted levels, with lots of unique logic in each level. Every puzzle is completely different from the next. This meant heavy use of visual scripting (Unreal Kismet), which was invaluable for quick prototyping and iteration, but made things harder to debug in the end.

Visual scripting languages are very easy to prototype with, but require a lot of discipline to keep clean and maintainable. When left unchecked, the project can devolve into quite a literal variation on the spaghetti code problem, and lots of effort needs to be invested in rewiring everything to make sure it is bug-free.

Another issue is that debugging tools are typically less powerful and evolved than the ones available for more widespread programming languages. One lesson we will draw from this is the need to implement prototypes instead of basing the final level directly on the first draft, which allows a programmer the opportunity to make the level script safer and more robust.

3) PROBLEMATIC CINEMATIC PRODUCTION PIPELINE Our cinematic production pipeline was one of the kinks we wished we could have worked out prior to starting, but unfortunately, that luxury is now reserved for our next title.

We wanted our game to be innovative in several respects, and particularly in the way the story was told. The challenge of telling such a complex story interactively is in the balance between how much players are playing, and how much they're watching. Vander's direction was to create empathy through interaction; the game doesn't tell you that Monster cannot resist frogs or that he gets mad when he eats one, so you must experience that firsthand. But we realized that a number of important parts of the story could only be told through cutscenes, so we wanted them to feel tightly integrated with the game, as if they took place in the same setting as the gameplay.

Unfortunately, this meant they had to be developed while the areas where they took place were still evolving—indeed, we iterated on many of the levels until the very end. This meant we had to make multiple changes to the cinematic direction, in and out points, animation, and subsequently music—all causing major headaches, for everyone.

We tried to improve the situation in various ways by developing new ways of integrating cinematics into the game to resolve some of these interdependencies, but since we did not have time to go back and redo previously authored cinematics, this meant maintaining several pipelines for doing the same thing, which did not simplify the process. In the future, we'll be much more careful about having a stable pipeline and better-defined constraints for each scene so that our cinematics production doesn't suffer as much from our very dynamic level-design process.

4) ORG CHART HIERARCHY ISSUES In 2010, the company was essentially four people, and we gradually ramped up after our successful E3 showing. We took time to carefully select new team members to make sure they would be a good fit to work with us, which meant that not every position was filled as early as it should have been. For example, we did not have a lead artist at the beginning, and that put a great deal of pressure on our environment artist to steer the art department.

Even once we got every member of our dream team on board, we still had to endure some growing pains. We worked for months as an even smaller group with no defined process before eventually introducing some hierarchy in the middle of the development process. This meant we had to make sure everyone was on board before making a decision, or else someone could feel bypassed or that his authority was being undermined.

We also hired a small number of interns to help move the project along faster toward the end. They did help, but we were not always able to make the most efficient use of their time, as we did not have a solid mentoring and leadership structure in place, and we were very busy producing the game itself. We will still be looking at potential interns for future projects, but with more careful planning and more structure in place.

5) OUTSOURCING Being a small company with limited funding, we could not afford to hire specialists of every single discipline needed to get a full game off the ground. We covered many of our needs by hiring versatile people who helped on several things at the same time, but for some other things we had to resort to outsourcing. This allowed us to afford the services of very talented people we could not have otherwise worked with, but it's always harder to communicate and iterate with someone when they're not in-house.

One particular aspect that fell victim to budget restraints was character modeling. Without an abundance of start-up resources, character modeling was contracted to some of Vander's talented friends who wanted to help with the project (and did so, immensely), but the factors of different time zones, schedules, subsequent breakdowns in communication, and pipeline delays contributed to a very long process that affected internal workflows.

Had we been able to throw money at the problem, we would have brought in experienced, full-time, in-house character modelers, but with our current resources, outsourcing is still an attractive option. The lesson to learn here is really in how we approach outsourcing from the outset. Having the right attitude, approach, combination of people, and most importantly, passion, can make all the difference in transforming outsourcing into a real collaborative experience.

CONCLUSION PAPO & YO was Minority's first game, and after successfully shipping it, we feel (as clichéd as it sounds) that dreams do come true. Highlights for us have come in all shapes and sizes, like the positive review in The New York Times, or the GameStop store manager who covered someone's balance of $2.04 just because he was buying PAPO & YO, or the superfan who's played over 70 times and is on a mission to have a Lula birthday cake for his daughter (also a superfan). It's a new kind of positive attention that we never anticipated, and we hope our fans see how much we appreciate it on a frequent enough basis.

Now that we've been through the first title, we're in much better shape to handle the next projects. We'll be able to start with a fully functional team from the start, rather than ramping it up over the course of the production, and we have added new senior members to our team, who can help us work out the kinks in outsourcing and hierarchy structures. We'll allow more time for polish, accounting for the time needed for the submissions process and related marketing efforts. We'll also be more mindful of the debugging time required for puzzle- and exploration-heavy games, while developing our workflow to ensure that what we build is more robust from the start.

We have an incredible sense of pride and a new level of achievement for having pulled off what we did. Undoubtedly, the bad reviews can be humbling. But nobody on the team had ever read fan mail for changing someone's life until now, and even if that's not weighed in Metacritic, we'll happily take that instead.

***

This postmortem was co-written by Deborah Chantson, community manager/writer for Minority Media, and Julien Barnoin, lead programmer and co-founder for Minority.

ARE PLAYERS PUSHING YOUR GAME IN DIRECTIONS YOU NEVER INTENDED IT TO GO? IN THIS ARTICLE, NILS PIHL EXPLORES HOW MULTIPLAYER GAMES ENCOURAGE GOOD AND BAD PLAYER BEHAVIORS.

WHEN PLAYERS MAKE THE RULES: ON MEMES AND THE METAGAME

HOW DO BAD PLAYER BEHAVIORS SPREAD, AND WHAT CAN YOU DO TO HELP STOP THEM?

An interesting thing about games is that the player always helps design them. No matter how simple or complex the game is, there is always room for our own creative input. We add new rules, new contexts, new narratives, and new measures of success, and we choose which of the original characteristics of the game we want to interact with. Games are much like books in this manner, and we will often find the most interesting things about games between the lines of the author's instructions.

When designing games, this is both a blessing and curse. How the player behaves within the context of the game has an enormous impact on how enjoyable the game will be for the player, and game designers often find themselves struggling with how to encourage the players to play in a way that will be rewarding. Managing the expectations and behaviors of the player is a daunting task, but one of tremendous importance. Games that are well developed in every sense can still fall short to an unhealthy in-game culture. The game is only as good as the players.

So how does one manage, or even anticipate, how players might behave within the game? To understand how, we will first have to gain a rudimentary understanding of behavior itself.

UNDERSTANDING PLAYER BEHAVIOR Behavior is the way someone acts in response to a particular situation or stimulus. It's important here that we don't confuse behavior, which is a model for describing someone's actions, with the actions themselves.

"I am going to sleep" is a good example of what an action might be, and "I will go to sleep after this TV show, even if I'm tired now" shows us what a behavior would be. Whereas an action could be described as a data point, behavior is a graph attempting to make sense of the data. If we have a good model for someone's behavior, we can extrapolate, derive, and experiment.

Some behaviors are particularly successful at achieving things that are good for us, while other behaviors can be wasteful, detrimental, and destructive to us. This helps us rationally choose some of our behaviors, and avoid others. We brush our teeth with toothpaste to keep our teeth white, but only very few will make the leap and attempt to brush their teeth with bleach.

But we aren't that good at avoiding destructive behavior. We routinely engage in behaviors that are bad for us, even when we are very well aware of the negative effects the behavior might have. Consider things like smoking, gambling, unprotected sex, and speeding. How can we explain these irrational behaviors?

The key lies in understanding that the roles we ourselves play in determining what behaviors are prevalent in our culture are fairly limited. Behaviors and ideas seem to have a life of their own.

The famous evolutionary biologist Richard Dawkins coined the term "meme" in his book *The Selfish Gene*, in an attempt to better explain what generates culture. As a game enthusiast, you may remember that Huizinga thought that our desire to play was what generates culture, but that is not a complete model. Dawkins found a way to explain how the things that play generated got to be so popular—how they could move from isolated behaviors into the realm of culture.

A meme is a chunk of behavioral code—a behavioral gene—that can get copied from one individual to another. Memes are the building blocks of behavior. The words and gestures we use, the phrases we choose, the way we fold our laundry, the way we get our hair cut, these are all memes, and they are ideas that can be observed, copied, and mutated.

In *The Selfish Gene*, Dawkins explains to us how natural selection acts on the genes, rather than the individuals. This gene-centric view of evolution has helped shed a lot

of light on some of the more peculiar aspects of biology. The theory shows us that certain genes are more successful at being reproduced than others. The more successful genes will outperform the less successful genes and, with time, we will see more of the successful genes than we will of the less successful genes.

This simple process generates organisms that are very well adapted to the environment they live in. The gene-centric view of evolution has helped explain things like diseases and cancer, and is now a more useful model than Charles Darwin's own model of evolution. The individual is a machine built by and for genes, with the sole purpose of replicating genes.

Although they obviously do not exist in any physical sense of the word, imagining that same process of natural selection on ideas helps us understand why bad ideas spread. The survival fitness of a meme is not determined by the effect it has on its host, but rather by how well it propagates to other hosts. Memes seemingly hijack our brains and make us into machines for spreading more memes. Behaviors and ideas have a viral life of their own, just like our genes do, and we are the sometimes-unfortunate hosts of this second replicator. Memes spread through observation (even involuntary observation), and they copy themselves and mutate into new, potentially viral, strains of ideas.

When designing games, we are not completely at the mercy of these memes; there are ways for us to guide the evolutionary process of memes to a place we want. Although we can't choose which particular memes will emerge from selection, we can alter the environment of selection itself. By carefully designing the environment that the memes will populate, we can make some predictions about what will emerge.

A fitness function is a model for evaluating the fitness of an entity. When programming things like genetic algorithms we are in complete control of the fitness function—we author it ourselves—but even when we are not the direct authors of the fitness function, we can approach it sideways and try to model how it would work in the environment we created.

In the real world, we can see things like giraffes evolving over time to fill a niche where they have an opportunity to thrive. The victory condition for a giraffe is to survive long enough to reproduce, and this will require a steady source of food, relative safety from predators, and a fair chance at competing for a mate. Their longer necks allow them access to a food source with less competition, and the population grows in response to the improved living conditions. Just like the possible emergence of something like giraffes can be predicted by seeing that there are untapped resources in the form of tall, lush trees, we can anticipate the emergence of certain behaviors by examining the victory conditions of the game.

THE UROBORIC CYCLE For every game, there are strategies (behaviors) for winning, and given enough time the behaviors we observe will drift toward strategies that are better at winning. Simply put, people will get better and better at winning the game with time. The rules that the designer of the game puts in place (the internal rules) are the first building blocks for the fitness function for behaviors, so we must take great care to make sure that the internal rules create a problem that is solved by behavior we want to observe. We need to create the game in a way that ensures that the optimal winning strategy is something that we want our players to do. When the desired behavior of a player is not aligned with the optimal winning strategy, you can end up with a product that is unenjoyable for many, hard to manage, and difficult to scale.

In any game environment with multiple players, the situation gets complicated further. The way that players behave will change the game, and thereby the fitness function, and give rise to a new generation of memes in response to the changed environment.

How to best play a game with several players depends greatly on the other players, and the prevalent behaviors that we can expect from them act as a new set of external rules that will alter the fitness function even further. This iterative process of uroboric balancing will continue until one meme is successful enough to dominate the memepool.

The internal rules of the game act as an initial first-generation fitness function for player behaviors. As different strategies are tried, the internal rules will help determine which strategies are more successful. An example of this could be how a slow-closing reticle in a tactical shooter will favor a slower, more methodical player movement meme. Player strategies in this environment will have to strike a balance between accuracy and mobility, and with time, players will intuitively play the game in an ever closer-to-optimal way.

