REVISITING ANDROID +++ POSTMORTEM: HUNDREDS +++ INTRO TO USER ANALYTICS

UBM Tech

MAY 2013
VOLUME 20 NUMBER 05

gd

THE LEADING GAME INDUSTRY MAGAZINE

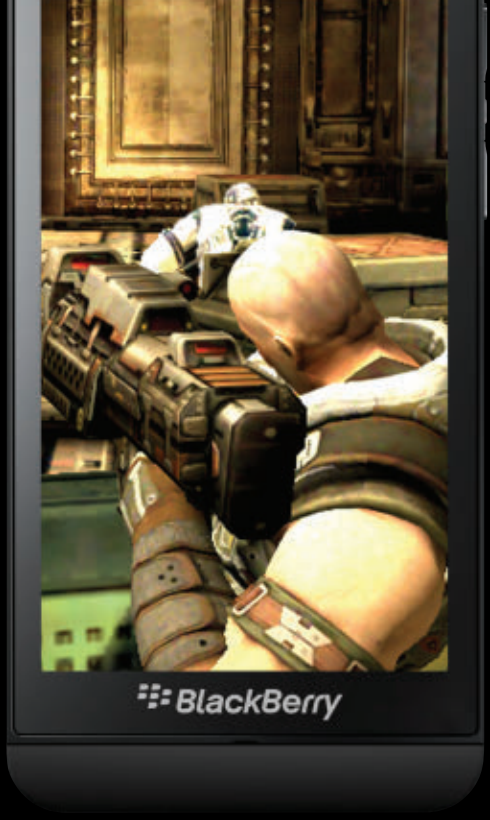
GAME DESTINATION: BLACKBERRY 10

It's where your game belongs.

Discover how you can create games
that keep them coming back for more.

BlackBerry®

BlackBerry® 10 offers a powerful and easy platform for game development. It's integrated with major development tools and leading game engines, including Unity, Marmalade and Shiva 3D. Plus, the leading BlackBerry 10 hardware produces a visually stunning and incredibly immersive gaming experience that really lets your masterpiece shine.



Get your game where it needs to be. Fast.

Users everywhere are hooked on the simplicity, elegance and blazing-fast performance that BlackBerry 10 delivers. They are enthusiastically snapping up amazing entertainment and apps to make their BlackBerry experience that much richer. All of this demand means that there has never been a better time for you to bring your game to BlackBerry.

It's easy to get started

By offering both native development tools and integration with the major development tools on the market, BlackBerry makes it simple for you to choose an option that works best with your individual skills and preferences. So you can develop your game faster and with the greatest flexibility.

POSIX

POSIX-based OS, support for openGL ES, OpenAL

16:9

Consistent form factor makes it easy for developers

Development tools using Microsoft Visual Studio and Eclipse EDT

Leading game engine and middleware support

Shadowgun image courtesy of SHADOWGUN, by MADFINGER. Alpha Zero images courtesy of Funkoi Ltd. Copyright 2013. All rights reserved.

Start mapping out your success
developer.blackberry.com/games

Visit the BlackBerry booth at the
Nordic Game Conference
Malmö, Sweden on May 22-24, 2013

Project Anarchy

by havok

FREE!

Full 3D Mobile Game Engine & Toolkit

Includes Havok Vision Engine together with access to Havok's indsustry-leading suite of Physics, Animation and AI tools as used in cutting-edge franchises such as The Elder Scrolls®, Halo®, Assassin's Creed®, Uncharted™ and Skylanders™.

- **+ Extendible C++ plugin-based architecture**
- **+ Comprehensive game samples and tutorials to get started**
- **+ Online community support with forums, Q&A and videos**
- **+ NO commercial restrictions on company size or revenue**
- **+ Upgrades for additional platforms, source and support avaialble.**

Create anarchy with us.

Developer contest coming soon!

YouTube

Sign up for the Project Anarchy newsletter on projectanarchy.com or join us on the interwebs for updates!

gd

003
CONTENTS_May 2013
VOLUME 20 NUMBER 05

Postmortem

Features

Departments

ASH HEAP OF HISTORY

THE COST OF DISRUPTION

Had I written this editorial two weeks earlier, I'd probably be writing about how enthusiastic I am to introduce our first-ever mobile-themed issue. In the past few years, mobile games have grown into a part of the industry no developer can afford to ignore, and the fact that *Game Developer* hasn't ever devoted an entire issue to the topic until now is rather shortsighted on our part.

Then I found out that *Game Developer's* parent company UBM Tech was axing all its print publications. That's right—if you haven't already heard, *Game Developer's* last official issue is the next one (June-July). Stick around for it; it's gonna be good.

Of course, the irony of announcing that we're ceasing publication in the issue that celebrates the new and exciting world of mobile game development is delicious, if rather bittersweet.

To those of you worried about your future in triple-A dev due to last month's salary survey, and those of you who were laid off because whatever you were working on didn't have enough future-friendly buzzwords to satisfy your management, we understand. We've got a support group going down by the bar, and we'll save you a seat.

CLOSING TIME In a sense, both the publishing industry and the game industry have experienced similar disruptive patterns from the rise of mobile computing platforms. On one hand, the fact that practically everyone carries around some kind of handheld, Internet-connected computer means that our potential audience has exploded. At any given moment, someone with a few spare seconds could whip out their phone and start playing your game or reading my articles. On the other hand, the design of these devices drastically changes the way people want to play or read; we want games to play in 30-second bursts and writing in 140-character chunks.

As creators, we know that there are truly great things you can do with short-form (well, more like microform, really) games and writing. But that isn't the work that inspired us to join this industry ourselves, and it can be hard to embrace wholeheartedly a new aspect of the medium knowing that the work we're doing isn't necessarily the kind of work that personally engages us. We can dig our heels in and resist the change as best we can, of course. It's a matter of pride; we *just* got the chance to make something good, and now it's being supplanted by something else.

EVERY NEW BEGINNING... When news of LucasArts's exit from game development hit the wire, I had a chat with former *Game Developer* EIC Brandon Sheffield and former Gamasutra news director Frank Cifaldi about why everyone was mourning the loss of the studio's legacy despite the fact that all the games we mourned hadn't seen any love for at least a decade. We don't miss STAR WARS: THE FORCE UNLEASHED; we miss MONKEY ISLAND and FULL THROTTLE and X-WING VS. TIE FIGHTER and GRIM FANDANGO. And, weirdly enough, we miss them even if we haven't played each of those games through; I only played a few of those games myself, but I was just as down as Frank was on that day, and he's probably going to name his firstborn Guybrush Cifaldi or something ridiculous like that. I think perhaps the best explanation comes from a webcomic called Achewood, on the strip for the day that Michael Jackson died:

"He was your Elvis, and when your Elvis dies, so does the private lie that someday you will be young once again, and feel at capricious intervals the weightlessness of a joy that is unchecked by the injuries of experience and failure. In other words, you two died a bit today. Welcome to the only game in town."

...COMES FROM SOME OTHER BEGINNING'S END There is pride, and then there is denial. The reality is that game developers who ignore mobile, or indies, or any other major trend in games do so at the risk of their careers and their relevance to the medium. (Same goes for editors.) So like Autobots, we transform and roll out, knowing that the job ahead of us is not to remake the works that inspired us to enter this industry in the first place, but to learn new things so we can make new things—things that might just be the Elvis for someone else.

Welcome to the only game in town.

-Patrick Miller
Editor, *Game Developer*
@pattheflip

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*
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
Alexandra Hall, Steve Stopps, Adam Saltsman, Nate Ralph, Dean Ellis, Steve Theodore, Soren Johnson, David Kanaga, Kim Pallister, Brandon Sheffield, Matthew Wasteland, Magnus Underland, Anders Drachen, Alessandro Canossa, Magy Seif El-Nasr
ADVISORY BOARD
Mick West Independent
Brad Bulkley Microsoft
Clinton Keith Independent
Bijan Forutanpour Qualcomm
Mark DeLoura Independent
Carey Chico Independent
Mike Acton Insomniac
Brenda Romero Loot Drop

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
Robert Steigleider *robert.steigleider@ubm.com*
t: 516-562-5134

REPRINTS

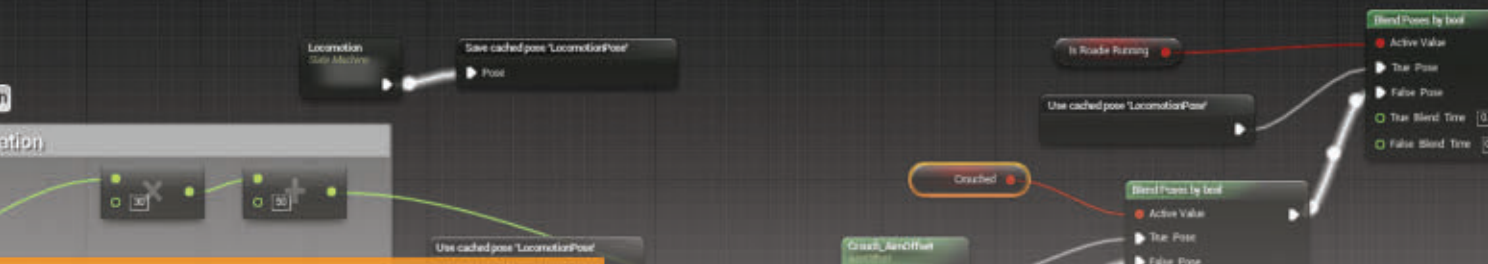
WRIGHT'S MEDIA
Jason Pampell *jpampell@wrightsmedia.com*
t: 877-562-5972

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

UNREAL ENGINE NEWS



What's in a Blueprint?