In a multiplayer environment, however, the players will have to compete with each other for limited resources. There are only so many kills to go around, and strategies can quickly develop that are more competitive. When this happens, all players will have to respond to this new competitive environment—an environment that was not initially designed by the game's creators (although it might well have been anticipated).

The players themselves become a part of the environment, constantly shaping the in-game culture toward better winning strategies.

MODERN WARFARE AND THE TRAGEDY OF THE COMMONS CALL OF DUTY 4: MODERN WARFARE is a tremendously popular first person shooter—fast-paced and nerve-wracking. Being good at it required great reflexes, fantastic hand-eye-coordination, and a fair deal of strategic thinking. Being good at CALL OF DUTY is hard, and not everyone can be competitive on the leaderboard... At least, that was the intent. The map Crossfire in MODERN WARFARE illustrates beautifully how a detrimental meme can gain a foothold because of the game's internal rules.

The Crossfire map was very popular because it supported a range of different play styles, but there was a crucial design element of the map that gave rise to some of the most frustrating and prevalent behaviors in MODERN WARFARE. The opposing teams would start on either end of the map, separated by various buildings and other obstacles. To get to the enemy, you would have to leave the relative safety of your starting location and move into the

more dangerous warzones in the middle of the map. To be a successful player on the Crossfire map, you had to strike a difficult balance between forward momentum and tactical retreats—it was a difficult map with a lot of room for improvisation. Topping the leaderboards on the Crossfire map could prove very difficult.

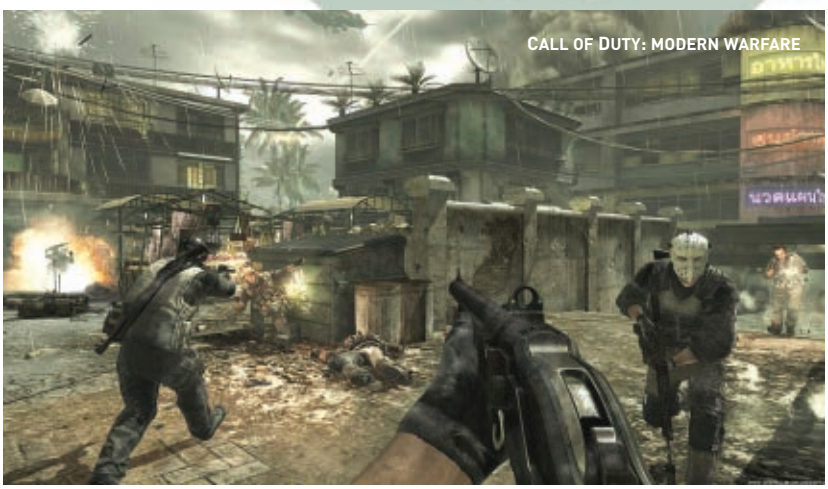
But players quickly found a way to get cheap points. One of the walls that separated the two teams early in the game was low enough for a grenade to be blindly thrown to the other side. There was a good statistical likelihood that an enemy would be on the other side, and chance would determine if you got a "free" point or not. The combination of predetermined starting points and insufficient obstacles between the teams had allowed one creative player to get a stylish kill.

The very first time it happened it was undoubtedly impressive—to have invented the technique, you needed to have a very intimate knowledge of the game and the map. The problem was that the meme was easy to copy, without requiring any particular skill at all. More and more people started blindly throwing grenades over that wall, which decreased the chances of each individual thrower getting a point—but increased the chances of a hapless opponent being unfairly killed early on. The "nade spamming" meme was the perfect storm of unfortunate circumstances.

The strategy was attractive to players because it was easy to use. Very little skill or exertion was needed to have a chance at a free point or two, so a lot of people gravitated toward it. The cost of trying it was also exceedingly low—in the fast-paced environment of the Crossfire map, it would be hard to use the grenades later on, so you might as well throw a few off early in the game and hope for the best.

And in hoping for the best lay the second rub. Random interval reward schedules can be incredibly addictive. The fact that you didn't know if you would hit an enemy or not turned throwing the grenade into a highly addictive gambling scenario. All you had to wager was a cheap grenade, and you could win very desirable points in exchange. It was a meme that was easy to copy because it was so easy to observe and understand the required actions, it was fairly successful at a low cost, and it was very addictive.

The more people copied the meme, the faster it spread, and after a while the meme was so prevalent that it was expected behavior. Both teams would throw grenades over the wall in the first 20 seconds of the game, and chance would determine how many players from either side would be taken out of combat before the battle even began. Needless to say, being taken out in the first 20 seconds of the game is incredibly frustrating—but if you chose to avoid the situation where you could get killed early, you would forfeit an opportunity to stay competitive with your teammates.


CALL OF DUTY: MODERN WARFARE

Once the meme had become popular enough, no one really stood to gain anything from it anymore. Too many people were nade spamming, and the overall fun of the game was taking a severe hit. The problem was that the meme proved very difficult to eradicate—the cost of changing the map itself was prohibitive, and for game hosts to kick off offenders proved to be a task of Sisyphean dimensions. The viral meme was spreading too fast to be contained. The internal rules of the game made it fantastically easy for the meme to spread; KillCams would replay the moments before your avatar's death, and the victims quickly caught on and reciprocated with more nade spamming. Teammates on both sides observed the behavior and joined in.

Although the behavior had been perfectly rational, beneficial, and entertaining before the meme become widely adopted, the situation changed with scale. There was only room for so many people to nade spam, but there was no mechanism from preventing everyone from having a go at it.

In game theory, what happened to MODERN WARFARE is referred to as the Tragedy of the Commons, and it is a surprisingly common occurrence in all kinds of everyday situations. But the problem of containing the spread, and preventing the Tragedy of the Commons, is better explained with another game theory classic: the Prisoner's Dilemma.

Because the meme was so easily copied, players would have to reach an agreement (explicit or otherwise) to cooperate in maintaining its spread. If everyone agrees not to nade spam, the game will be more enjoyable for everyone. The problem is that once people have stopped nade spamming, the winning potential for someone trying it gets very high—and it only takes one defector to nudge the uroboric cycle back toward the Tragedy of the Commons. Richard Dawkins, the father of meme theory, explains the phenomena of both the Tragedy of the Commons and the Prisoner's Dilemma very well in his 1987 documentary *Nice Guys Finish First*.

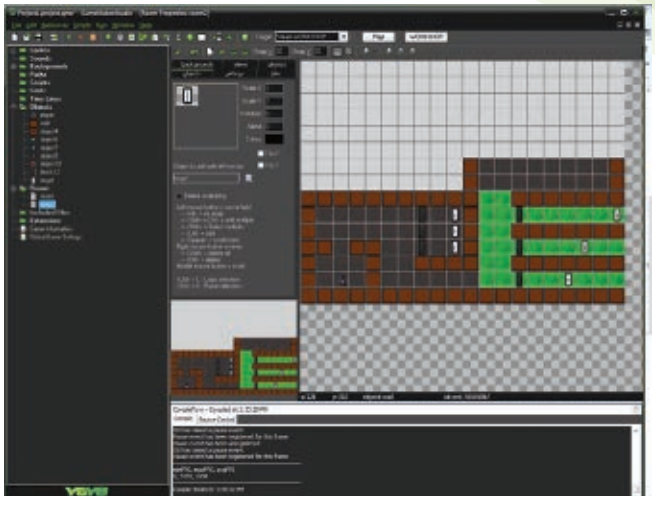
Interesting to note is that the KillCam was designed as a way of deterring people from engaging in negative behaviors like nade spamming and spawn camping. Although the KillCam introduced an element of penalizing the bad behaviors, it also created a new and very effective vector for the memes to spread. Measures like the KillCam are double-edged swords: On one hand, they punish wrongdoers, but recruit more wrongdoers on the other. At the end of the day, Modern Warfare's internal rules gave birth to an optimal winning strategy that was not in line with the desired behavior of players.

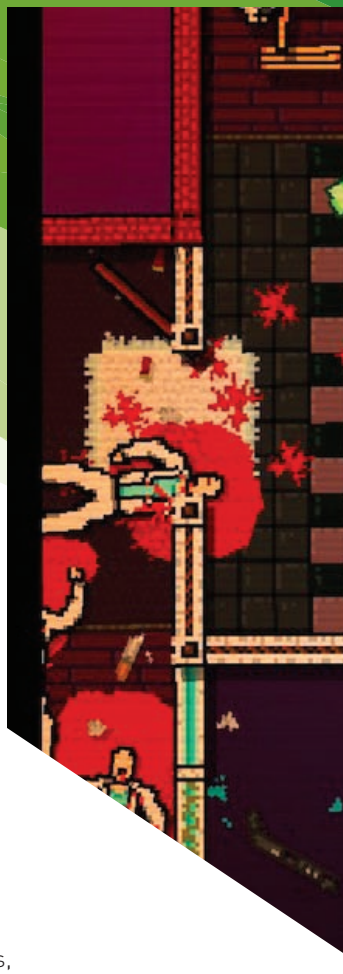
IN CONCLUSION Designers should take great care when creating the internal rules of their products, as the in-game culture that the game will inevitably have can have a greater impact on the product than your own direct design input. The meme can be a very powerful friend or foe, and although the meme cannot be tamed, we are ultimately the architects of the environments in which they spawn.

***

Nils Pihl is one of the founders of Mention LLC (www.mentionllc.com), an international consulting firm that specializes in engagement design, behavioral engineering, and game mechanics.

Editor's note: This article is an excerpt from an article originally posted on Gamasutra. You can find the full text here: http://bit.ly/12hReNW





GAMEMAKER STUDIO v1.1.7

GameMaker Studio is an integrated development environment for the creation of 2D games mostly through GUI actions rather than a programming language. It can create executables that run on Windows, Mac OS, iOS, Android, Windows Phone 8, and HTML5. GameMaker has a long history, starting way back in 1999. Since then, it has evolved with the times, added a lot of necessary features, and recently was bought by YoYo Games and re-released as GameMaker Studio. Along the way, GameMaker produced many thousands of little games, and a few gems like the free version of SPELUNKY, and more recently HOTLINE MIAMI. (Note that while earlier versions of GameMaker ran on multiple platforms, GameMaker Studio only runs on Windows.)

GameMaker Studio v1.1.7

YoYo Games
www.yoyogames.com

[Free version available; paid licenses range from $50 to $500]

SYSTEM REQUIREMENTS

Windows XP/Vista/7 (32-bit and 64-bit)
512 MB RAM
Screen resolution of 1024 x 600

PROS

1] Low-code approach good for introducing basic game-dev concepts
2] Accessible price tiers
3] Multiple-platform export features can save time and maximize exposure

CONS

1] UI is clunky and hard to navigate
2] Behaviors are tricky to edit in more complicated projects
3] Debugger lacks advanced features

MAKING GAMES WITH GAMEMAKER GameMaker Studio's goal is to allow people to make games without having to deal with actual programming languages. Most everything is done through a GUI, clicking on buttons and dragging and dropping components. GameMaker Studio does a great job of distilling the basic concepts needed for most 2D games, and provides some well-chosen fundamental building blocks: Sprites, Rooms, Objects, Events, Actions, and Timelines. It also offers some advanced functionality such as 2D physics and even some 3D support.

For users to be able to create games without having to resort to a programming language, the software's user interface needs to support and expose a wide range of concepts. GameMaker Studio does a great job of presenting lots of events that objects can respond to, and letting you hook up a huge variety of actions in response to them.

For example, creating an enemy that moves horizontally and bounces off the wall is extremely simple. You need to start by creating a sprite (possibly animated) that will represent the enemy. Then, you create an object that uses that sprite for a visual representation. On the object itself, you can add the event handling: The Created event sets a horizontal speed there, and the collision event against walls reverses its velocity. Done.