Unlocking Epic Games' new Unreal Engine 4 Blueprint visual scripting system

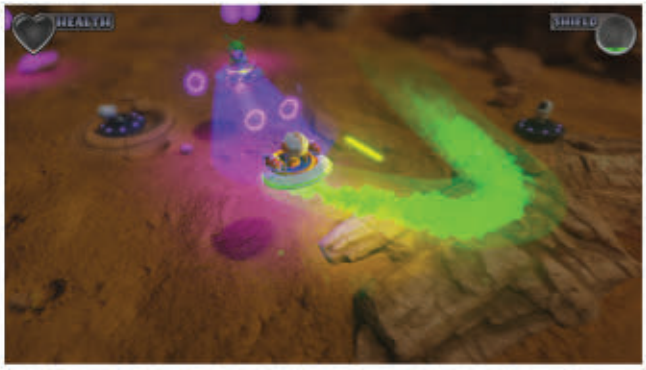
Legions of developers rely upon Unreal Engine 3's Unreal Kismet, which is used to quickly build gameplay prototypes and level events. With Unreal Engine 4 Epic introduces Blueprints, a revolutionary new visual scripting system that replaces Kismet and provides a massive leap in functionality and productivity.

Blueprint visual scripting empowers artists and designers like never before by providing access to low level engine functions and the ability to rapidly prototype without having to write a single line of code.

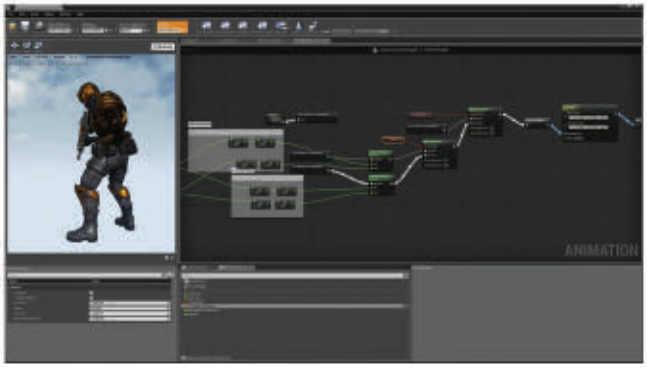
Supported by built-in debugging tools and enhanced by Unreal Engine 4's brand new interface, Blueprints deliver all the power of Kismet along with the ability to visually script reusable components for gameplay, AI, player controls, geometry creation and numerous other features. These components can be used as pervasive parts of the world, so when a Blueprint is updated, the change affects all parts of the game where the Blueprint is present.

Blueprints can be utilized for gameplay behavior, animation blending, level building and design, and as mentioned before, object construction. Epic is only scratching the surface of what Blueprints can do so far. The only limitation is one's imagination, and it will be exciting to see how Unreal Engine 4 licensees unleash their creativity with the new toolset.

For additional info and the latest news about Unreal Engine, please visit unrealengine.com.



Above: Epic recently presented the "Infiltrator" real-time demo to highlight Unreal Engine 4's latest features. The vehicle conveyor apparatus shown here is a Blueprint. **Center**: Everything in this alien hovership game, which was created by Epic's principal artist Shane Caudle in his spare time, is made entirely with Blueprints. This includes the HUD, input handling, animation, gameplay controls, enemy AI, character setup, weapons, projectiles, and effects. **Bottom**: Unreal Engine 4's new persona animation system provides state support and animation blending. Blueprints enable quick adjustment of speed, transitioning between states, fine-tuning a character's pitch, yaw and much more.



Come see Epic at upcoming industry events: **Electronic Entertainment Expo (E3)** (June 11-13, Los Angeles, CA), **Develop Conference** (July 9-11, Brighton, UK), **ChinaJoy** (July 25-27, Shanghai, China).
Email licensing@epicgames.com for appointments and sign up for our newsletter at unrealengine.com.

INDIE STORE FOR INDIE DEVS

NEW PAY-WHAT-YOU-WANT STOREFRONT SERVICE MAKES IT EASY FOR INDIE DEVS TO SELL DIRECTLY TO PLAYERS

The pay-what-you-want business model has attracted plenty of interest across the Internet over the last few years, as creative types of all kinds have struggled to find ways to turn their passion projects into something that can also pay the bills. But how is an indie dev supposed to take the time to build their own PWYW storefront on top of actually making a game? Independent game dev Leaf Corcoran built his own service called itch.io (www.itch.io) in order to sell a few games he made for the Ludum Dare game jam, and thought others might like to use it too.

PATRICK MILLER: What was the inspiration for itch.io?

LEAF CORCORAN: The idea came to me fairly recently. I had completed a few games for Ludum Dare and I wanted to put up mini web sites for each one with a download link and some screenshots. Although I have experience making web sites, doing something like this is kind of a pain because there is a good amount of manual work. So I thought just a simple game-page generator would be nice.

I put the idea off, but a few weeks later I was thinking about how I keep on seeing Bandcamp music pages everywhere, and from what I could tell it seemed like they were doing pretty well with a pay-what-you-want model. So I took a week off from my day job and did the majority of the programming then, though it took about five more weeks until I finally released it.

PATRICK MILLER: Why get behind pay-what-you-want?

LEAF CORCORAN: I really can't speak from experience on whether pay-what-you-want is a good model for games—this whole thing is a big experiment to me. I'm sure there are a lot of indie game developers out there who have made games that just sit around, and they probably never considered getting money for them, but with a system like this in a few clicks you can set it up so if someone likes your game they can give you some money. I think that's one of the most profound things about the site.

PATRICK MILLER: How much work does it take for an individual dev to start their own storefront? What's your cut?

LEAF CORCORAN: It's very easy to get started; it doesn't take long to make a game page and you can do it immediately after registering an account. I think it's really important to have a fast and simple way to get your game ready to be distributed. The game pages are all about the game. There are no logos or links for itch.io (except for a tiny link on the bottom). I think that's pretty important. When I looked around for other similar marketplaces, they were all plagued with a bunch of crap that didn't add to the game.

With each game you can also upload files. The files can be of any type, so you are free to give out other goodies with your game like soundtracks, art books, or whatever you can think of.

By default, anyone can download the files, but if you set a minimum price you can restrict download access. A minimum price of 0 means that the user will be prompted to pay, but they can skip to the downloads if they wish. The download URLs are restricted such that it's not possible to share a direct link to a file, so everyone gets the opportunity to pay before they download. There is no DRM or download client; the files are downloaded right from a browser and they are kept exactly as you upload them. This means you can upload games for any platform. Also, I try to give the game developers insight to their pages—view, download, and purchase statistics, nice graphs, and so on. Currently all payments are handled by PayPal. I take a 10% cut of all transactions as a fee for the service.

PATRICK MILLER: How has reception been so far?

LEAF CORCORAN: Pretty good. I've posted it on a handful of game developer message boards, along with Facebook, Google+, and Reddit. I've still got a lot of marketing to do, though, before I'm satisfied. I get a lot of feature suggestions, which is pretty cool—it's one of the reasons why I released it as early as possible. About 80 people have signed up so far, with 33 games created and five dollars and one cent of transactions. Small beginnings, but things are looking good so far.

PATRICK MILLER: What's next in the pipeline as far as new features?

LEAF CORCORAN: I've got a lot of ideas; here are some of the major ones: I only support downloadable games right now, but I'm coming up with ideas for hosting online games (Flash, Unity, HTML5). Also, I want to add other online payment providers besides PayPal, and build an "explore" page so people can browse the games database.

—Patrick Miller

FAREWELL LUCASARTS

DEVS SAY GOOD-BYE TO A BELOVED STUDIO

DAY OF THE TENTACLE.

In early April, the Walt Disney Company announced it was shutting down all of LucasArts's internal game development and turning it into a licensing house. Faced with this news, *Game Developer* sister publication Gamasutra asked game devs to answer the following question: *What is it about the classic LucasArts adventure games that makes them timeless?* Here is a selection of the answers we received.

"They often broke the fourth wall and made you feel like you were in on the joke. As if the joke was 'Can you believe we get to make these things?' It was exhilarating and inspiring. Even today, my fantasy of what game development nirvana feels like stems from my experience playing those games, and the insinuation that they were created in the most liberating and creative environment on earth."

—Mike Mika (Other Ocean Interactive)

"What I love about the classic LucasArts adventures as a game-industry person is that it seems like every single person in the game industry has played them. So you can use any of the games as shorthand when discussing something ('We need a Manny-like character for this'), and more importantly, you can instantly learn a lot about someone by what LucasArts adventures they like most ('OK, you're a MONKEY ISLAND guy? Got it. Still focused on putting the hamster in the microwave in MANIAC MANSION? Cool. You like THE DIG?! Right on...'). Because everyone has played them, they're basically a Rorschach test for people in the game industry at this point."

—Chris Charla (Microsoft Studios)

"LucasArts writers and designers like Ron Gilbert and Tim Schafer brought to the table a wonderful knack for character and dialogue, and I think the reason people still talk about those early titles today is that we all have favorite scenes that we still remember. They also achieved a kind of unstudied greatness that comes from not taking yourself too seriously. MONKEY ISLAND II actually ended gameplay with a long list of things you could go out and do other than play video games. Who does that now?"

—Peter McConnell (LucasArts alum)

"They were built on a winning combination of low-stress mechanics and propelled by genuinely good writing. Many of the characters had heart and soul, the imagined worlds they inhabited were crafted with an impressive attention to detail, and in many cases the personalities and some aspect of the creators came through. Each of the mid-'90s LucasArts adventure video games is memorable on its own, but taken together they represent a studio's glorious golden age that, in a fate similar to Atlantis (sorry), seemed to suddenly and cataclysmically sink beneath the ocean waves in the late '90s."

—Craig "Superbrothers" Adams

"Honestly, I don't even know where to begin; they had so many titles that stand out. Their adventure games basically defined an era, with MANIAC MANSION, DAY OF THE TENTACLE, ESCAPE FROM MONKEY ISLAND, FULL THROTTLE, and GRIM FANDANGO. INDIANA JONES AND THE FATE OF ATLANTIS is probably one of my favorite adventure games of all time. Indiana Jones and the adventure game genre go together... well, like fedoras and bullwhips. And for me, FATE OF ATLANTIS delivered not only one my favorite adventure stories but also one of the best Indiana Jones stories ever. Certainly better than THE CRYSTAL SKULL. Sorry, George."