At any point, you can press the Play button and GameMaker Studio will build your game and run it in a window. Building and running the game—at least with small projects and a fast computer—only takes a second or two, so you can iterate fairly quickly.

USER INTERFACE NEEDS WORK For an environment that attempts to substitute programming languages with a GUI, I would expect a slick, super-streamlined user interface. GameMaker Studio's UI is not; it's clunky, ugly, and nonstandard. Maybe it's because GameMaker Studio is still built on top of the old codebase from the original GameMaker (which was originally written in Delphi), but in any case, it's a chore to use it.

HOTLINE MIAMI

The UI just feels substandard, from the initial ugly green-on-black editor colors (which can fortunately be changed), the glitchy behavior while resizing windows, a clutter of multiple floating windows in a parent area, or not having visual feedback about where an item will be dropped when you drag it over other items. You could dismiss those things as simply being effects of using an outdated GUI library, but then there are some inexplicable design choices, like an inability to copy and paste events between objects, and just one level of undo. And forget about any modern amenities like autocompleting variable names in the (fortunately few) places you have to type things. The GUI would have been subpar back in 1999, but today it's simply inexcusable.

TAKING STUDIO FOR A SPIN GameMaker Studio does a good job of getting a new user up and running very quickly through extensive tutorials and sample projects. I followed a couple of the tutorials and immediately had a good idea of how to go about implementing just about any simple 2D game. I *do* wish that it also included a good reference document in HTML format, or had one available online instead of the clunky Windows Help file one.

Creating new objects and behaviors is really easy, and you can put together a basic game in a matter of minutes. GameMaker Studio provides a huge range of events and actions available to compose your behaviors. This is one place where it shines due to its long history, as I'm sure they were accumulated over time as different games needed them.

In the rare case that an action for what you want is missing, you can create your own custom action in GameMaker Language (GML). But you can really do a surprisingly large variety of games without having to touch GML, so in that respect, GameMaker Studio is a success.

Debugging support is decent, but not great. You can run the game, pause, examine variables (by typing them by hand), and step execution frame by frame. Unfortunately, you can't add breakpoints or step through the logic that occurs in each frame from the debugger, and considering how logic is spread among a bunch of objects, each reacting to events and triggering other events in turn, it's a pretty significant omission.

One-button publishing for different platforms is a very attractive feature on paper. The version I reviewed only had the ability to publish to Windows (as an installer or a zip file), but other versions can create App Store-ready iOS executables. Dealing with different types of input for different platforms is a challenge (mouse vs. gamepad vs. touch), but GameMaker Studio manages to abstract out most of those differences. Your objects can respond to specific input events, whether they were generated by a keyboard press or a screen touch. Obviously, interface features that rely on inputs with no analogous touchscreen input (like mouse hover, for example) will need to be re-implemented, but most basic input functionality will work across platforms just fine.

GameMaker Studio's fundamental space units are pixels, which I found shocking. That makes the game logic dependent on your resolution, which is a bad idea for any kind of multiplatform (or for a platform that has many resolutions). GameMaker Studio does alleviate some of those problems by providing a scaling factor to match the target device's native resolution, but that might result in slightly blurry graphics.

RICK O'SHEA.

Similarly, the default unit of time is a "step," which is the time of a frame (either at 30Hz or 60Hz). Apart from being a very counterintuitive way to measure time for new users (how long is 5,000 steps?), it also means that running at a different framerate (either because it's a different platform or because the game performance bogs down at some point) will cause the game to slow down or speed up. Fortunately the program provides some ways to get the real time elapsed for the current frame, which you can use in your calculations, but all the event code and alarms rely on "steps."

GameMaker Studio provides source-control integration by integrating with Subversion already installed in your development PC, which is interesting (and actually the way I prefer it). However, that might not be the best way for the intended audience, since it requires them to install Subversion separately. Instead, I would have considered having some kind of simple source control built in to the IDE, or maybe not even provide any kind of source control at all.

One major flaw I found is the inability to reuse behaviors of any kind (without writing custom GML scripts). It seems that making different enemies with some shared behavior involves copying and pasting the same event handling and actions into the new enemies. This is an extremely error-prone process, and can take up a lot of time as soon as you start making some changes. I'm glad they kept things simple and didn't do some kind of inheritance or component system, but I was hoping to be able to group some events or actions and reuse them easily across objects.

This leads me to the biggest flaw of GameMaker Studio: Even though you can create very simple behaviors really quickly, the resulting logic becomes extremely complicated for anything but the most trivial projects. The flow of actions is spread between many different events and alarms, instead of expressed simply in a single page of code, and it's simply impossible to follow. Making significant changes to an existing game is a very delicate process, and when combined with the lack of good debugging, makes development of any kind of medium- to high-complexity game extremely difficult.

WHO ARE THE GAMEMAKERS? The question in the front of my mind while I was using GameMaker Studio was "who is this intended for?" YoYo Games's website claims that it's for "entry-level novices and seasoned game development professionals equally."

For seasoned professionals: I would not recommend GameMaker Studio for quick prototyping. Even though you can get some simple objects moving on the screen quickly, iterating on behaviors becomes very cumbersome and slow because of the constant clicking and dragging. Once you have a basic main loop update and rendering, adding behaviors in code is much, much faster.

In spite of my criticisms, I think GameMaker Studio is an excellent tool for teaching children or people with absolutely no programming knowledge how to make games. The skills they acquire with GameMaker Studio won't be transferable to other environments, but they will learn about the fundamental concepts that go into making a game: the different components, how they relate to each other, what kind of logic is involved, and so on. Learning that alone is arguably the biggest step toward starting to make games. The fact that they can hit the ground running, play sounds, and see things moving on screen right away is a huge boon to keeping motivation high and allowing people to learn quickly. Afterward, they can move to a code-based environment like JavaScript, Python, or even Codea on iPad.

***

Noel Llopis is a game-industry veteran turned indie-game developer. He avoids violence in his games and instead relies on creativity and sharing. His latest games include CASEY'S CONTRAPTIONS and FLOWER GARDEN.

Intro to openFrameworks

TEST THE MOBILE GAME DEV WATERS WITH OPENFRAMEWORKS

Mobile development is all the rage, but getting started can be a chore for console or desktop developers due to platform differences. If you work with Android, you can use C but have to deal with a large quantity of disparate devices. If you work with iOS, you generally have to touch some Objective-C. But if you want to have a feel for programming interactivity on touch devices without committing to a whole new environment, consider trying openFrameworks, a multiplatform creative coding environment in C++ that will get you a leg up on having an update/draw loop, a drawable canvas, and a way to read input from the various sensors on the device. It's not the right platform if you want to make the next INFINITY BLADE, but it powers some high-quality, popular games like SPELLTOWER and newer versions of SUPER HEXAGON, so it is by no means a dead end.

WHY MAKE MOBILE STUFF? As engineers, we turn assets and ideas into workable products. We are gatekeepers between ideas and usable things, as well as improvisers who must fill in the unavoidable gaps in spec docs. Having a holistic view of how your code will perform with people and machines is important in the process of developing games that feel good and are easy to use. And sometimes, platform paradigm shifts make us reconsider certain interface patterns; for example, the lack of a "hover" state in touch devices has effectively killed the tooltip, and as a result developers must place more emphasis on having buttons that are obvious in their function without further interaction. In terms of game control itself, there are dimensions to explore related to directness of control vs. abstracted control and the increased physical performativity.

GETTING SET UP For this tutorial, you'll need a Mac running OS X 10.7 or above, and an iOS device running iOS 5.1 or above. Register an Apple ID (you already have one if you use iTunes or iCloud) at https://appleid.apple.com, and use the Mac App Store application to download and install Xcode. You can also find Xcode at Apple's Developer site, but the Mac App Store version gets delta patches, which will considerably speed up your updates.

Next, download the latest release of openFrameworks at www.openframeworks.cc/download (as of this writing, the latest version is 0073) and unzip it. Open the directory in the Finder. You can open some files in the Examples directory to poke around some of the code before diving in to your own app. Each example explores a different bit of functionality of openFrameworks or the iPhone, including the touch sensors, the IMU, and the GPS.

A PROGRAM, STEP-BY-STEP First, create a new project. In order to ensure that the proper libraries are included, openFrameworks now includes a project generator, so just open the projectGenerator directory, run projectGenerator.app, and name the project whatever you want.

Next, click on Addons to see some of the libraries that have openFrameworks wrappers. You don't need to select ofxAccelerometer, ofxiPhone, or ofxMultiTouch, as those are included by default. Once you're done browsing the libraries, select "Generate Project," and open the generated project—we're going to start digging in.

The file main.mm contains some boilerplate code to set up the rendering context. If you're on a device with a high-DPI/Retina display, you'll want to get a reference to the app's window and enable Retina support. After that, you'll call ofSetupOpenGL() to set up the rendering context. Change the arguments to match the resolution of the device that you are targeting (see **Listing 1**). Then run your oF app—in this case testApp. The meat is in testApp.h/mm (the .mm extension indicates that the file contains objective-C++, a dialect that allows objective-C and C++ to mingle). If you wish, you may rename the files and the class contained therein (don't forget to change the ofRunApp call in main.mm to match) to something more appealing.

Note that you can expand iOS + OFLib.xcodeproj → addons → ofxiPhone → src to see all the oF wrappers for iOS

[LISTING 1: INITIALIZING THE WINDOW, OPENGL CONTEXT, AND OF APPLICATION.]

```
int main(){
  //
  // Retina-specific code
  //
  ofAppiPhoneWindow *iOSWindow = new
ofAppiPhoneWindow();
  iOSWindow->enableRetinaSupport();

  //

  ofSetupOpenGL(iOSWindow, 640, 960, OF_
FULLSCREEN);
  ofRunApp(new testApp);
}
```

LISTING 2: ADDING SHIP STATUS VARIABLES.

```
// Add vars for ship status
class testApp : public ofxiPhoneApp {

    void deviceOrientationChanged(int
newOrientation);

  private:
    // Ship status
    ofPoint currentPosition;
    ofPoint thrust;
    float angle;
}
```

LISTING 3: INITIALIZING THE SHIP STATUS VARIABLES FROM LISTING 2 IN THE SETUP() OF TESTAPP.MM.

```
// Set up variables
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 960

void testApp::setup() {
  ofBackground(0, 0, 0);
  thrust = ofPoint(0,0);
  angle = 0;
  currentPosition = ofPoint(SCREEN_
WIDTH/2,SCREEN_HEIGHT/2);
}
```

LISTING 4: USING THE DRAW() ROUTINE TO DRAW OUR SHIP.

```
// Draw a triangle to represent the ship
void testApp::draw(){
  ofPushMatrix();

  ofTranslate(currentPosition.x,
currentPosition.y);
  ofRotate(ofRadToDeg(angle));

  ofSetColor(0, 50, 200);
  ofTriangle(-5, 0, 0, -15, 5, 0);

  ofPopMatrix();
}
```

LISTING 5: WRITING OUT OUR TEST GAME'S BASIC MOVEMENT PARAMETERS.

```
// Add a variable to store touch data
class testApp : public ofxiPhoneApp {

    float angle;

    // Input
    ofPoint lastTouch;
}

// Handle touch events to control the ship
void testApp::touchDown(ofTouchEventArgs &
touch){
  lastTouch = ofPoint(touch.x, touch.y);
}

void testApp::touchMoved(ofTouchEventArgs &
touch){
  // Find touch vector
  ofPoint currentTouch = ofPoint(touch.x,
touch.y);
  thrust += 0.16f * (currentTouch -
lastTouch);

  // Update angle
  angle = atan2(thrust.y, thrust.x) + M_
PI_2;

  lastTouch = currentTouch;
}

// Apply changes to ship's position based on
thrust in update
void testApp::update(){
  // Update position
  thrust *= .99f;
  currentPosition += thrust;

  // Boundary wraps
  if (currentPosition.x > SCREEN_WIDTH) {
    currentPosition.x = 0;
  } else if (currentPosition.x < 0) {
    currentPosition.x = SCREEN_WIDTH;
  }

  if (currentPosition.y > SCREEN_HEIGHT) {
    currentPosition.y = 0;
  } else if (currentPosition.y < 0) {
    currentPosition.y = SCREEN_HEIGHT;
  }
}
```

LISTING 6: ADDING A TRAIL OF SMOKE PARTICLES.

```
// Add container to store prior positions of
the ship
class testApp : public ofxiPhoneApp {

    // Input
    ofPoint lastTouch;

    // Trail point container
    deque<ofPoint> pointList;
}

void testApp::update(){

  // Append current point to list for
drawing smoke trails;
  // offset by 5px so it appears to be
coming from behind the ship.
  pointList.push_
back(ofPoint(currentPosition.x - (5 *
sin(angle)), currentPosition.y + (5 *
cos(angle))));
  if (pointList.size() > 50) {
    pointList.pop_front();
  }

}

// Draw circles from past position of
increasing size to represent smoke trail.
void testApp::draw(){
  ofSetColor(255, 255, 255);
  for (int i = 0; i < pointList.size(); ++i)
{
    ofCircle(pointList[i].x, pointList[i].y,
i/6.0f);
  }
```

core functionality. The "sound" and "utils" directories are particularly juicy.

Now look over the boilerplate in the sample app. You'll initially see a list of slightly more than a dozen empty methods. The setup() method should be self-explanatory; it is called once on program launch. The update() & draw() methods are called every frame, in that order. exit() is called once, when the program exits. Clean up here.

The remaining methods all handle events. The touch events should be fairly obvious: Whenever a user touches down, moves a touch, or ends a touch, an event occurs that passes the touch (an object with an ID and some coordinates and other metadata) to your program. The lostFocus() and gotFocus() events occur when a user backgrounds or resumes your app, respectively. Your app will no longer receive slices of processor time in the background, but its memory contents will be preserved until the system starts to run out of memory (more on that later). Take note of this if you're using time steps in your update function, because you might have an enormous timestep between updates if the app has been backgrounded.

`DEVICEORIENTATIONCHANGED()` is called when a user rotates their device, say, from portrait to landscape. Note that there are two portrait orientations, portrait and portrait upside, and two landscape orientations, landscape with the home button on the left and landscape with the home button on the right, as well as face-up and face-down. (These are the official descriptions; this level of verbosity will become familiar if you engage further with the iOS SDK.)

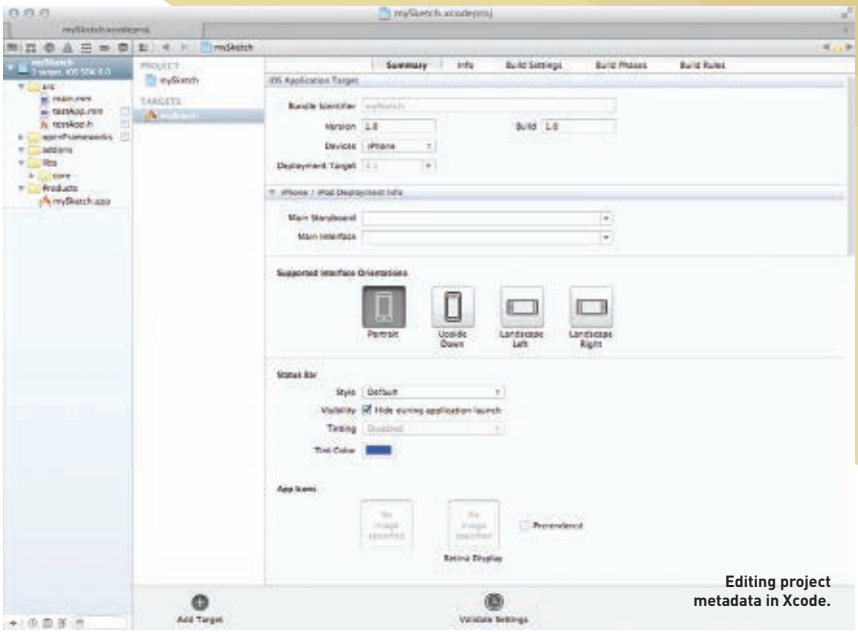
`GOTMEMORYWARNING()` is called whenever iOS starts to run out of memory. It may not be your fault, and this may get called while your application is in the background. If you take up very little memory, or if your app frees sufficient memory when asked, your application remains open. If your app uses much memory and does not free much, it will be shut down immediately, preventing the user from quickly resuming the app.

For the sake of brevity, we'll use member variables of the openFrameworks testApp class to handle our data storage.

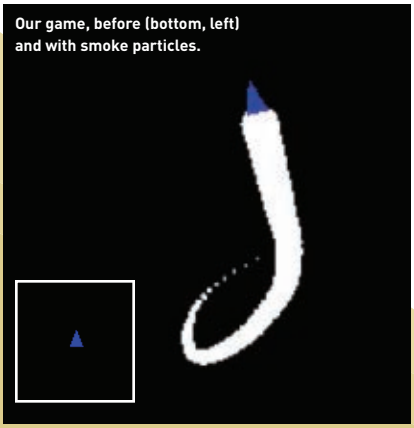
DRAWING Let's draw a triangle to represent a space ship. Add member vars to testApp.h (**Listing 2**), and initialize them in the setup() of testApp.mm (**Listing 3**). ofPoint is a standard Vector3 class.

Now we'll want to draw the ship in the draw() routine (**see Listing 4**). Fortunately openFrameworks's wrappers all have conveniently explicit names: ofPushMatrix(), ofTranslate(), ofRotate(), and ofPopMatrix() all do exactly what they sound like they do (and, in this case, call the similarly named glPushMatrix() and its associated functions). Note that whenever you call ofSetColor(), everything you draw will be that color until you call ofSetColor() again. You can pass RGB or RGBA values to ofSetColor, although you will have to call ofEnableAlphaBlending() (and ofDisableAlphaBlending() when you are done alpha blending) to render with alpha. ofTriangle takes three ofPoints or six floats to define the points of the triangle.

Now that we're drawing to the screen, we can easily visualize interaction. As you can see in **Listing 5**, we'll let the player control the ship by using touch-movement to define vectors of "thrust," similar to the game BIT PILOT, and we'll wrap the ship around the edges of the screen as in ASTEROIDS. Note that we're performing some tasks in the touchMoved event handler that might make sense in update(); this is to ensure that we don't miss any input between frames.



Editing project metadata in Xcode.



Our game, before (bottom, left) and with smoke particles.

Right now, our ship is small and can move quickly, which makes it easy to lose track of. In **Listing 6**, we'll make our game draw a tapered trail of smoke particles, which will make the movement of the ship easier to track and will look kinda cool in the process. Gamefeel!

NEXT STEPS Now we're going to skip the rest of the "making an actual game" part and focus on the other steps you'll take to make the product App Store-ready. You'll need a homescreen icon, a launch image, and an app name. If you click on the blue icon in the sidebar that reads "mySketch," (see figure below), you'll be able to specify valid device orientations and drag in normal-DPI and high-DPI icons and launch images. (The "Prerendered" option for the icon defines whether a shiny band is drawn across the top half of the icon.)

To change the name of the app as it appears on the home screen, select the mySketch target, click build settings, and find the "Product Name" field. When you're ready for release, register your app on iTunesConnect, in Xcode select Product->Archive (note that you'll need to have an iOS device, not a simulator, selected for your scheme in order to archive), then go to the Organizer (the icon for which is in the upper-right), select your build, and click Distribute. Those are the basic steps to getting on the App Store; by now, you should have enough information to readily search for answers should you become stuck. ip

***

Jonathan Beilin is the iOS person at DIY.org, a self-directed learning and invention site for young makers.

DECLARATION OF DEPENDENCE

NO ARTIST IS AN ISLAND

We like to think of ourselves as free spirits. We're artists! Independent and creative people who march to the beats of our own drums. Of course, most of us don't have the luxury of actually living that way—outside the indie-est end of the indie scene, most of us spend our days as parts of teams (often enormous teams). We get frustrated by the creative costs of technical tunnel vision—the way it stifles creativity, generates bureaucracy, and eats away at the pride of ownership that ought to make this job so much fun. On a less exalted plane, the fact that we are small cogs in big machines involves a lot of down-to-earth frustration and hassles. The fact is, we're dependent on each other—and so are our files.

We suffer from the sheer friction generated by lots of people—even smart, professional people—working on their little isolated projects, sub-projects and sub-sub-projects. We often experience the complex set of personal and technical relationships embedded in our workflows as hassles. You come in today, sync up while grabbing coffee, and discover that somebody redid the brick texture you were using, which made your charming red-brick cottage looks like an ugly 1950s housing project. Or maybe you awoke to find your email overflowing with nastygrams from the design department: Your innocent effort to smooth out a hitch in the walk cycle has broken combat for half the maps in the game. This is the gritty reality of working in a highly interdependent medium where nothing really stands on its own. It's a pain.

IT ALL DEPENDS If you want to stay sane in this kind of work, you need to get a good handle on the way your work relates to that of other people around you. Changes can flow from one part of the production to another in unexpected ways, and if you don't master the complex web of dependency between people and files, you'll be worn down by it.

To stay sane, you need good forensic tools for understanding and tracking dependencies between files. This ability, unfortunately, does not typically come standard with version control software, largely because there are such a bewildering array of possible relationships: models that use textures, animations that share rigs, levels that include props and scripts—the list goes on forever. It's impossible to list them all in detail, but on the highest level there are basically three kinds of dependencies: direct dependencies, asset dependencies, and implicit dependencies.

DIRECT DEPENDENCIES Simple relationships, where one file depends on a handful of others to do its job, are the most common example of interdependence. The classic examples are models that use texture files, or environments that include references to props. In most cases, these kinds of dependencies are explicitly recorded in a Max or Maya file as texture or file references.

Because simple dependencies are direct parts of the tools we work in every day, they can be supremely irritating. Luckily, though, the problems they cause are mostly minor. When you forget to check in a texture or referenced model, the next user to open the file will quickly discover what's missing (and won't be able to make new changes until the oversight is fixed). Of course, this doesn't mean these problems aren't infuriating: Try checking in a file that spams 50 dialogs about textures that you included from your thumb drive and see what happens if you don't believe me.

If you want to be a real professional—and worry about the quality of your secret Santa gifts—you need to be diligent about managing your own dependencies. Carefully prepping your check-ins and making sure to work on the latest versions of other people's files are the basics of good manners in our business.

Still, people are fallible, so it's wise to provide artists with tools to police these kinds of relationships. These could be as simple as a game engine providing predefined set fallback textures that appear in place of required files: As long as these are garish enough to attract attention, they can alert team members to missing dependencies without actually crashing the game and bringing development to a stop. More helpfully, tools can check art files for dependency relationships and make sure that relevant files are always checked in together. A Maya script that ensures files can't be referenced from outside the main project directory means no more thumb-drive embarrassments. A Perforce trigger that warns "The model you're trying to check in uses two textures that are not currently in source control—would you like to add them?" takes only a few hours to create, but can save many times that over the course of production.

ASSET DEPENDENCIES The next major type of dependencies come up when one complete asset includes others—each of which may itself be a small set of simple dependencies. A common example would be a model of a house that includes elements from a library of standard windows, doors, and appliances. Each of the included files reference shaders or textures of their own—or potentially, even more assets with their own inputs. This network of inputs and outputs can quickly become very complex. If you're familiar with Maya's Hypergraph or Max's Schematic view, you get the idea—and you know how quickly these kinds of webs of dependencies become impossible to disentangle visually or mentally.