—Mark Rubin (Infinity Ward)

"In adventure games, verbs are mechanics and writing is gameplay. The two can live in harmony. LucasArts made some of the best—by turns thrilling, funny, strangely morbid—and I will always be grateful for that."

—C.J. Kershner (Ubisoft Montreal)

"Classic LucasArts adventures are timeless because their influences are timeless. Frankenstein and film noir and buddy comedies and teen movies, classic pulpy sci-fi and swashbuckling movie serials crossed with irreverent, real, believable characters living in outrageous worlds. I loved FULL THROTTLE's neo-noir before I knew what film noir was. I loved DAY OF THE TENTACLE's Bernard, Hoagie, and Laverne archetypes before I'd ever seen the teen-movie source material. But regardless of the specific references and inspirations, classic LucasArts adventures are timeless because great, clever, earnest, memorable, human writing is timeless, and that is the foundation on which all those great games were built."

—Steve Gaynor (The Fullbright Company)

"We'd spend months thinking about our games... brainstorming with the other brilliant designers, refining, reworking, revamping, tossing out the parts that didn't work (or the entire concept) and starting again. One of our edicts was 'don't ship shit' and we wanted to make sure we never did."

—David Fox (LucasArts alum)

—Frank Cifaldi

This article is an excerpt of an article originally posted on Gamasutra. You can find the full-length version here: http://bit.ly/YXJG2L

Developer: Mahdi Bahrami and Moslem Rasouli
Release date: September 22, 2012
Development time: 2 months
Development budget: $0
# of lines of code in the game: About 2,000
A fun fact: After putting the first version of the game on my blog, many people uploaded gameplay videos of their experience of playing Farsh to YouTube. For the final version of the game, I used those videos for gathering design feedback.

FARSH, a Persian-themed puzzle game with a clever carpet-unfurling mechanic, was a Student Showcase Finalist in the 2013 Independent Games Festival. It's just the latest game from Mahdi Bahrami, a young Iranian developer with a knack for concocting original gameplay concepts, who's studying at the NHTV Breda University of Applied Sciences in the Netherlands. We checked in with Mahdi to discuss Farsh and the ups and downs of developing while Iranian.

ALEXANDRA HALL: ***What got you started creating games?***
MAHDI BAHRAMI: I started learning programming about eight years ago with QBasic and then Visual Basic. After a few years, I found XNA. Making a game called PERSIAN TETRIS was one of my first steps to become a game developer. I'm not a longtime video game player. What I remember from childhood is sitting next to my brother and watching him play games. I think that helped me have a better perspective on video games.

ALEXANDRA HALL: ***It's interesting that you're not a longtime player. What drives you to want to create games?***


PHOTO CREDIT Joram Walters.

MAHDI BAHRAMI: Yeah, I'm not a longtime player, but being able to create my own games was a dream for me. Learning programming was the key to achieving my dream.

ALEXANDRA HALL: ***Several of your games (FARSH, NEGA KON!, BO) seem simple at first, but get surprisingly complicated. Where do you find your ideas?***
MAHDI BAHRAMI: When I'm going to make a new game, before I start to work on it, I think a lot about the main idea behind it. I don't start working on a game if I think it's not different enough.

ALEXANDRA HALL: ***So you are only interested in making original games?***
MAHDI BAHRAMI: Yeah, even if original games are not always successful, I won't stop trying to design unique games.

ALEXANDRA HALL: ***Are there many other Iranian/Persian game makers?***
MAHDI BAHRAMI: There are so many talented artists in Iran for any kind of art. I know many game developers in Iran. Farhoud Farmand, Dead Mage Studio, and Sourena Game Studio have potential to make great games.

ALEXANDRA HALL: ***Would you want to collaborate with other game creators? I noticed Moslem Rasouli created the excellent music in FARSH.***
MAHDI BAHRAMI: Yeah, collaborating with Moslem made FARSH much better than what it was in the beginning; he is an example of the young talented artists living in Iran, and of course I [would] like to collaborate with other talented people.

ALEXANDRA HALL: ***What are the positives and negatives of being a game developer in Iran?***
MAHDI BAHRAMI: I feel good about being an Iranian game developer. I have the opportunity to become familiar with many details of Persian culture. My games are somehow inspired by my culture and it helps me to make more original games. On the other hand, I have many problems just because of my nationality. Just as an example: FARSH is one of the IGF finalists, but I was not able to travel to the U.S. because my visa application was rejected. This is only one of the difficulties I have.

I'm not able to sell my games officially as an Iranian on the App Store, Xbox Live Indie Games, etc. Also, as an Iranian I'm not allowed to have a PayPal account. And I'm not allowed to submit my games to some of the competitions.

ALEXANDRA HALL: ***Have you considered emigrating elsewhere?***
MAHDI BAHRAMI: I'm currently studying in the Netherlands (it's about seven months since I came here) but those difficulties are still with me, because I'm still Iranian. As I said before my visa application for the U.S. was rejected recently, even though I am living in the Netherlands.

ALEXANDRA HALL: ***Do you have any message you'd like to share with your game-making peers around the world?***
MAHDI BAHRAMI: I would like to ask them to visit my blog and contact me if they can help me to overcome the difficulties I mentioned.

ONCE A PIRATE, ALWAYS A PIRATE

RON GILBERT RETURNS TO THE HIGH INDIE SEAS

Ron Gilbert, creator of classic LucasArts games such as THE SECRET OF MONKEY ISLAND, recently left Double Fine after completing The Cave. Now independent, he's collaborating on an iOS game with Clayton Kauzlaric, SCURVY SCALLYWAGS IN THE VOYAGE TO DISCOVER THE ULTIMATE SEA SHANTY: A MUSICAL MATCH-3 PIRATE RPG.

ALEXANDRA HALL: ***Here you are independent again, and making funny games. Good feeling, or best feeling?***
RON GILBERT: I don't know if I'd say it is a good or bad feeling, it's just a different feeling. I had a lot of fun working at Double Fine on THE CAVE and wouldn't trade it for anything. But I've always liked working on very small teams as well, and it will be fun to do that again. I find that 90% of my gaming these days happens on my iPhone and iPad, so I'm looking forward to being able to focus on those platforms.

ALEXANDRA HALL: ***How do you stay afloat in this economy?***
RON GILBERT: Stay small and keep focused and build stuff that has a unique twist to it. Grassroots PR/marketing is key for small indie developers, and the ones that do it well enjoy some good success. "Build it and they will come" is a lie. :-)

ALEXANDRA HALL: ***THE CAVE has been percolating since before Maniac Mansion. Did it change much since that early conception?***
RON GILBERT: A lot changed. The original idea was not an adventure game and there were only three characters. When I started working on it again a few years ago in what became THE CAVE, I added the seven characters because I have always wanted to revisit that from MANIAC MANSION.

AH: ***It's nice to see prominent game designers of the 1980s and 1990s returning to the spotlight. What's fueling this? Kickstarter, nostalgia...?***
RON GILBERT: There is definitely a maturing of the industry, and by maturing I don't mean we're all getting old. The base of people playing games has exploded in the last five or six years, due to Facebook and the iPhone and Android, and they aren't all looking to play the same type of game and that's creating a lot more diversity. Some of the older genres are seeing a comeback. It's all new to a

different generation of players. I do think nostalgia plays a role, and Kickstarter is great for addressing that need.

ALEXANDRA HALL: ***Disney's control of the LucasArts IPs is one of the major factors stopping you from working on your classic properties. Is retaining IP one of the most important things for developers? Has your stance on this evolved?***
RON GILBERT: Controlling and owning your IP is key. If you can avoid it, never give up IP. It's a myth that publishers will work harder to promote the game if they own the IP. Give publishers an exclusive window and first right of refusal and all that, but don't give up your IP. I've been making games for close to 30 years and from MANIAC MANSION to MONKEY ISLAND to PUTT-PUTT and PAJAMA SAM to DEATHSPANK and on, I don't own any of it and that's been a huge regret.

ALEXANDRA HALL: ***LucasArts is essentially no more. Can you share some thoughts on this end of an era?***
RON GILBERT: When Disney bought Lucasfilm last year, its being shut down was inevitable. As much as I like to think they paid $4 billion to get the rights to Monkey Island, the fact is Disney bought Lucasfilm for the *Star Wars* movies and that's it. Everything else was just an accounting rounding error. My hope is that I can get the rights to MONKEY ISLAND back, and maybe Gary [Winnick] and I can get MANIAC MANSION back. That would be fun. MONKEY ISLAND KART RACING. You know you want it. It's very sad that they are gone. I grew up there and even though I hadn't worked there since 1992, it always felt like home to me.

P.S. I was kidding about MONKEY ISLAND KART RACING...but it would probably sell really well. Sad, but true.

Who Went Where

- Silvia Seibert has been appointed CFO of Slightly Mad Studios, the creators of the NEED FOR SPEED SHIFT games.
- Rod Fergusson, the former GEARS OF WAR producer who joined Irrational Games to help finish BIOSHOCK INFINITE, is moving on now that Infinite has shipped to wide acclaim. No word yet on where he'll end up.
- Giordano Contestabile, PopCap's longtime franchise manager for BEJEWELED, has left to join Tilting Point Media, where he'll work to fund both established indie teams and new IP for mobile platforms.

New Studios

- Melbourne-based Loveshack Entertainment is the new of home of three senior developers from EA's Firemint development studio. The team's initial project, FRAMED, is described as "an innovative narrative-based puzzle game in which players must rewrite the story."
- WB Games San Francisco is a new studio that will focus on developing free-to-play mobile and browser-based games.
- Ovosonico, a new studio founded by SHADOWS OF THE DAMNED director Massimo Guarini, is working on a new IP in partnership with Sony Computer Entertainment Worldwide Studios Europe.
- Pascale Audette, former studio head of Disney Online Studios, and Lance Priebe, cofounder of Disney's CLUB PENGUIN virtual world, recently founded Hyper Hippo Productions, which will focus on creating children's games for digital platforms. Their first title will be MECH MICE.