The complexity of asset dependencies makes it hard for an artist to unravel the entire web of relationships when something needs to be tracked down. Consider the problem facing an animator working on scene who notices something wrong with one of the characters' hats. Mentally, it's obvious (that guy's hat looks wrong), but figuring out where the bad hat came from requires working backward through all of the references until the right bit is found. It might be buried two or three layers of file references deep.

Of course, the animator trying to find the bad hat may not even be poking through

something straightforward like a Max file reference dialog. The hat may be called by a bad line of script or an incorrectly formatted entry in "character_hats.xml." When the dependencies move out of the relatively simple world of Max and Maya, things really get hairy.

IMPLICIT DEPENDENCIES Implicit dependencies occur when changes to one file affect the ways in which other files work. The classic example of implicit dependency is a character skeleton. Most engines require a character's skeleton to be the same everywhere it appears. If an artist accidentally renames, deletes, or adds a bone to an existing character, the entire pipeline may grind to a halt—despite the fact that the dozens (or hundreds) of files affected by this change are themselves unchanged, and probably still look the same to someone who opens them up in Max or Maya.

Implicit dependencies are evil. They can break existing content completely or change its meaning without any warning. They are also quite difficult to track, because they don't correspond to simple file references. In the case of the character skeleton, for example, the problems are going to show up all across the production as cryptic error messages or in-game bugs. A simple dependency is self-diagnosing, since when a model raises an error flag that says "I can't find texture X," the answer is obvious: Go find texture X. Implicit dependencies, however are merely implicit—so it's not altogether clear where the problem lies, unless you're one of the technologically privileged caste who knows all of the unwritten rules of your game. Most likely, finding and fixing the problem will involve a long troll through recent changes and a lot of speculative talk, looking for something "suspicious" that gives away the problem. That's hardly an efficient use of anybody's time.

The only real solution to the problem of implicit dependencies is not to have them. When a dependency relationship between files exists, it should be made explicit somehow. For example, the dependency between a master skeleton file and the animations that depend on it could be expressed by using the file referencing functionality inside the DCC tool. This would automatically push changes from the master file to the animations. Use of referencing in a case like this turns a nasty, potentially dangerous implicit dependency into a vanilla simple dependency, albeit one which, depending on the pipeline, might also necessitate a reprocessing or re-export of the files in question. Nonetheless, it's an improvement over leaving the relationship between files lurking in the tangled underbrush of the production.

Alternatively, an implicit dependency can be made explicit by recording it in a database or by creating a file (usually called a "sidecar file") whose job is to explicitly track the relationship. For the animation example, there might be an XML file that lists all of the animations that depend upon the master file. When the master is changed, the users can be warned about the large number of files that may need to be updated. Conversely, when the individual animations are opened, they can consult the tracking file to make sure they are in sync with the master and warn users if an update is required. Tools can use the master file to know which files need to be fixed. Moreover, the master file has version history so changes in the relationships can themselves be tracked.

TOP: Graph views are an excellent way to understand dependency relationships CENTER: ... Except when they're not. Graphs work well for a local view of dependencies, but they're worse than useless for getting the big picture. BOTTOM: An asset-centric view makes more sense than a big graph. Find the asset first and then see it's inputs and outputs. This example is a dedicated app, but the same information could easily be in a wiki page.

Explicit tracking of dependencies outside of the DCC tools also makes it easier to track relationships both upstream and downstream. Max or Maya can only track upstream dependencies: A model file knows which textures it needs, but the texture files have no idea how many models, if any, are using them. If those relationships are tracked in sidecar files or a database, the entire network of connections can be visualized and graphed.

BREAKING THE CYCLE OF DEPENDENCY The overall dependency network for a big production can be staggeringly complex, but the basic building blocks are just files and the links between files. It's fairly simple for scripts to collect this information by analyzing art files or sidecar files. Some teams prefer to do this in an offline tool; others put the entire web of relationships into a relational database. Once you decide to tackle the problem, implementation isn't really that tough—computer science has lots of tried and true approaches to understanding dependencies. The real challenge is presentation: getting this information in front of the team in a usable format. It seems natural to try to view dependency information as a tree or node view, similar to the outliner or graph views in DCC tools. This works well for small subsections of the project (say, for understanding the inputs of a single asset) but breaks down quickly for larger amounts of data.

On the other hand, websites—which are built around hypertext links between pages—are a natural way to represent the links between assets: Once you've analyzed the graph, creating HTML or wiki pages that correspond to your assets makes it easy for artists to navigate this complex information using familiar tools.

The asset page for a model would include upstream links to the textures included in the model, as well as downstream links to scenes or game levels that use the model. It's also a good idea to include summary pages for tracking big-picture items: total numbers of assets, resource usage, assets grouped by review status, and so on.

With a good dependency tracking system, it's possible to see optimizations that would be invisible on the level of individual files. For example, you could find a subset of the prop library that has a high degree of texture sharing and hence takes up less memory as a unit, or spot which character has the most expensive animation set and needs to be trimmed down. Also, seeing the dependencies gives artists advanced warning about the potential impact of their decisions. When you're thinking about tweaking a prop, it's nice to know if it shows up in five houses or 50 before you try anything too fancy.

Some people will tell you "there's no 'I' in 'team,'" but for us artists, there really is. Our working and creative lives are tough enough without the constant drizzle of minor irritants that come from badly managed collaboration. Good tools and good practices can make this a lot less irritating, freeing us to concentrate on important stuff—like bemoaning our lack of creative freedom and the artistic blinders of our mass-market medium. pp

***

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.

LIFE AFTER SHIP

DESIGN YOUR GAMES TO HAVE AN EXTRA LIFE OR THREE

When *Magic: The Gathering* first entered the gaming scene back in 1993, the mere idea of a game based on an ever-evolving pool of collectable cards was just a zygote of an idea. Richard Garfield and the rest of Wizards of the Coast knew the game had real potential, but no one knew how the game experience would really play out.

It's not surprising that they got some things wrong. Their limited playtesting was not nearly enough to find all the convoluted strategies players would devise, and they had no historical data to check for problem spots. Sophisticated analysis of the game did not yet exist—they did not know (or fully appreciate) how powerful drawing cards would be in their game, and thus printed a card that allowed a player to draw three cards for one mana. And many of the rules were written ambiguously, so new expansions that introduced new rules brought in unexpected conflicts—and made it clear that card rules language needed to be much more structured and unified than it was previously.

Wizards of the Coast also underestimated their own popularity. They expected players to buy a deck and a couple of booster packs. Hardcore players started to buy booster packs by the case. Rare cards that Wizards of the Coast assumed would show up only once or twice in a deck ended up being highly sought after, and soon devoted players were packing four of each (the legal limit) in their decks, and destroying their less-invested opponents in the process, often in a couple of turns. The value of the best rares shot into the stratosphere, creating a legitimate aftermarket for cards.

Wizards has succeeded beyond their wildest dreams, reinventing the board-game industry (and saving America's game and comic book stores) in the process. But it was clear that *Magic* had some bad structural problems that needed to be addressed. Fortunately, *Magic* had a winning core game design, which gave them the resources and time they needed to fix these structural issues. *Magic* was a game that had a long life after ship, and its game designers took advantage of this to great effect.

FINDING THE DESIGN SPACE For most board games and single-player video games, expansion packs are the norm. Thanks to our age of connectivity, this trend is now buttressed by downloadable content (DLC) for games on online platforms. Some games, though, are uniquely suited for this sort of life after shipping. Collectable card games, such as *Magic: The Gathering* and *Pokémon*, are built hoping to capitalize on this as a core business model.

Massively multiplayer titles, such as WORLD OF WARCRAFT and STAR WARS: THE OLD REPUBLIC, are also games that beg for a post-launch strategy. Most MMOs opt to offer players occasional downloadable content (often for free, but sometimes for a small charge), and some bolster this with an occasional expansion pack. Again, the business model is central to the approach here; subscription models want to keep players engaged and happy for long, continuous periods of time, whereas games that depend on microtransactions are content with people stopping back for a brief visit, to purchase and consume the new content.

All of this is obvious, of course. What is less obvious is that designers need to identify their post-

launch realities. What design will allow the game's expansion if it's successful? Is this a free expansion you expect everyone to acquire? Is it paid for? Is it premium expansion content that will go on a store shelf, or is it DLC? If it is a competitive game, are you expanding in ways that are mostly cosmetic, or are you encouraging players to invest in making their characters more powerful? If characters are getting more powerful, is there actually room in the math for that?

PURE ESCALATION In WORLD OF WARCRAFT'S first three expansion packs, Blizzard simply made the numbers bigger. Players wanted more levels, more and cooler gear, and tougher challenges. However, there were some complications. For one thing, the stats of players had already been increasing—Blizzard had continued to release new raid content, and added new, improved gear for each tier of content they added to lure players into that content.

This meant that people right off the boat in new expansion areas started in a completely different place. Those who had been raiding would play through a significant chunk of the new expansion content without getting any gear upgrades at all, whereas those who hadn't would find "standard"-quality items comparable to raid gear by accomplishing trivial quests, in a desperate attempt by the designers to close that gap.

But there was a more insidious problem: The power curve in the game was exponential in nature, which meant for the math to work naturally, the gains between levels needed to grow bigger and bigger. As of this writing, a top weapon in the game now offers +947 stamina and deals 1996 damage per second, and this number easily grows to five digits when combined with abilities and modifiers. By comparison, Perdition's Blade (a top drop off of a Ragnaros from Molten Core, an early raid) did 61.7 DPS, and has no stat bonus (other gear he drops offers no more than +22 stamina). The number of hit points a raid boss today needs to have to survive 25 players dealing out that much DPS over the course of a 10-minute fight is staggering, and the numbers involved are large enough that the development team posted a blog to open discussions with the players about how to bring these numbers back into a sensible reality (http://us.battle.net/wow/en/blog/3885585).

ADDING COMPLEXITY Hardcore players who love a game frequently don't want just more of the same game. This works in some genres, such as first-person shooters and RPGs, where players can be satiated with expansions with more maps and more story. More often, though, these players want more mechanics and rule tweaks—even those that love the game are bored with it after a certain period of time. But this, too, raises dangers, mostly of overcomplicating the game for casual players.

This happens rather frequently in board games; lots of board games get expansions, and in many cases, the expansion packs make the game better. Kingsburg took the opportunity to patch the rules, fixing a couple of issues that allowed more potential strategies to emerge. *Game of Thrones's* second expansion fixed balance issues so fundamental to the game that they were made part of the core gameplay in the second printing of the game.

In other cases, additional layers of complexity can make the game impenetrable to new players. Seven Wonders was widely praised by the board-game community as an excellent short and casual card game, in large part because it was quick to play and easy to teach—ideal for playing with casual audiences and/or while waiting for latecomers to show up on game night. They've launched two expansions since the original release, which have added tremendous depth, but the game is no longer a casual, friendly game that can easily be explained to new players in less than five minutes. A bewildering array of new cards and symbols need to be taught, and even experienced players need to reach for the rulebook.

The issue is that the two audiences need different things. Hardcore players need new patterns to learn and new strategic elements to challenge them. A new player, on the other hand, needs to pick up all the rules, addendums, and strategies that the more experienced player has accumulated over years in one massive information dump.

The hardcore fans of the game, of course, come at expansion content with all of their accumulated knowledge to date. Raid

WORLD OF WARCRAFT.

fights in WORLD OF WARCRAFT in the latest expansion are orders of magnitude more complex than the fights they initially shipped with, and experienced players welcome facing these more intricate challenges. However, this means that new raiders have a much more difficult learning curve to overcome than they did in that first round of content. This accumulation of difficulty is often forgotten, although Blizzard has found they need to make a significant effort to reduce this learning curve as much as possible.