CO-LOCATED WITH ADC 13
APP DEVELOPERS CONFERENCE

LEARN. NETWORK. INSPIRE.

GAME DEVELOPERS CONFERENCE® NEXT

LOS ANGELES, CA
NOVEMBER 5–7, 2013
EXPO DATES: NOVEMBER 5–6

2013

GDCNEXT.COM

UBM Tech

THE 13TH ANNUAL GAME DEVELOPERS CHOICE AWARDS

The 13th Annual Game Developers Choice Awards took place on Wednesday, March 27, 2013. In addition to the winners of the standard categories, which were decided through the votes of game developers, the recipients of three special honors were selected by the Game Developers Choice Awards advisory board.

LIFETIME ACHIEVEMENT WINNER:

Dr. Ray Muzyka and Dr. Greg Zeschuk

The Lifetime Achievement Award recognizes the career and achievements of a developer who has made an indelible impact on the craft of game development and games as a whole.

PIONEER AWARD WINNER:

Steve Russell

The Pioneer Award celebrates those individuals who developed a breakthrough technology, game concept, or gameplay design at a crucial juncture in video game history, paving the way for the myriads who followed them.

AMBASSADOR AWARD WINNER:

Chris Melissinos

The Ambassador Award honors an individual or individuals who have helped the game industry advance to a better place, either through facilitating a better game community from within, or by reaching outside the industry to be an advocate for video games and help further our art.

AUDIENCE AWARD WINNER

DISHONORED (Arkane Studios/Bethesda Softworks)

GAME OF THE YEAR

JOURNEY (Thatgamecompany/Sony Computer Entertainment)

DISHONORED (Arkane Studios/Bethesda Softworks)
THE WALKING DEAD (Telltale Games)
MASS EFFECT 3 (BioWare/Electronic Arts)
XCOM: ENEMY UNKNOWN (Firaxis Games/2K Games)

INNOVATION AWARD

JOURNEY (Thatgamecompany/Sony Computer Entertainment)

MARK OF THE NINJA (Klei Entertainment/Microsoft Studios)
FTL: FASTER THAN LIGHT (Subset Games)
THE UNFINISHED SWAN (Giant Sparrow/Sony Computer Entertainment)
ZOMBIU (Ubisoft Montpellier/Ubisoft)

BEST AUDIO

JOURNEY (Thatgamecompany/Sony Computer Entertainment)

HOTLINE MIAMI (Dennaton Games/Devolver Digital)
SOUND SHAPES (Queasy Games/Sony Computer Entertainment)
ASSASSIN'S CREED III (Ubisoft Montreal/Ubisoft)
HALO 4 (343 Industries/Microsoft Studios)

BEST DEBUT

Subset Games (FTL: FASTER THAN LIGHT)

Humble Hearts (DUST: AN ELYSIAN TAIL)
Polytron Corporation (FEZ)
Giant Sparrow (THE UNFINISHED SWAN)
Fireproof Games (THE ROOM)

BEST DOWNLOADABLE GAME

JOURNEY (Thatgamecompany/Sony Computer Entertainment)

THE WALKING DEAD (Telltale Games)
SPELUNKY (Derek Yu/Andy Hull)
TRIALS: EVOLUTION (RedLynx/Microsoft Studios)
MARK OF THE NINJA (Klei Entertainment/Microsoft Studios)

BEST GAME DESIGN

JOURNEY (Thatgamecompany/Sony Computer Entertainment)

DISHONORED (Arkane Studios/Bethesda Softworks)
MARK OF THE NINJA (Klei Entertainment/Microsoft Studios)
SPELUNKY (Derek Yu/Andy Hull)
XCOM: ENEMY UNKNOWN (Firaxis Games/2K Games)

BEST HANDHELD/MOBILE GAME

THE ROOM (Fireproof Games)

GRAVITY RUSH (SCE Japan Studio/Sony Computer Entertainment)
HERO ACADEMY (Robot Entertainment)
SOUND SHAPES (Queasy Games/Sony Computer Entertainment)
KID ICARUS: UPRISING (Sora/Nintendo)

BEST NARRATIVE

THE WALKING DEAD (Telltale Games)

SPEC OPS: THE LINE (Yager Entertainment/2K Games)
MASS EFFECT 3 (BioWare/Electronic Arts)
DISHONORED (Arkane Studios/Bethesda Softworks)
VIRTUE'S LAST REWARD (Chunsoft/Aksys Games)

BEST TECHNOLOGY

FAR CRY 3 (Ubisoft Montreal/Ubisoft)

PLANETSIDE 2 (Sony Online Entertainment)
HALO 4 (343 Industries/Microsoft Studios)
CALL OF DUTY: BLACK OPS II (Treyarch/Activision)
ASSASSIN'S CREED III (Ubisoft Montreal/Ubisoft)

BEST VISUAL ARTS

JOURNEY (Thatgamecompany/Sony Computer Entertainment)

BORDERLANDS 2 (Gearbox Software/2K Games)
FAR CRY 3 (Ubisoft Montreal/Ubisoft)
DISHONORED (Arkane Studios/Bethesda Softworks)
HALO 4 (343 Industries/Microsoft Studios)

REVISITING ANDROID

TAKING THE PULSE OF ANDROID GAME DEVELOPMENT

How's Android doing these days? From the consumer standpoint, Android tablets and smartphones appear to be gaining a little more ground on iOS every year. From the developer perspective, however, early reports of widespread piracy, painful device fragmentation issues, and the perception of a rather tightfisted audience scared many devs away. *Game Developer* caught up with devs from industry giants, indie powerhouses, small studios, and individual developers to see how the Android platform is treating them in 2013.

GREE

Ken Chiu (SVP of social games), Anil Dharni (SVP of studio operations), Andy Keidel (VP of engineering)

Prior dev background (platforms): iOS, Android, Facebook

Shipped Android titles: CRIME CITY ANDROID, MODERN WAR ANDROID, JACKPOT SLOTS ANDROID

Preferred toolset: Open to all tools depending on the needs of the game. Important to have a strong awareness of all technologies.

Is fragmentation still a major issue for you? Which devices do you target?

Ken Chiu: It definitely is not a worry but is absolutely a consideration. Android is a huge mobile platform and it is important for us to make sure we have a big presence there and ensure we are always supporting the users and their needs.

Andy Keidel: Our goal is to ensure compatibility and high performance of our titles with all Android devices, which means we are constantly having to think about different specs and make sure that our game features and mechanics aren't limited by anything around the Android hardware. Our biggest consideration is really making sure that we have the right specs to best take advantage of smartphone hardware and separately the tablet hardware.

Do you have any tips for optimizing the Android dev process?

Andy Keidel: We've come up with several best practices to improve our Android development workflow. To handle the many variations in screen sizes, we specify UI dimensions for the most common screen specs, and then match and scale the UI dynamically for all other screens. Another key to efficient development workflow is a solid understanding of the lifecycles of the Android application, activities, services, etc. While we have found most Android-specific features to be helpful in structuring our application codebase, we developed a custom approach to our in-memory game database to provide easier support for live content updates, which we perform several times per week.

How have your games sold on Android?

Ken Chiu: Our games have done really well on Android platforms. All of the titles we have released on Google Play—MODERN WAR, CRIME CITY, JACKPOT SLOTS—have been pretty consistently in the top-30 grossing since their launch (which for something like MODERN WAR, means it's been almost a year). So we are seeing tremendous success on Android devices.

Which app stores do you support? How do your Android sales compare to your sales on other platforms?

Ken Chiu: We support iTunes, Google Play, and Amazon. We have seen tremendous success on all of the stores.

Anil Dharni: We have always worked exceptionally hard to create game experiences that are exciting for gamers, and as such have always seen strong retention rates, strong support of the titles, and great IAP revenues. Our users have been proven to be extremely engaged on both iOS and Android, showing us that there is a very even divide, and we don't see one OS showing higher success than the others. We plan on continuing to support them both and make sure our games have long lifetimes.

Overall, have you found Android dev to be worth the extra work? Are you looking into other mobile platforms?

Ken Chiu: Absolutely. Android is one of the biggest and fastest-growing markets right now. Android phones and tablets are

breaking out all over the world and are most definitely a power to keep an eye on. That being said, I think the key for us has been balance and making sure we always support both iOS and Android. As for other platforms, right now we really look to only those two (Android and iOS), but we are always keeping our eye on other markets.

SPRY FOX

David Edery (CEO), Ryan Williams (director of danger)
Prior dev background (platforms): Web, Steam, Android, iOS
Shipped Android titles: STEAMBIRDS, TRIPLE TOWN
Preferred toolset: Unity

Is fragmentation still a major issue for you? Which devices do you target?

David Edery: It is not a major worry for us. Fragmentation is a fact of life. We deal with it, and we're a very small company with an extremely small customer-support team. TRIPLE TOWN was developed by a single engineer for both iOS and Android. If we can handle it, anyone can. Our QA library is mainly comprised of Nexus devices, the more popular Samsung devices, and a couple of Kindle Fires.

Do you have any tips for optimizing the Android dev process?

Ryan Williams: I've found the biggest boost in development productivity comes from reducing the time between writing code and experiencing the results of the code in-game.

For Unity, which is the engine we've used for mobile development, this means taking advantage of the hot code reloading functionality. It's not well-documented or supported, but if you are playing your game in the Unity Editor and you save a change to the code, Unity will compile the new code and start using it within a few seconds. It's awesome to use this to work on procedural generation, because you can simply set your generation to repeatedly loop in one screen, and edit your code in the other, and see the changes in action right away. Bugs that take a while to reproduce become much faster to solve: Play the game until you get to the point that the bug appears, and then start hacking on the code.