PLANNED OBSOLESCENCE *Magic: The Gathering* started in a different place. As they weren't expecting players to chase and build all-rare decks, they discovered that the game was simply way too fast once players did so. Players could frequently win in three or four turns. Printing more powerful cards would accelerate the problem. Wizards of the Coast tried printing cards that were less powerful, and discovered, perhaps unsurprisingly, that they wouldn't sell. They quickly hit upon the idea of a Restricted list for cards (cards that could not be used in tournament play), to weed out the ludicrous cards, but the play environment was still too fast, and worse, the sheer number of interactions that were available in the pool was becoming impossible to track and design for as new expansions rolled in.

So they introduced obsolescence in their game design, with the establishment of "Standard" as the primary form of Tournament play. Standard tournaments allow only the inclusion of the latest two "blocks" of cards (where a block is comprised of up to three expansions) and the most recent "core" set to come out, effectively limiting the card space to a fairly manageable number. This had a lot of obvious upsides: By waiting a few expansions, the game could be slowed down as those older, more powerful cards disappeared from the sets. It also created an incentive for the players to continually buy new cards, and not just be content with their old collection.

Planned obsolescence had a lot of subtly deep design implications as well. Every block could have strongly different metagames; in one block, counterspells would be common, and in others, they'd be hard to come by. The designers could make major changes to the design philosophy of the game, as they did when they redistributed various spells to different colors in order to even out the usage of those schools. And perhaps most importantly (to those of us designers who fancy ourselves as mad scientists, anyway), designers could take bigger chances with weirder mechanics, knowing they would eventually rotate out of the pool.

PLANNING FOR GROWTH This is not hugely applicable for those of us making digital games, except for one core, inescapable fact: *Magic*, a game that struggled out of the gate under the weight of many large structural mistakes, is now designed with expansion in mind. It is designed, at its very core, to grow, to change, and to reinvent itself with every expansion. This level of foresight serves it well: *Magic* will reach its 20th anniversary in 2013. The latest *Magic* expansion, *Return to Ravnica*, is selling like gangbusters, and high-level Magic players now play for thousand-dollar purses in sanctioned tournament play.

We no longer live in a world where a game is ever truly finished. If your game is beloved by fans and sells enough to satisfy the suits in your building, there will be a desire for expansion content, for booster packs, for downloadable content, and even for cosmetic skins. Designers are well advised to plan for this success, and to know where, exactly, the game's design allows them to go to satisfy these appetites.

***

Damion Schubert is the lead systems designer of STAR WARS: THE OLD REPUBLIC at BioWare Austin. He has spent nearly a decade working on the design of games, with experience on MERIDIAN59 and SHADOWBANE as well as other virtual worlds. Damion also is responsible for Zen of Design, a blog devoted to game design issues. Email him at dschubert@gdmag.com.

THE GAME-AUDIO LITERATURE REVIEW

HOT FOR GAME AUDIO

Your first foray into the deep waters of a new discipline is easy when Cupid shoots an arrow through the heart of your interactive amore. Once you've become entranced by your newfound love, it's up to you to put the effort into getting to know each other a little better. Sound, like love, is often hard to articulate in words, but thankfully these resources will help fuel the fire of your growing game-audio relationship.

I BROUGHT MY PENCIL When it comes to mapping the family tree of our sonorous interactive past, the comprehensive *Game Sound* by Dr. Karen Collins traces directly to the roots of our hardware progenitors. *Game Sound* covers the fundamentals of sound synthesis, hardware specifications, and an in-depth explanation of the historical high points of interactive and dynamic sound and music. For me, reading it for the first time was like taking a Tron-inspired step into the consoles and cabinets of my youth, colored with stories from creatives blazing a trail at the forefront of a new audio technology.

Meanwhile, longtime *Game Developer* contributor Alexander Brandon's *Audio for Games* takes a more direct and practical approach to explaining the inner workings of a traditional game development pipeline with a keen ear toward audio's role. Not purely an explanatory text, the book vacillates between presenting practical knowledge and offering specific insights into the tools, processes, and people skills that make for a smooth work methodology. While some examples will be seen as dated in today's rapidly advancing community, they represent a moment in time at the beginning of our current, waning console generation whose fundamentals transcend.

GIMME SOMETHING TO WRITE ON There is no faster way to gain a deep understanding of the principles of game development and its relation to audio than to crack open a game and peer inside its limitless abyss. *The Game Audio Tutorial* from Richard Stevens and Dave Raybould lifts up the hood of the Unreal Engine and walks you through the complex pathways of its audio functionality. Through step-by-step examples from a working Unreal project, they provide a hands-on interactive playground that illustrates each concept. Whether you learn better through reading and following along to steps detailed in the book, or are the kind of person who rolls up your sleeves and starts taking things apart within Unreal, this is a winning combination for trying to decode the lovers' whispers between game audio and game development.

I DON'T FEEL TARDY A formative part of my education was spent nose-deep in OpenAL documentation. I might not have understood much, but there was little else that would give a peek behind the curtain of game audio at the time. I soon discovered the Firelight's FMOD Designer toolset, and shortly thereafter Audiokinetic's Wwise hit the scene (not to mention Microsoft's XACT, which has since come and gone), providing an accessible introduction to game-

audio concepts that aligned with my emerging knowledge of game engines. These tools served as a bridge between the programming language and tools used to create sounds.

These days, though, there are plenty of documentation, manuals, and resources out there that can help introduce a new game-audio worker to the tools of her trade. Recently, the *Wwise Project Adventure* (which I developed at the request of Audiokinetic) was released to provide a comprehensive introduction to creating a Wwise project from start to finish. This acts as a companion to their already-extensive knowledge base and brings a cohesiveness to the complete process.

Firelight Technologies has also published some massive tomes of game-audio wisdom, most recently the *Basic Functionality Guide* for its forthcoming FMOD Studio toolset from Stephan Schütze. While each of these seeks to familiarize the reader with a given toolset or pipeline, the universal truths of interactive audio abound throughout. Additionally, many game-audio hopefuls who are eager to expose their mastery of these tools and techniques have created countless blog posts and video tutorials, which can be used simply as a step-by-step introduction to many interactive audio concepts. You can find many of them chronicled under their respective tags at gameaudiorelevance.iasig.org.

I THINK THE CLOCK IS SLOW The output of Rob Bridgett over the last 10 years has been a steady pulse of intelligent, forward-thinking, and practical writing, and much of it is assembled in *From the Shadows of Film Sound*. His articles always seem to arrive exactly on time, or well in advance of discussions that are at the forefront of our discipline. Bridgett casts an eye toward insights gleaned from other entertainment mediums, couples those insights with the unique aspects of what makes games shine, and envisions a future for game audio tempered by experience and emotion.

Similarly, George Sanger's *The Fat Man on Game Audio* leads with emotion, serving less as a technical manual and more as a spirit guide and way of life. By weaving a tapestry of anecdotes, inside jokes, and startling epiphanies, Sanger brings out the human side of game-audio history and helps frame the formative years as a rock-'n'-roll saga of epic proportions. Interspersed with pictures of living legends and bygone heroines, it's a lovingly crafted fan digest that chronicles the beating heart of the industry.

CLASS DISMISSED It's a gift, in our tiny corner of the game development world, to have such a rich selection of diverse written inspiration to choose from. Whether you're newly enamored or have shared a long love that continues to deepen, there are more ways than ever to speak the international language of game audio. af

"I think of all the education that I missed, but then my homework was never quite like this." —Van Halen

***

Damian Kastbauer is rotating approximately 78 RPM at LostChocolateLab.com and on Twitter @lostlab.

Resources

Collins, Karen (2008) - *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design.*
Brandon, Alexander (2004) - *Audio for Games: Planning, Process, and Production*
Stevens, Richard and Raybould, Dave (2011) - *The Game Audio Tutorial: A Practical Guide to Sound and Music for Interactive Games*
Bridgett, Rob (2010) - *From the Shadows of Film Sound*
Sanger, George (2003) - *The Fat Man on Game Audio: Tasty Morsels of Sonic Goodness*

AVOIDING TUNNEL VISION

HOW FELLOW DEVS CAN HELP YOU TAKE IN THE BIG PICTURE

Back when I worked for Xbox Live, I frequently commented on the dangers of what I called "developer tunnel vision." Nearly all of the devs I spoke with were not paying attention to a diverse set of industry news sources. What's more, they were focused on at most a couple of similar platforms, and were ignoring the rest of the market. (Back then, everyone was talking about XBLA/PSN; today it's Steam/iOS; tomorrow it will be something else.)

At the time, this seemed completely insane to me—even suicidal. Didn't these devs understand how quickly things change in our industry? How quickly their current efforts could be rendered irrelevant by shifts in the marketplace, or by strategy shifts made by the platforms? Developer tunnel vision... It was so obviously reckless and short-sighted!

But then I started my own development studio. Almost immediately, I stopped dedicating several hours a week to following industry news, and found myself giving it a couple hours a month—if I was lucky. I started fixating on a couple of major platforms. Turns out, it's damned hard to make games, be a good father and husband, and do anything else at the same time.

JUGGLING PLATFORMS I justify it by comparing myself to other indies. Spry Fox is actively engaged with Google Play, Apple iTunes, Amazon's Appstore, Steam, and a bevy of web portals like Facebook, Armor Games, and Kongregate. Compared to most indies who are fixated on just iTunes and/or Steam, that's pretty good, right? Of course, since I've stopped reading, I'm missing crucial context about what's happening in the very ecosystems that we're focused on. You can only get so much insight into the market dynamics of iOS and Android by studying the performance of TRIPLE TOWN. And I'm clueless about emerging platforms, European and Asian game portals, and too many other things to mention.

This article is partially a mea culpa. Folks who work at game platforms (myself formerly included) tend to be pretty judgmental about what developers should and shouldn't be doing. Actually being a game developer is an eye-opening and humbling experience. But more importantly, I want to take this opportunity to encourage my fellow devs to do the one thing that can help counteract tunnel vision: Talk to each other as regularly and often as possible! It takes less time than comprehensively consuming several news sources a day and tends to be more fun, too.

BY DEVS, FOR DEVS Is there a developer meetup in your area? Join it. If there isn't, consider starting one. Places to start your search: the IGDA chapter list (www.igda.org/chapters) and meetup.com. Are you a member of a decent industry mail list? If not, join one and/or make one of your own. The Stanford Graduate School of Business hosts an open-to-the-public game industry list that is large and diverse, if unfortunately quiet. It's at least a place to start (http://stanford.io/KpCuRP).

Once you've connected with other developers, do yourself and everyone around a favor and don't keep secrets. Talk about the games you're launching. Share details about their performance. Describe things that surprised you (player reactions, revenue fluctuations... whatever.) The more you share, the more the people around you will hopefully be inclined to return the favor. And their feedback on the information you share may turn out to be invaluable.

Most game developers are never going to fully avoid tunnel vision. One way to counteract that is by making friends and sharing ideas and data. Too many of us are fixated on our "trade and design secrets." But odds are that your secrets are worth less than you think, and your ignorance is a greater liability than you can possibly imagine. At least, I'm pretty sure that's true for me! b

***

David Edery is the CEO of Spry Fox and has worked on games such as REALM OF THE MAD GOD, STEAMBIRDS, and TRIPLE TOWN. Prior to founding Spry Fox, David was the worldwide games portfolio manager for Xbox Live Arcade.

WHO WANTS TO MAKE SOCIAL GAMES?

WADING THROUGH THE APOLOGIES

This Thanksgiving I met a young fellow who worked in games, and we discussed our respective career paths a bit. He's a UI programmer/designer combo, which is a rare and useful bird in the aviary that is game development. We spoke about the importance of a cohesive user experience, how UI is the gateway to that, and how he and one artist had been given carte blanche to go through his entire game, screen by screen, to make sure everything was user-friendly, intuitive, and seamless. UI design is often one of those things left until the very end of the development process, which leads to games with visual experiences that aren't cohesive, so this was impressive.