This doesn't come for free, though; the process by which the editor does the reloading involves a custom serialization of every game object, and not every data type is supported by this serialization. Specifically, dictionaries and generic user-defined classes do not work with hot reloading, and once you go through a hot reload cycle, they become null. The hot reloading takes place entirely outside of your code and control, and the only way your code can detect that a hot reload has happened is that all the unserializable member variables are suddenly null. If you want to use hot reloading as part of your development process, you have to either refrain from using unserializable types entirely, or find some workaround. I really wanted to use dictionaries, so I wrote a small class that copies the contents of the dictionary to a pair of lists (which are serializable), and restores the dictionary from the lists when it's found to be null. This adds a fair bit of boilerplate to each dictionary's declaration, so I feel that it could be improved further, but it definitely works.

How have your games sold on Android?

David Edery: We've done very well on Android. TRIPLE TOWN has been downloaded over four million times, and Google Play is the top-selling platform for TRIPLE TOWN. I'm not sure off the top of my head what exactly our conversion rate is there relative to iTunes and Amazon's Appstore, but it is relatively healthy. Amazon has been a bust for us, unfortunately.

Overall, have you found Android dev to be worth the extra work? Are you looking into other mobile platforms?

David Edery: Yes, it's worth it. We're always considering other platforms, but we are not developing for other mobile platforms.

ANIMOCA

David Kim (CEO)

Prior dev background (platforms): iOS, Web, PC, consoles

Shipped Android titles: STAR GIRL, PRETTY PET SALON, MY CAR SALON, ROBO5, PRETTY PET SALON SEASONS, LORD OF MAGIC, PRETTY PET TOY STORE, PIG RUSH, MY CAR SALON 2, PET CAFÉ, PRETTY PET TYCOON, TOP MODELS: SPORTS EDITION, and dozens of others

Preferred toolset: Depends on the app in question, but all the way from Unity to Cocos2d-x, plus our own proprietary tools

Is fragmentation still a major issue for you?

David Kim: Android fragmentation is an exaggerated issue. It has never been a worry for us, and we tend to test for greater compatibility than most developers. Different screen sizes are a manageable issue as long as you support the basic Android form factors and let the OS handle scaling. Processor and graphics power do limit which games can be played on a device, but people who buy ultra-cheap low-end phones are probably not expecting to play too many games on them. That said, PRETTY PET SALON will run on most hardware, and it achieved substantial success, so it is possible to work around the limitations.

In terms of software fragmentation, this varies hugely country by country. We publish our findings on our blog, and you can see the radical differences between high-end and low-end markets there (http://www.animoca.com/en/2013/02/animoca-data-high-end-hong-kong-and-low-end-india-jan-feb-2013/).

We look at it as an opportunity—managing compatibility became a competitive differentiator for us. Really, the fact that the market is fragmented into various hardware and software just means that consumers have choice, which is a good thing. Anyone struggling with these fragmentation issues simply needs to figure out which devices and OS versions are most popular among the consumers/regions they want to target, and then tailor their apps to those devices and Android versions. It can require slightly more development time and QA resources to address the Android market, but not as much as you'd think if you look at some of the alarmist reports on the issue.

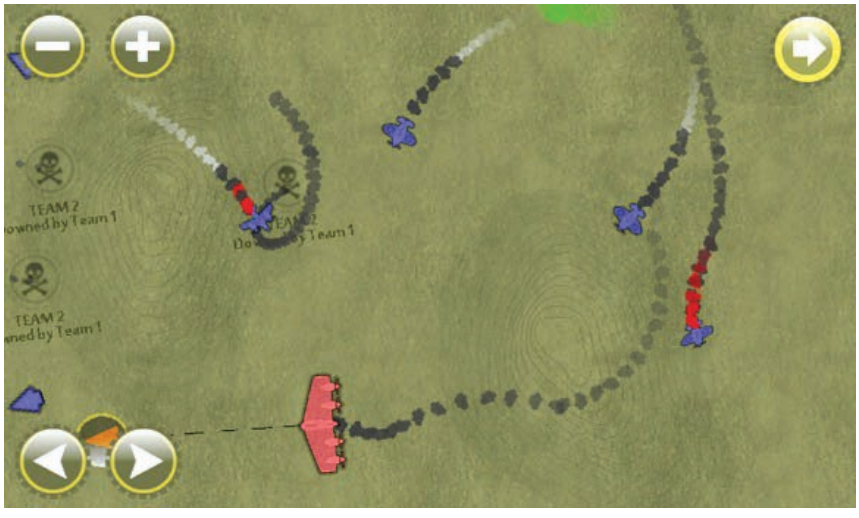
Do you target and test for specific devices?

David Kim: Yes. Although our general principle is to target and test for a multitude of devices in order to offer everyone an experience that is as good as possible, we still have to prioritize the devices most used by players of Animoca games, which vary by region and by country. In India, most users are on lower-end, more affordable phones, the top 10 phones for Animoca users tend to be relatively inexpensive Samsung devices, such as the Galaxy Y, Galaxy Fit, and Galaxy Ace, while the most common device in the higher-end market of Hong Kong is the pricier Galaxy Note 2, other high-end Galaxy products, and a couple of high-end Sony devices.

Do you have any tips for optimizing the Android dev process?

David Kim:

- Build a testing framework in your games where the game plays itself. This is a great aid for the QA team as it adjusts the speed and moves through the levels faster.
- Minimize external calls to external servers via effective use of caching as well as batching multiple calls together.
- Test against high-latency networks. Animoca has an internal tool that can traffic shape various network conditions.



- Catch your exceptions, including run-time exceptions.
- Use a crash-reporting tool within your codebase and your development workflow. Examples include Bugsense and Crittercism.
- When you have to do concurrent programming, don't roll your own from lower-level constructs. Try to use high-level facilities such as AsyncTask, ThreadPoolExecutor, etc.

How have your games sold on Android?

David Kim: Our games have performed well on Android, and many of them have appeared in the Top 100 or even the Top 10 charts. Some of our apps, like PRETTY PET SALON and STAR GIRL, have reached the Top 10 of the Top-Grossing Apps list, so we are pleased. In total, our apps have been downloaded more than 120 million times. We had high expectations for what Android could deliver because we are big believers in open systems, but when we entered the market Android was definitely playing second fiddle to Apple's iOS. I'd have to say that changed dramatically in 2012: Last year Android really came into its own.

Which app stores do you support? How do your Android sales compare to your sales on other platforms?

David Kim: Our Android games can be found on Google Play, Amazon's Appstore, the NOOK Store, Samsung Apps, and across many alternatives including carrier-specific app stores. Some stores do drive more installs than others, but in general we've found it beneficial to be in as many as possible.

Typically we have found that Android apps monetize slightly lower on a per-user basis than iOS, but this is a very complex matter. The Apple product spectrum is more expensive than Android, and most studies have shown that Apple users tend to be financially better off than Android users. That means that iOS users are more likely to have greater disposable income and as a result are more likely to spend more on their devices, compared with Android users. This is not surprising, because Android offers a remarkable number of "budget" hardware options catering to the less wealthy, and the platform is (after all) a few years behind iOS in terms of in-app purchasing and the associated culture. Android also tends to be much more popular in poorer emerging markets like India or China, where several barriers to payment exist (culture, comparatively lower purchasing power, lack of easy payment options, etc.)

But all that is changing very fast: The Android high end is becoming slicker and flashier with every new phone released, which drives up device cost and grants Android developers access to a wealthier user base. And the sheer number of Android users today and projected over the next couple of years will lead to correspondingly huge opportunities. The popular view that Android monetization is automatically inferior to iOS is rather myopic, especially in the medium- to long-term when you factor in the scale of the Android audience.

Overall, have you found Android dev to be worth the extra work? Are you looking into other mobile platforms?

David Kim: Yes, Android development is absolutely worth the slight amount of extra work it requires to manage fragmentation, and we do recommend it to other developers because right now this is definitely a forward-looking platform. Android is our primary focus right now, but our DNA is very much cross-platform and we wouldn't want to tie ourselves exclusively to one platform. Windows 8 is a beautiful system that already has some great hardware, and Blackberry is making an interesting case for the BlackBerry 10.

VECTOR UNIT

Ralf Knoesel (CTO), Matt Small (CEO)

Prior dev background (platforms): PC, N64, PS1, PS2, Xbox, Xbox360, PS3, Wii

Shipped Android titles: RIPTIDE GP, SHINE RUNNER, BEACH BUGGY BLITZ

Preferred toolset: Our own game engine/tools, plus Bullet Physics and FMOD

Is fragmentation still a major issue for you?

Ralf Knoesel: Fragmentation is a worry, yes, but not a major worry anymore. We design our games to support any aspect ratio from 4:3 to 16:9 and beyond; lack of Neon floating-point math is not a big deal anymore with today's clock speeds and CPU performance; OS version support is not such a big deal for us since we're now able to require 2.3 (Gingerbread) so that we can use NativeActivity.

The biggest issue for us is the tendency for the manufacturers to ship with subtle shader compiler bugs in their OS flavors. This can be a bit of a nightmare because the same device can have different bugs based on which OS version the consumer is running.

Do you have any tips for optimizing the Android dev process?

Ralf Knoesel: For dealing with texture compression, we use ETC1 for non-alpha textures, and DXT5 for textures containing alpha. For devices that don't support S3TC, we decompress at load time (starting with lower-level mips on RAM-limited devices). This significantly simplifies asset packaging.

Matt Small: On the content side, the biggest gotcha is UI. But designing the UI for

multiple resolutions and aspect ratios does not have to be a huge headache. The key thing is to make it flexible, with elements that float, stretch, or are locked to different edges of the screen. Also don't integrate the front-facing UI elements (like buttons) too tightly with the background, or make your backgrounds aspect-ratio-dependent.

We design for a 3:2 default aspect ratio, and make everything slide up/down or left/right from there. The backgrounds are made up of tiling/stretching graphic elements or 3D scenes. That way you can basically throw your UI against any aspect ratio and you're good to go.

Do you target and test for specific devices?

Ralf Knoesel: At this point we have a decent selection of popular devices representing the various GPU manufacturers, plus a few quirky troublesome devices that are popular enough to warrant support. Generally, we have coverage for the top five to 10 devices according to the developer console statistics. When a hot new device comes out that we know is going to be huge, we pick one up and add it to our roster.

Matt Small: If we get a lot of support emails on a device we don't have, we have to balance the popularity of the device with the severity of the problem and the time we think it'll take to fix it. If it's a serious problem on a popular device, we'll buy one and fix it.

How have your games sold on Android?

Ralf Knoesel: Our games have been doing well on Android, better than expected. Our most recent free-to-play game BEACH BUGGY BLITZ is about to hit 10 million downloads on Google Play.

Matt Small: All of our games have been profitable. Our first game, RIPTIDE GP, has over 250K installs, at an average MSRP of $1.99, and we made it for about $100K, so that's pretty good. It's not ANGRY BIRDS money, but it's more than enough to keep a small company going.

How do your sales compare to your experience with other platforms?

Ralf Knoesel: The main advantage of the Android platform for a small developer is the fact that the Google Play store ranking system is more difficult to manipulate by the big publishers with their user-acquisition spends. Rankings are based on more than simply number of installs in a period of time. This means that quality apps tend to creep to the top and have some decent inertia once they get there. Regarding the consumer, we find that Android consumers are less willing to pay for apps/in-app purchases and piracy is much more rampant. However, we're able to overcome this with volume.

Matt Small: Our experience has been the opposite of what you generally hear: We have done significantly better on Android than on iOS. We have noticed that in our free-to-play game-conversion rates are about two to three times better on iOS, but the volume on Android more than makes up for it. That said, oddly enough Android players seem less concerned about price on pay-to-play games. From time to time we'll put Riptide GP on sale from $1.99 to $0.99—on iOS we see a big (temporary) jump in sales that more than makes up for the 50% discount. But on Android, it barely moves the needle.

Which app stores do you support?

Ralf Knoesel: On Android, we have used Google Play, Amazon, and B&N NOOK. Google Play has by far been the most successful. We have tried some other app stores in the past (carrier-specific, prepaid cards, etc.), but sales on those have been insignificant. We are also partnering with Incross to sell in Korea (SKT T-Store, etc.).

Matt Small: Our experience with carrier- and hardware-specific app stores has been pretty abysmal. We get royalty payments of like $2.50 for a month's worth of sales. Amazon and NOOK have been pretty good for us, but my feeling is that their audiences lean more toward casual and puzzle-type games, and the discovery mechanisms in those stores are not as well developed. Like Ralf said, the vast majority of our revenue comes from the Play Store.

Overall, have you found Android dev to be worth the extra work?

Matt Small: Android development is absolutely worth the work, but you have to plan to be cross-platform from the very start of development. It's a lot harder to take a game that was specifically made for iOS and then belatedly port the thing over.

Ralf Knoesel: Also, we are currently making our games available on BB10 where we are seeing moderate success (more than expected) due to the "early to market" effect.

HIDDEN VARIABLE STUDIOS

Charley Price (CCO)

Prior dev background (platforms): GBA, PC, Xbox 360, PS3, Flash, iOS, Android

Shipped Android titles: BAG IT!

Preferred toolset: Unity

Is fragmentation still a major issue for you? Which devices do you target?

Charley Price: It's always a worry, but in some ways the variety of devices is so broad, it pretty much demands that you define your min spec criteria early in the development process so you can make sense of it all. Given that we simultaneously develop for iOS as well, our min spec decision hardware-wise on that platform helped to implicitly inform the min spec choices we made for Android.

Knowing the risks of different screen sizes/aspect ratios, we've developed our titles with floating UI elements that anchor (or are positioned relative to) the edges of the play space, and made sure that slight variations in viewable space don't impact gameplay dramatically. As such, when new devices and aspect ratios were released, we've generally been able to support them right out of the gate.

As for devices, a combination of App Store data and Flurry analytics gives us a pretty good sense of what types of devices are being used to play our game. As such, we tend to focus much of our attention on our first-gen NOOK Color and Kindle Fire (among our most popular devices with narrow aspect ratios) and our Nexus One (as a min spec).

Do you have any tips for optimizing the Android dev process?

Charley Price: Most of our Android-specific notes end up being directly related to our experiences developing in Unity. For example, many of the risks of creating Android-specific plug-ins can be mitigated using inexpensive, off-the-shelf solutions from sites like Prime31.com.

Going into more detail, one of our engineers, Clark Kromenaker, wrote up some thoughts on his blog (http://supersegfault.com/unity-on-android-save-data-pitfall/) about some of the pitfalls inherent in storing data on Android devices (for example, local memory vs. SD card memory), which would likely be a good read for new Android devs.

How have your games sold on Android?

Charley Price: We first released BAG IT! back in late 2011 on both iOS and Android simultaneously. Since then, we have released it on almost a dozen different Android app stores, both worldwide and foreign territory-specific (including Samsung, Mobiroo, and Fuhu to name a few less-common examples), but the only ones that have really generated a meaningful financial return for us thus far have been Google, Amazon, and B&N NOOK.

Overall, our iOS versions of BAG IT! have outperformed our Android SKUs in terms of total downloads 7:1 (although if you remove one big iOS promo event it's closer to 3:1). However, if you just count paid downloads, iOS and Android as a whole are neck and neck (with iOS slightly ahead). That said, IAPs still favor iOS, with almost a 3:1 advantage over Android.

These numbers are generally pretty surprising to fellow mobile devs, many of whom have opted to eschew Android as a platform for fear of lack of sales. In particular, both Google Play and B&N NOOK have been pretty equal in terms of sales, both outperforming Amazon by a reasonable margin. One of the reasons we suspect we've been so successful on the B&N NOOK store is the fact that it currently only supports premium games (with free "trial" versions available for full titles). As such, as a premium title, you're able to avoid the glut of freemium games on other platforms.

Funny thing is, we originally didn't have any intention of launching the game on the B&N NOOK. As it happens, we were showing the game to some folks at an Android Developer's Conference, and one of the B&N NOOK reps suggested we check out their device. We loaded up the APK then and there, and it worked like a charm. A few hours of setup later and we were pretty much ready to roll on their store—which ended up being one of our most consistent revenue streams.

As another example, we were approached about releasing BAG IT! on the Nabi Fuhu (a kid-centric Android tablet), which required some custom content and support for the hardware. Given the uncertain return on investment, we were skeptical of investing the time and energy required. We were able to arrange a device preload deal with our Lite SKU. If nothing else, this helped to build awareness of our game on the platform, and gave us some experience with preloads that we hope to leverage moving forward.

Overall, have you found Android dev to be worth the extra work?

Charley Price: Absolutely. For us, releasing on Android has been a critical aspect of our success. After some mild success on iOS at launch, we (like many others) soon experienced a pretty precipitous decline (with oscillations thereafter). Meanwhile, our sales on Google Play exploded out of the gate (courtesy of a timely feature), and—even after the feature ended—each of our Android SKUs remained strong, consistent performers. Beyond the revenue, we also continued to accumulate heaps of great customer reviews. This gave us confidence that the issues we were experiencing on iOS were primarily discoverability issues, as opposed to concerns about the overall quality of the product. So we continued to actively support the game, and pursued other promotional opportunities—eventually leading to a temporary free promotion (supported by Appoday) that netted us over three million new downloads. Were it not for our performance on Android, we may have shifted our focus to other projects earlier and missed out on our iOS opportunity.

Are you looking into other mobile platforms?

Charley Price: Currently, the time required to get the game ported over relative to the known ROI makes it a risky proposition. Instead, we have spent those resources focusing on our second and third projects. However, alternative platforms are definitely still on our radar! We are constantly looking at trends; for example, we're eager to see how the OUYA performs, and already have tentative plans to bring BAG IT! (and future titles) to that platform as well.

SHINY SHOE

Mark Cooke (cofounder)
Prior dev background (platforms): Xbox, Xbox 360, PS2, PS3, GameCube, iOS, Android, Sifteo Cubes
Shipped Android titles: OFFWORLD
Preferred toolset: Unity

Is fragmentation still a major issue for you?

Mark Cooke: Thanks to the great work done by the engineers at Unity, it has been easy for us to get our games running on a wide variety of Android hardware. As a tiny studio, Shiny Shoe has access to a limited number of devices to test on, but every device we've been able to get our hands on runs our games.

What is frustrating and creates a concrete support problem is that there are so many devices out there that there is literally no way we can support them all. We receive a few support requests a week where a particular part of the game is nonfunctional on a specific device/OS combination. I want to help these customers, but we literally can't; we don't have the time or money to get the hardware we need to find and fix these bugs.

When we shipped OFFWORLD we decided to launch on both Google Play and the Amazon App Store, another form of dealing with Android fragmentation. This necessitated additional work in terms of IAP processing and other store-specific changes. This work wasn't free in terms of engineering and QA time, and in our case it wasn't worth it; we have somewhere on the order of 100 times the number of downloads on Google in comparison to Amazon. Overall, fragmentation hasn't been a huge issue for us.

Do you have any tips for optimizing the Android dev process?

Mark Cooke: Make sure to design your UI to support multiple resolutions and aspect ratios from the start. We did this on OFFWORLD and it made it easy to launch on a ton of different devices with minimal UI work. We took a fairly simple route to achieve this that can work for many games; basically, we resize an orthographic camera dynamically based on the vertical screen resolution and the resolution that our UI was authored at. In combination with that we used a Unity plug-in called Multiplatform toolkit, created by fellow indie Owlchemy Labs out in Boston, which allowed us to reposition UI widget containers based on aspect ratio.

One thing I would recommend against if you can avoid it is targeting multiple texture-compression formats via different APKs with manifest files that limit target hardware by the <supports-gl-texture> property. As I understand it, there is only one texture-compression format universally supported across all Android devices: ETC1. That format doesn't support an alpha channel. OFFWORLD uses a large number of textures and almost all of them have an alpha channel, meaning if we wanted to ship a single APK we'd have to switch over to uncompressed 16- or 32-bit textures. This was scary—we were really concerned about either using too much runtime memory or harming visual quality by switching to uncompressed 16-bit color.