I was reminded of a talk I saw at GDC China, where Double Fine designer Joe Kowalski discussed making UI that served to expand or reinforce a game's universe, rather than just being functional. My new acquaintance said yes, this was what he was trying to do as well, especially since this was a social game. And when he said the word "social," I could actually feel the ellipses forming as the conversation trailed off.

Then began the apologies: "We've actually been able to strip out all the things users hate, like paying to win, and all that sort of stuff, and we're targeting the core, so hopefully we can actually do something fun with it."

OLD-SCHOOL DEVELOPERS JUST DON'T LIKE SOCIAL How familiar was that latter part of the conversation? How many times, when you learned that a veteran industry acquaintance was now in social games, have you had this apologetic back and forth? They say, "Well, we're really trying to make this one interesting," and "This one's not that bad, though," and you trot out set phrases like, "Yes, it must be really interesting having that close a relationship with the player" or, "So how much do you pay attention to metrics versus intuition?"

Inevitably the conversation returns to the industry at large, the launch of the Wii U, how HALO 4 is out now, and whatever else you're actually interested in. This got me thinking: Who actually wants to make social games?

Social games are still where much of the money is (though that money is dwindling), and game development is a job, after all. Sometimes you have to take the work and make your rent. But it seems very few veteran game developers, especially those in the West, actually want to make these games.

This is what gives rise to the plethora of GDC talks titled things like "Are social games legitimate?"

Social Wars.

Or the incredibly frequent discussions of how social games "don't have to be evil." The apologies are right at the tips of our tongues when we discuss this entire branch of the industry. They're simply not the games we as developers want to play. There are some mobile arcade-style games, but there's also that social mobile genre where you click things on a timer.

I've noticed that veteran game developers tend to feel that console and PC games are "real," while social and mobile are not. I find myself having this bias, and I know those who work in social resent it. The trouble is many of these games tend not to be skill-based, and they're not targeted at developers, demographically. They're targeted at our moms and our kids. But didn't we get into this industry to make the games we want to play?

To make sure I wasn't crazy, I asked social developers on Twitter: When talking about your job, do you get defensive? Do you make excuses? Most said yes, and those who said no immediately got defensive, which said something. Partially they're just sick of traditional developers looking down their noses at them. But one person summed up the sentiment many were getting at by saying, "There's some cool stuff about it, but I'd rather make real games."

SO WHO DOES LIKE SOCIAL? It's a simple answer: Businesspeople like making social games. Social feels like a business more than a place where you manufacture dreams and whimsy. The money is in social because it's run and ruled by business folks who first saw games as a place to make money, and then pulled in the veteran developers to make it happen. And who knows this better than China?

China never had a console market. They don't have nostalgia for the past—not for 8-bit chip tunes, not for 16-bit chunky pixel graphics, not for flat-shaded polygons. The old consoles pretty much didn't exist in that market, at least not legally or officially. (As an aside, when I gave a talk at GDC China, one part that had attendees nodding in agreement was when I said, "Retro is big in the West—this probably doesn't make sense to you.")

China's market began with free-to-play, and that's the business they know. But it is a business, and they're aware of it. That's why the market there has gotten so precise about its monetization. It's looked at as a job—one that can make a good deal of money. There never was another successful model in the region.

That comes at a cost, of course. I actually saw an attendee at GDC China ask of a speaker, during a post-talk Q&A (paraphrased): "I understand your metrics for monetization; that's all well and good. But do you have any metrics or charts you can show us related to how much fun we need to add for retention purposes?"

I can imagine the guy going back to his team and saying, "No, no, no, guys, according to this chart here, we're going to have to add six more Fun Units if we want people to keep playing."

I'm a huge proponent of making the games you want to play. I want more diversity in games, but I want that to come from different people coming in and making the games they want to play. Let's get 60-year-old housewives making our social games—why not? But let's also stop apologizing. If you're in it to make money as a business, be straight about it. If you really enjoy making games for the social demographic, that's awesome too. But I say: If you don't actually like making and playing social games, just don't make them. ic

***

Brandon Sheffield is director of Oakland, California–based Necrosoft Games, and editor emeritus of Game Developer magazine. He has worked on over a dozen titles, and is currently developing two small-team games for PlayStation Mobile.

gamescom

Koelnmesse Inc.
8700 West Bryn Mawr Avenue
Suite 640 North, Chicago,
Illinois, 60631
Tel. +1 773 3269920
Fax +1 773 7140063
info@koelnmessenafta.com
www.gamescom-cologne.com

gamescom 2013: The entire gaming world in one place

The concept of the world's largest trade fair and event highlight for interactive games and entertainment is unique: it networks the entire value-added chain, from development and publishing to retail and the consumer. As the largest event of its kind in the world and the leading trade fair, it provides discussion platforms on all levels. It covers the entire spectrum of the international gaming scene:

- PC games
- Online games
- Video games
- Browser games
- Social games
- Mobile games
- Gaming hardware

The concept provides individual platforms for all target groups:

ENTERTAINMENT AREA: emotional gaming presentation for all – the world's largest playground for interactive entertainment
BUSINESS AREA: international meeting point for exhibitors, trade visitors and media representatives
GDC EUROPE: largest European developer conference

gamescom 2013 started with an exhibition space of 140.000 squaremeters. The trade fair and event highlight for interactive games and entertainment demonstrates with its leitmotif what the games world can expect in Cologne from 21st to 25th August 2013: The international games community – developers, providers, trade visitors, media representatives, retailers and thousands of gamers – meets at gamescom 2013 in order to experience together spectacular innovations and to celebrate the games and entertainment event of the year.

Koelnmesse and its partners, headed by the BIU (the German Trade Association of Interactive Entertainment Software), are already working flat out to further develop gamescom as Europe's central business and entertainment platform. gamescom awards are further developed due to the great popularity. The BIU also expects exciting novelties and innovations at gamescom when it comes to hardware and software innovations.

We look forward to welcoming you to gamescom 2013!

gamescom at a glance

- Business area, halls 4/5
- Entertainment area, halls 6-10
- Presentations of news and innovations of the entire industry
- gamescom award
- Games Developers Conference
- gamescom festival, City of Cologne

gamescom 2012 was a complete success

- 603 exhibitors from 40 countries
- 275.000 visitors in total
- 24,500 trade visitors (52% abroad)
- 5.300 journalists from 52 countries
- More than 120,000 additional visitors at City-Festival

FOCUS ON

Stardock Entertainment
15090 N. Beck Rd.
Plymouth, MI USA
Tel: 734-927-0677
Fax: 734-927-0678
jobs@stardock.com
Please use: Lead Game Designer for your Subject
www.stardock.com

Stardock® ENTERTAINMENT

Lead Game Developer Opening

Stardock Entertainment is looking for a motivated and talented Lead Game Developer to join their team in Plymouth, Michigan. They offer an opportunity to lead the technical development of the company and to work with in a team-oriented, informal and flexible work environment. This position involves handling the technical aspects of games development and requires excellent multitasking, organizational, communication, and team collaboration skills. In this position you will lead, coach and grow a highly motivated and well balanced team. As the technical lead for the studio, you'll need to have expertise in:

- C++ coding and systems design
- Monitoring performance, memory usage, planning/executing performance and memory optimizations
- Monitoring stability, troubleshooting and resolving issues
- Designing, documenting and communicating system architecture
- Establishing and enforcing coding standards
- Maintaining a schedule and hitting deadlines

About Stardock

Stardock Entertainment has been developing and publishing games for over two decades. They are best known for developing the Galactic Civilizations series and publishing the Sins of a Solar Empire series. Stardock thrives on individual excellence, adaptability, and fun, while maintaining a structured environment.

requirements

- Experience leading a team of programmers
- At least 5 years of game industry experience (preferably with turn based strategy games)
- Expert C++ coding and systems design experience
- Excellent communication and mentoring skills
- Excellent organization and time management skills
- Excellent delegation skills

GLASSEGG digitalmedia

Turning Concepts Into Reality

Glass Egg Puts Quality at the Heart of it's Game Outsourcing Service

Glass Egg Digital Media is an experienced game outsourcing company in Ho Chi Minh city, Vietnam. Started in 1995, Glass Egg has grown into a successful game outsourcing service provider specializing in 3D game assets creation. Up do date, we have expanded our specialty into other aspects of game development such as 2D concept art and mobile game development.

Our Culture of Integrity

We deliver what we say we're going to deliver, on time and on budget. We do not ever over promise. Making games is hard, and things almost never go as planned. We understand that a client needs a reliable partner who can be flexible and work with them in challenging situations. And in then end we are proud to play a part in the making of great games.

We strive to treat our people well and work every day to build an environment of freedom, responsibility, and respect for and trust of the people we work with. We have built a team of the most capable managers and senior producers/artists, having more than 13 years of experience. Overall, our staff has an average tenure of 7 years. A low employee turnover guarantees a strong and united team to enhance the efficiency of each project. In addition, we have two parallel QA system s, both for Art and Tech to ensure our products are of the best quality a game outsourcing company can provide.

Our Proven Performance Says it All

Production in Vietnam began in 1995 on a small scale with simple 2D background art and animation; and by 1997 our team was responsible for 100 percent of a game's production, including the game coding and programming. In other words, we have built up experience from many game outsourcing projects. Our portfolio includes successful AAA titles such as FORZA MOTORSPORTS I – II – III – IV, FORZA HORIZON, NIGHT AT THE MUSEUM 2, BATTLEFIELD 2, DIRT 1 – 2 – 3, LORD OF THE DRAGONS, THE CLUB, PURE... We have strong partnerships with big companies such as Microsoft, Activision Blizzard, Electronics Arts, Namco Bandai, Ubisoft, Codemasters and many others, who consider Glass Egg as a real help with their game outsourcing demands.

To date, Glass Egg has had a global customer base including many leading game companies in 4 continents: America, Europe, Australia and Asia.

Our Amazing Staff

We never forget that people are ultimately our most valuable resource. A fun, comfortable and efficient working environment is what we aim for. We succeed by remembering "quality" as one of our core values. And our existing customer base gives us the confidence to meet high demands of game outsourcing.

Glass Egg Digital Media
REE Tower, 17th Floor
9 Doan Van Bo Street
District 4
Ho Chi Minh City
70000, Vietnam
Tel: +84.8.3943 1389
Fax: +84.8.3943 1388
contact@glassegg.com
www.glassegg.com

core competencies

- 3D modeling and texturing
  - Vehicles
  - Environments
  - Props/Objects
  - Characters
- 2D concept art
  - Characters
  - Environments
  - Vehicles
- Mobile development
  - iOS
  - Android
- Publishing services
  - Vietnam
  - Some other South East Asian territories

Who We Are

With more than 100 million registered players, InnoGames is one of the world´s leading developers and providers of online games. Our products (Tribal Wars, The West, Grepolis, Forge of Empires and others) are available in more than 30 languages. Our headquarter is located in Hamburg, Germany. We also have local offices in South Korea and Brazil.

The Business Principle

Our users have the option of playing InnoGames products completely free of charge and without restrictions for as long as they want. We also offer players the possibility of enjoying added benefits in the game by purchasing fee-based premium features. At the same time, InnoGames places a high priority on providing entertaining gameplay even without a premium pricetag.