So with that in mind we tried to support the three major hardware-supported compressed texture formats—PVRTC, ATC, and DXT—via separate APKs that included texture data for each platform. This is a nightmare and if you don't have a good build process set up you're in for pain. We eventually launched with this setup but quickly ran into a huge issue—some models of the Samsung Galaxy S2 (a very popular Android device at the time we launched) didn't have hardware that supported any of the aforementioned compression formats. This means that the Google Play store thought our game didn't work on many Galaxy S2s and thus people with that device couldn't download it.

After going through all of that we decided to scrap our setup and switch

to uncompressed textures. That took additional work to reduce texture sizes where we could get away with it, but ultimately it has worked out much better for us and allowed all models of the Galaxy S2 to download and play our game.

How have your games sold on Android?

Mark Cooke: Contrary to conventional wisdom, OFFWORLD has done significantly better on Android than on iOS. We're not really sure why, though! We received minor features by both platform holders that helped drive early installs. After the initial bump we dropped off in a major way on iOS but have continued to find an audience on Google Play. Our revenue on Google Play is currently three times higher than our revenue on iOS. Also, I know this is out of the ordinary, but for us ARPU is fairly similar between the platforms. Amazon has been a total bust for us unfortunately.

Overall, have you found Android dev to be worth the extra work? Are you looking into other mobile platforms?

Mark Cooke: I always consider this question on a case-by-case basis, but for OFFWORLD, because we are using a cross-platform development tool like Unity, I think it was worth the extra work. Unity has done the majority of the engineering work for us, easing development significantly. I'm interested in potentially releasing on Windows Phone to test that market, but not until Unity supports it.

THE GAME BAKERS

Emeric Thoa (CEO)

Prior dev background (platforms): iOS, Android, Xbox 360, PS3, Wii

Shipped Android titles: SQUIDS, SQUIDS WILD WEST

Preferred toolset: Cocos2D, Unity

Is fragmentation still a major issue for you?

Emeric Thoa: It's not a worry as such, it's just additional work we need to plan for. We know we will have to deal with several versions: an HD and SD version to match the different capabilities of the devices, and a freemium and a premium version to match the different stores' requirements. We are still learning what's worth doing and what's not, but so far we want to believe that reaching the widest audience possible is the way to go.

How have your games sold on Android?

Emeric Thoa: We released SQUIDS on Android as a freemium as part of our sponsoring deal with Tapjoy, and made probably around $700! That looks like a disaster, obviously, but I wouldn't blame Android for that. SQUIDS was not designed as a freemium game at all and it's only natural that it didn't succeed as a free-to-play game. The sequel, SQUIDS WILD WEST, is releasing March 8 and will be a premium app this time.

Which app stores do you support? How do your Android sales compare to your sales on other platforms?

Emeric Thoa: SQUIDS was a success on iOS, but as I said, it released as a premium, so it's hard to compare. But surprisingly enough, the IAPs are also better on iOS. The conversion rate is higher, and so is the ARPPU. My bet is that Apple users are more used to paying for digital content, but I believe that this difference is going to fade. We'll be able to compare better with our next game, COMBO CREW, as it'll be a simultaneous launch on iOS and Android.

ACTION BUTTON ENTERTAINMENT

Brent Porter (cofounder)

Prior dev background (platforms): iOS, Android

Shipped Android titles: TNNS

Preferred toolset: Whatever best fits the job

Is fragmentation still a major issue for you?

Brent Porter: I guess it only becomes a problem when you already have something built and suddenly realize, "Oh gosh, I have to get this to work on 100 other devices now." If you do some planning from the beginning there shouldn't be too much reason to worry.

Do you have any tips for optimizing the Android dev process?

Brent Porter: Again, understanding from the beginning that a lot of elements will have to be flexible helps. I think this is why you see a lot of games with resolution-independent art styles. Some people get grossed out by something that "looks like a Flash game," but when you are dealing with different screen sizes and resolutions it suddenly makes a lot of sense to go with something flexible.

How do your Android sales compare to your sales on other platforms?

Brent Porter: There exists a perception that you aren't going to see Android consumers make purchases like you might elsewhere. I guess it has to come from somewhere! Maybe Android users are more likely to invest in a certain type of game. This might explain why you hear about vastly different numbers from different developers and their experience with Android vs. iOS.

Overall, have you found Android dev to be worth the extra work?

Brent Porter: Of course Android dev is worth it if you think you will be getting some attention. And everyone wants their game to get attention so I guess the answer is always sort of yes? It's difficult to see beforehand if a particular project is going to *definitely* require and make money with an Android version. I mean, everyone wants their game to sell like JETPACK JOYRIDE, and Halfbrick Studios definitely benefits from getting their game in front of more people.

Maybe there's an even more important question. Do comments like "God, when will we get an Android version!!??" and "I cared until I realized it was only for iOS, lol" put stress and fear into your heart? If so, it's time to start planning your future in Android development!

APP DEVELOPERS CONFERENCE

CO-LOCATED WITH GDC 13 NEXT

APP DEVELOPERS CONFERENCE 2013

LOS ANGELES, CA
NOVEMBER 5–7, 2013
EXPO DATES: NOVEMBER 5–6

ADCONF.COM

UBM Tech

BY ANDERS DRACHEN, ALESSANDRO CANOSSA, AND MAGY SEIF EL-NASR

INTRO TO **USER ANALYTICS**

HOW TO TURN YOUR PLAYERS' SESSIONS INTO VALUABLE DESIGN FEEDBACK

The science of game analytics has gained a tremendous amount of attention in recent years. Introducing analytics into the game development cycle was driven by a need for better knowledge about the players, which benefits many divisions of a game company, including business, design, etc. Game analytics is, therefore, becoming an increasingly important area of business intelligence for the industry. Quantitative data obtained via telemetry, market reports, QA systems, benchmark tests, and numerous other sources all feed into business intelligence management, informing decision-making.

Two of the most important questions when integrating analytics into the development process are what to track, and how to analyze the data. The process of choosing what to collect is called feature selection. Feature selection is a challenge, perhaps especially when it comes to user behavior.

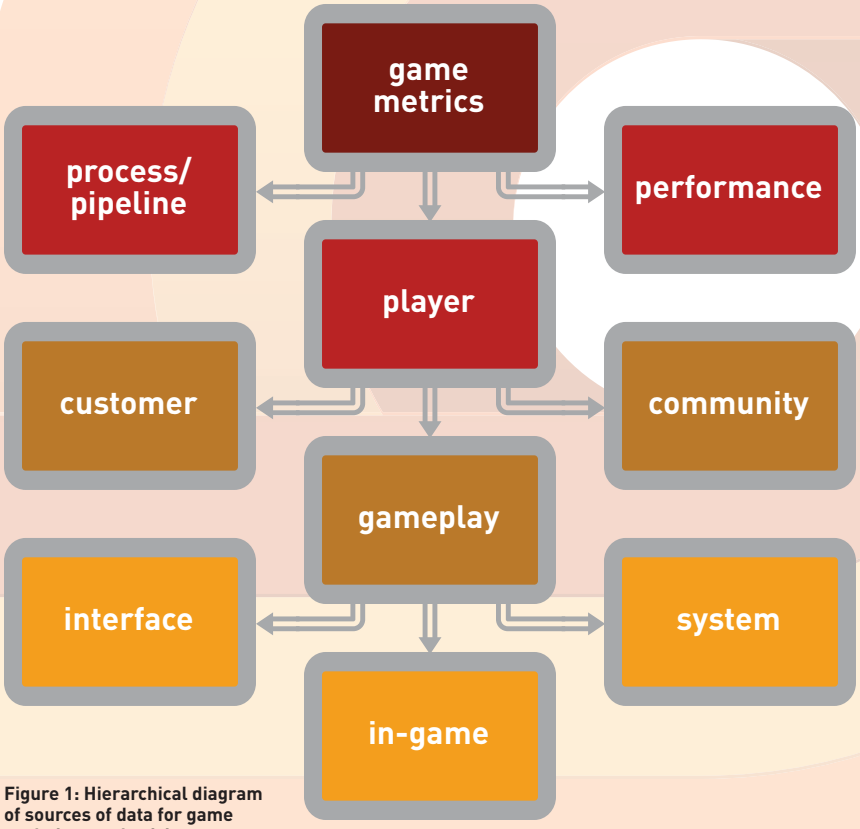


Figure 1: Hierarchical diagram of sources of data for game analytics emphasizing user metrics.

There is no single right answer or standard model we can apply to decide what behaviors to track; there are instead several strategies that vary in goals: e.g., improve the user experience or increase monetization. In this article, we will attempt to outline some of the fundamental concerns in user-oriented game analytics, with feature selection as an overall theme. First, we'll walk through the types of trackable user data, and then introduce the feature selection process, where you select how and what to measure. Importantly, this article is not focused on F2P and online games—analytics is useful for all games.

DATA FOR ANALYTICS

The three main sources of data for game analytics are:

Performance data: These are related to the performance of the technical- and software-based infrastructure behind a game, notably relevant for online or persistent games. Common performance metrics include the frame rate at which a game executes on a client hardware platform, or in the case of a game server, its stability.

Process data: These are related to the actual process of developing games. Game development is to a smaller or greater degree a creative process, but still requires monitoring, e.g., via task-size estimation and the use of burndown charts.

User data: By far the most common source of data, these are derived from the users who play our games. We view users either as customers (sources of revenue) or *players*, who behave in a particular way when interacting with games. The first perspective is used when calculating metrics related to revenue—average revenue per user (ARPU), daily active users (DAU)—or when performing analyses related to revenue (churn analysis, customer support performance analysis, or microtransaction analysis). The second perspective is used for investigating how people interact with the actual game system and the components of it and with other players, by focusing on in-game behavior (average playtime, damage dealt per session, and so forth). This is the type of data we will focus on here. These three categories do not cover general **business data**, e.g., company value, company revenue, etc. We do not consider such data the specific domain of game analytics, but rather as falling within the general domain of business analytics.