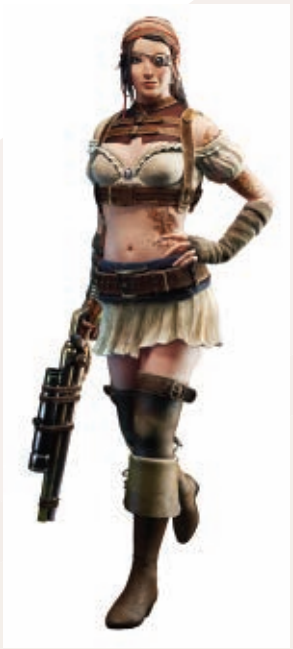
This principle has great advantages to the user. In contrast to traditional PC games, the product features here are already well-known before the player makes the decision to pay for added benefits - or not. The specific advantages offered by the premium features are also transparent. "What you see is what is what you get" is the underlying principle.

In our games, we focus on high long-term motivation. Our game Tribal Wars has been online for ten years now and continues to attract millions daily.

Partner with us

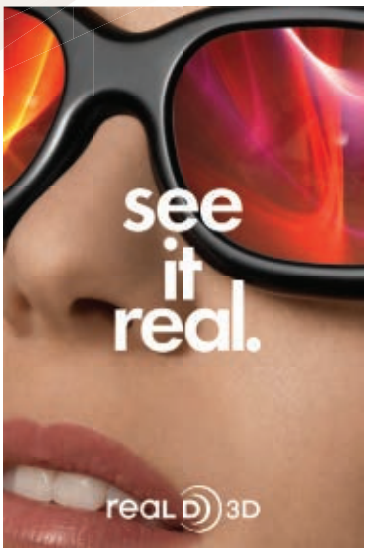
We have a good working network with trustworthy partners in all important game markets. As we are rapidly growing, we are looking for promising new partnerships. Are you a development studio experienced in mobile and browser games? Do you have high media competence and would like to cooperate with us? Or are you highly talented and want to boost our human resources? No matter your skill set - we are eagerly waiting for you. Just contact us

InnoGames GmbH
Harburger Schlossstr. 28
21079 Hamburg
Germany
+49 40 788 9335 0
info@innogames.de
www.innogames.com



We are looking for:

+ Business Partners
  - High quality development studios
  - Effective media cooperations
  - Partners for our in-house affiliate program
+ Talent to boost our human resources
  - Skilled mobile developers
  - Creative game designers
  - Artistic graphic designers
  - Experienced marketing, analytics and business development experts



RealD Inc.
100 N. Crescent Dr.
Beverly Hills, CA 90210
310.385.4000
reald.com/gaming

reaL D 3D

A Global Leader in 3D Technology

From Mars to the movies to the living room. RealD 3D technologies have enabled the exploration of distant planets, the projection of the biggest movies of all time and gaming in lifelike and immersive 3D. Developer of the world's most widely used 3D projection technology; provider of technologies to NASA and with technologies licensed to some of the world's biggest consumer electronics brands, RealD is focused on creating technologies that deliver premium 3D visual experiences on all content wherever audiences are.

Today's 3D—Perfecting the Visual Image

A great 3D experience requires both a high-quality display and high-quality content. Deficiencies in either can sour the experience. With display technology rapidly improving and moving toward glasses-free 3D, and the worldwide 3D display market expected to reach 226 million units in 2019[1], RealD has created technologies for game developers that remove some of the barriers in creating Stereo3D content.

Traditional Stereo3D rendering can cause distortions in how the image is presented to the player. RealD developed the 3D Game Developer Toolkit to correct these distortions and handle some of the more complicated camera configurations. Game developers are now free to use depth for creative gameplay and storytelling innovations.

3D Game Developer Toolkit

AutoCAM—Our proprietary AutoCam algorithm automatically adjusts camera separation to achieve the optimum depth in each scene. Scenes with a shallow depth require greater camera separation to create a deep sense of volume, while scenes with a large depth need less camera separation. AutoCAM analyzes each scene and calculates the optimum camera configuration in real-time.

Depth Budget Allocation—In traditional stereoscopic rendering solutions, objects in the foreground tend to have an exaggerated sense of volume while background objects appear more flat. Our DBA technology allocates the available depth evenly throughout each scene to create a more natural sense of volume. This greatly increases the immersion of every scene.

Already integrated into Unreal Engine 3 and UDK, RealD's 3D Game Developer Toolkit can bring a premium 3D experience to any game on any platform.

1 Display Search, Display Technologies and Market Forecast Report, October 2012

Integrate RealD technologies into any platform

- PC
- Consoles
- Mobile
- Arcade
- VR/Simulation

ADVERTISER INDEX

For more information visit www.jointhegamenetwork.com

gd Game Developer (ISSN 1073-922X) is published monthly by UBM 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 UBM 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: $59.95; all other countries: $69.95 (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.

M. WASTELAND & M. UNDERLAND

STACK TRACE AND THE DEATH OF A.A.A. DEVELOPMENT, PART 2

THE GRIPPING STORY OF OUR GAME DEVELOPMENT DETECTIVE COMES TO ITS SHOCKING CONCLUSION!

ILLUSTRATION: JUAN RAMIREZ

There's days when there's a murder, and then there's days when someone as big as A.A.A. Development is found dead. This was the second of those types of days.

But who would want to kill the man known as "Triple-A"? Not everyone liked him, to be sure, but who really stood to benefit if he got out of the way?

The first suspect I spoke with was Mobi LePhone. She was as flippant as ever, but I couldn't pin her with a motive. Then there was this new guy Mobi had tipped me off to—Mr. Waggle. What kind of name is that, anyway? I decided I needed to find out.

***

Instead of a doorbell, Mr. Waggle's apartment had a sign on it instructing me to put my hands in the air beside my head. I knocked instead. I heard someone fumbling around inside, and the door swung open to reveal a portly man with a mop of dark curls.

"Greetings! You must be the famous detective, Stack Trace!" he said. "You are hardcore? Casual? You are, perhaps, what they call midcore. But it is no matter. I, Mr. Waggle, like all kinds of people. Now, kindly step inside."

I tried to slip by him while he closed the door, but he jumped back.

"Please! Too close," he said. "I need at the least three feet of space around me at all times."

His brightly lit apartment was decorated with funky wall decals and oversize beanbag chairs in bright, primary colors.

"Nice place," I said, and settled into one of the beanbags.

"Actually, won't you stand up?" he said. "I cannot recognize your shape if you are sitting."

"Mr. Waggle—look, I just wanted to ask you about your relationship with Triple-A."

"We had an excellent relationship! So many synergies, you see. I was—wait, where did you go? You disappeared! I can't—oh, there you are. You've been there the whole time? That's strange. Anyway, as I was saying: Synergy!"

"Business is good for you if it's good for him? What's the future now that he's out of the picture?"

"Excuse me, but could you preface everything you say to me with 'Mr. Waggle'? Speak up, too."

"Mr. Waggle, are you saying you'll be out of business now?"

"I guess I am saying that! I'm, uh, sad. But surely you must know this: If anyone wanted Triple-A out of the picture, it was the one they call Bookface."

Bookface. Maybe he wanted to muscle in on Triple-A's business? Plausible enough.

"Mr. Waggle, thank you. I'll be on my way to see Bookface, then." I rocked myself out of the beanbag chair—those things are really hard to get out of once you sit in them—when suddenly I heard a faint but telltale screeching metallic sound emanating from the next room.

There was only one person I knew in all GameDev City who listened to dubstep.

***

Just as the sick drop started, I said, "Hey, Wags, be careful. You never know who this griefer will come at next."

"Don't worry, I think I'll be..."

He paused, and I saw worry crossing his face as I spun on my heel.

"I didn't start with 'Mr. Waggle,' Mister Waggle..."

"You what? I, uh, I guess I can sometimes understand—"

"That's alright. Because I should have been saying—Mister Development!"

I surged forward and yanked the toupee off his head. A choked peep escaped from his lips. There he was: Aaron Alexander Akbar Development—pale, sweaty, and a little worse for wear, but as alive as a forum on launch day.

"You faked your own death and then invented this bizarre identity of Mr. Waggle. Why?"

"No use hiding it any longer, Mr. Trace. I'd been gambling again. I kept putting more money into every bet, thinking the returns just wouldn't stop. Well, eventually people lost interest in me, and, well, I got hard up on cash. To escape my debts I figured I could reinvent myself, become someone new: a dapper man-about-town, a real hit with the kids and the ladies. I just wanted to appeal to the masses, okay?"

He collapsed in the nearest beanbag chair and sighed. "By the way, Stack... How did you know it was me?"

I was already on my way out, but I turned and pulled down my hat.

"It's simple, Mr. Development. I just followed all the dialogue trees."

***

Matthew Wasteland writes about games and game development on his blog, Magical Wasteland (www.magicalwasteland.com). Email him at mwasteland@gdmag.com. Magnus Underland writes about games and other topics at www.above49.ca. Email him at magnus.underland@gmail.com.

To view Part #1, visit: gdmag.com/blog

GDC 13

GAME DEVELOPERS CONFERENCE® 2013

SAN FRANCISCO, CA | MARCH 25–29, 2013 | EXPO DATES: MARCH 27–29, 2013

JOIN US FOR FIVE DAYS OF SESSIONS, TUTORIALS, BOOTCAMPS, AND ROUNDTABLE DISCUSSIONS ON A COMPREHENSIVE SELECTION OF GAME DEVELOPMENT TOPICS TAUGHT BY LEADING INDUSTRY EXPERTS.

learn

SUMMITS AND TUTORIALS
(MONDAY & TUESDAY)

- AI
- Free to Play Design & Business
- Game Narrative
- GDC Education
- Independent Games
- Localization
- QA
- Smartphone & Tablet Games

MAIN CONFERENCE SESSIONS
(WEDNESDAY–FRIDAY)

- Audio
- Business, Marketing & Management
- Game Design
- Production
- Programming
- Visual Arts
- Monetization (SPONSORED)
- Game Career Seminar (FRIDAY)

network

GDC Play
(TUESDAY–THURSDAY)

Business Center
(WEDNESDAY–FRIDAY)

Career Pavillion
(WEDNESDAY–FRIDAY)

inspire

INDEPENDENT GAMES FESTIVAL
(MONDAY–FRIDAY)

game developers choice awards
(WEDNESDAY)

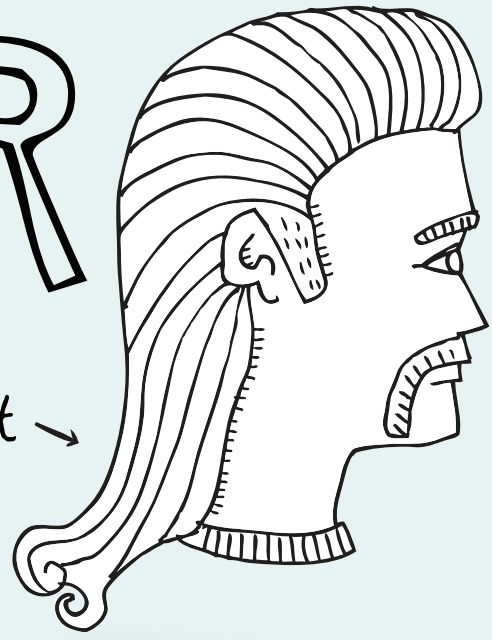
Expo Floor
(WEDNESDAY–FRIDAY)

VISIT GDCONF.COM FOR MORE INFORMATION.

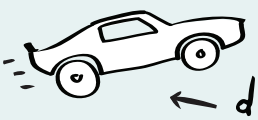
UBM Tech

THERE ARE LOTS OF WAYS TO BE AN EVEN

COOLER GAME DEVELOPER



like grow a mullet →



← drive a sweet old Camaro

or you can use **BINK VIDEO**.

You get an amazing, super =FAST video and audio codec - all in a simple, clean API. And Bink Video runs on **EVERY PLATFORM**!

BINK VIDEO

USING BINK VIDEO NOT ONLY MAKES YOU cooler, IT MAKES YOU rad!

RAD GAME TOOLS ©

www.radgametools.com
(425) 893-4300