DEVELOPING METRICS FROM USER DATA

Many people have proposed different methods of classifying user data over the past few years. From a top-down perspective, a development-oriented classification system is useful, as it serves to funnel user metrics in the direction of three different classes of stakeholders—for example, as follows (see **Figure 1**):

Customer metrics: *Covers all aspects of the user as a customer—for example, cost of customer acquisition and retention. These types of metrics are notably interesting to professionals working with marketing and management of games and game development.*

Community metrics: *Covers the movements of the user community at all levels of resolution, such as forum activity. These types of metrics are useful to community managers.*

Gameplay metrics: *Any variable related to the actual behavior of the user as a player inside the game (object interaction, object trade, and navigation in the environment, for example). Gameplay metrics are the most important for evaluating game design and user experience, but are furthest from the traditional perspective of the revenue chain in game development, and hence are generally underprioritized. These metrics are useful to professionals working with design, user research, quality assurance, or any other position where the actual behavior of the users is of interest.*

Customer metrics: As a **customer**, users can download and install a game, purchase any number of virtual items from in-game or out-of-game stores and shops, spending real or virtual currency, over shorter or longer timespans. At the same time, customers interact with customer service, submitting bug reports, requests for help, complaints, and so on. Users can also interact with forums, official or not, or other social-interaction platforms, from which information about these users, their play behavior, and their satisfaction with the game can be mined and analyzed. We can also collect information on customers' countries, IP addresses, and sometimes even age, gender, and email addresses. Combining this kind of demographic information with behavioral data can provide powerful insights into a game's customer base.

Community metrics: Users interact with each other if they have the opportunity. This interaction can be related to gameplay (combat or collaboration through game mechanics) or social (in-game chat). Player-player interaction can occur

in-game or out-of-game, or some combination thereof—for example, sending messages bragging about a new piece of equipment using a post-to-Facebook function. In-game, users can interact with each other via chat functions, out-of-game via live conversation (TeamSpeak or Skype), or via game forums.

These kinds of interactions between players form an important source of information, applicable in an array of contexts. For example, a social-network analysis of the user community in a F2P game can reveal players with strong social networks—who are the players likely to help retain a big number of other players in the game by creating a good social environment (think guild leaders in MMORPGs). Likewise, mining chat logs and forum posts can provide information about problems in a game's design. For example, data-mining datasets derived from chat logs in an online game can reveal bugs or other problems. Monitoring and analyzing player-player interaction is important in all situations where there are multiple players, but *especially* in games that attempt to create and support a persistent player community, and which have adopted an online business model, which includes many social online games and F2P games. These examples are just the tip of a very deep iceberg, and the collection, analysis, and reporting on game metrics derived from player-player interaction is a topic that could easily take up several volumes.

Gameplay metrics: This subcategory of the user metrics is perhaps the most widely logged and utilized type of game telemetry currently in use. Gameplay metrics are measures of player behavior: navigation, item and ability use, jumping, trading, running, and whatever else players actually do inside the virtual environment of a game (whether 2D or 3D). Four types of information can be logged whenever a player does something or something happens to a player in a game: *What is happening? Where is it happening? At what time is it happening?* And: *Who is involved?*

Gameplay metrics are particularly useful for informing game design. They provide the opportunity to address key questions, including whether any game world areas are over- or underused, if players utilize game features as intended, and whether there are any barriers hindering player progression. These kind of game metrics can be recorded during all phases of game development, as well as following launch.

Players can generate thousands of behavioral measures over the course of a single game session—every time a player inputs something to the game system, it has to react and respond. Accurate measures of player activity can include dozens of actions being measured per second. Consider, for example, players in a typical fantasy

WORLD OF WARCRAFT.

MMORPG like WORLD OF WARCRAFT: Measuring user behavior could involve logging the position of the player's character, its current health, mana, stamina, the time of any buffs affecting it, the active action (running, swinging an axe), the mode (in combat, trading, traveling), the attitude of any NPC enemies toward the player, the player character name, race, level, equipment, currency, and so on—all these bits of information simply flow from the installed game client to the collection servers.

From a practical perspective, you may want to further subdivide gameplay metrics into the following three categories (in order to make your metrics more searchable, for instance):

In-game: *Covers all in-game actions and behaviors of players, including navigation, economic behavior, as well as interaction with game assets such as objects and entities. This category will in most cases form the bulk of collected user telemetry.*

Interface: *Includes all interactions the player performs with the game interface and menus. This includes setting game variables, such as mouse sensitivity and monitor brightness.*

System: *System metrics cover the actions game engines and their subsystems (AI system, automated events, MOB/NPC actions, and so on) initiate to respond to player actions. For example, a MOB attacking a player character if it moves within aggro range, or progressing the player to the next level upon satisfaction of a predefined set of conditions.*

To sum up, the array of potential measures from the users of a game (or game service) can be staggering, and generally we should aim for logging and analyzing the most essential information. This selection process imposes a bias, but is often necessary to avoid data overload and to ensure a functional workflow in analytics.

INTEGRATING ANALYTICS

Bias is introduced in the dataset both by the selection of the features to be monitored and also by the measuring strategies adopted, and that happens to a large degree when analysts work in a vacuum. If those responsible for analytics cannot communicate with all relevant stakeholders, critical information will invariably end up missing and the full value of analytics will not be realized.

Analytics groups are placed differently across companies due to analytics arriving to the industry from different directions, notably user research, marketing, and monetization, and this can lead to a situation where the analytics team only services or prioritizes their parent department. Having a strong lateral integration—making sure that the analytics team communicates with all the teams, for example—helps to avoid this issue. This also helps alleviate the common problem that the analytics teams, without having sufficient access to design teams, are forced to self-select features to track and analyze, without having the proper grounding in the design of the game and its monetization model.

Even for a small developer with a part-

time analyst this can be a problem. Another typical problem is that the decision about which behaviors to track is made without involving the analytics team. This can lead to a lot of extra time spent later on trying to work with data that are not exactly what is needed, or needing to record additional datasets. Good communication between teams also helps alleviate friction between analytics and design.

Importantly, analytics should be integrated from the onset of a production—all the way back in the early design phases. Early on, devs should plan out which kinds of behavior they want to track, and how frequently they want to track it. This allows for optimal planning of how to ensure value from analytics to design, monetization, marketing, etc. Analytics should never be slapped on sometime after the beta. In this way, analytics is similar to other tools like user research, in that it ideally is embedded throughout the development processes, and after launch.

FEATURE SELECTION

Knowing that there is an array of things we can measure about user behavior, how do we then select among them? And do we really have to make choices here? Sadly, yes. In real life, we rarely have the resources to track and analyze all possible user behaviors, which means we have to develop an approach to analytics that considers cost-benefit relationships between the resources required for tracking, storing, and analyzing user telemetry/metrics on one hand, and the value of the insights obtained on the other. It is also important to be aware that the analyses needed during different stages of production and post-launch varies. For example, during the latter phases of development, tuning design is vital, but many metrics related to monetization cannot be calculated because purchases have not been made by the target audience yet.

We will discuss this in more detail below, but in short, by following this line of reasoning, the minimum set of user attributes that should be tracked, stored, and analyzed should include considerations as to the following (see **Figure 2**):

1) General attributes: The attributes that are shared for users (as customers and players) across all games. These form the core metrics that can always be collected, for any computer game—for example, the time at which a user starts or stops playing, a user ID, user IP, entry point, and so on. These form the core of any game analytics dataset.

2) Core mechanics/design attributes: The essential attributes related to the core of the gameplay and mechanics of the game. (For example, attributes related to time spent playing, virtual currency spent, number of opponents killed, and so on.) Defining the core design attributes should be based directly on the key gameplay mechanics of the game, and should provide information that lets designers make inferences about the user experience (whether players are progressing as planned, if flow is sustained, death ratios, level completions, point scores).

3) Core business attributes: The essential attributes related to the core of the business model of the company, for example, logging every time a user purchases a virtual item (and what that item is), establishes a friend connection in-game, or recommends the game to a Facebook friend—or any other attributes related to revenue, retention, virality, and churn. For a mobile game, geolocation data can be very interesting to assist target marketing. In a traditional retail situation, none of these are of interest, of course.

4) Stakeholder requirements: In addition, there can be an assortment of stakeholder requirements that need to be considered. For example, management or marketing may place a high value on knowing the number of Daily Active Users (DAU). Such requirements may or may not align with the categories mentioned above.

5) QA and user research: Finally, if there is any interest in using telemetry data for user research/user testing and quality assurance (recording crashes and crash causes, hardware configuration of client systems, and notable game settings), it may be necessary to augment to attributes on the list of features accordingly.

When building the initial attribute set and planning the metrics that can be derived from them, you need to make sure that the selection process is as well-informed as possible, and includes all the involved stakeholders. This minimizes the need to go back to the code and embed additional hooks at a later time—which is a waste that can be eliminated with careful planning. That being said, as the game evolves during production as well as following launch (whether a persistent game or through DLCs/patches), it will typically be necessary to some degree to embed new hooks in the code in order to track new attributes and thus sustain an evolving analytics practice. Sampling is another key consideration. It may not be necessary to track every time someone fires a gun, but only 1% of these. Sampling is a big issue in its own right, and we will therefore not delve further on this subject here, apart from noting that sampling can be an efficient way to cut resource requirements for game analytics.


BATTLEFIELD 4.

PRESELECTING FEATURES

One important factor to consider during the feature selection process is the extent to which your attribute set selection can be driven by pre-planning, by defining the game metrics and analysis results (and thereby the actionable insights) we wish to obtain from user telemetry and select attributes accordingly.

Reducing complexity is necessary, but as you restrict the scope of the data-gathering process, you run the risk of missing important patterns in user behavior that cannot be detected using the preselected attributes. This problem is exacerbated in situations where the game metrics and analyses are also

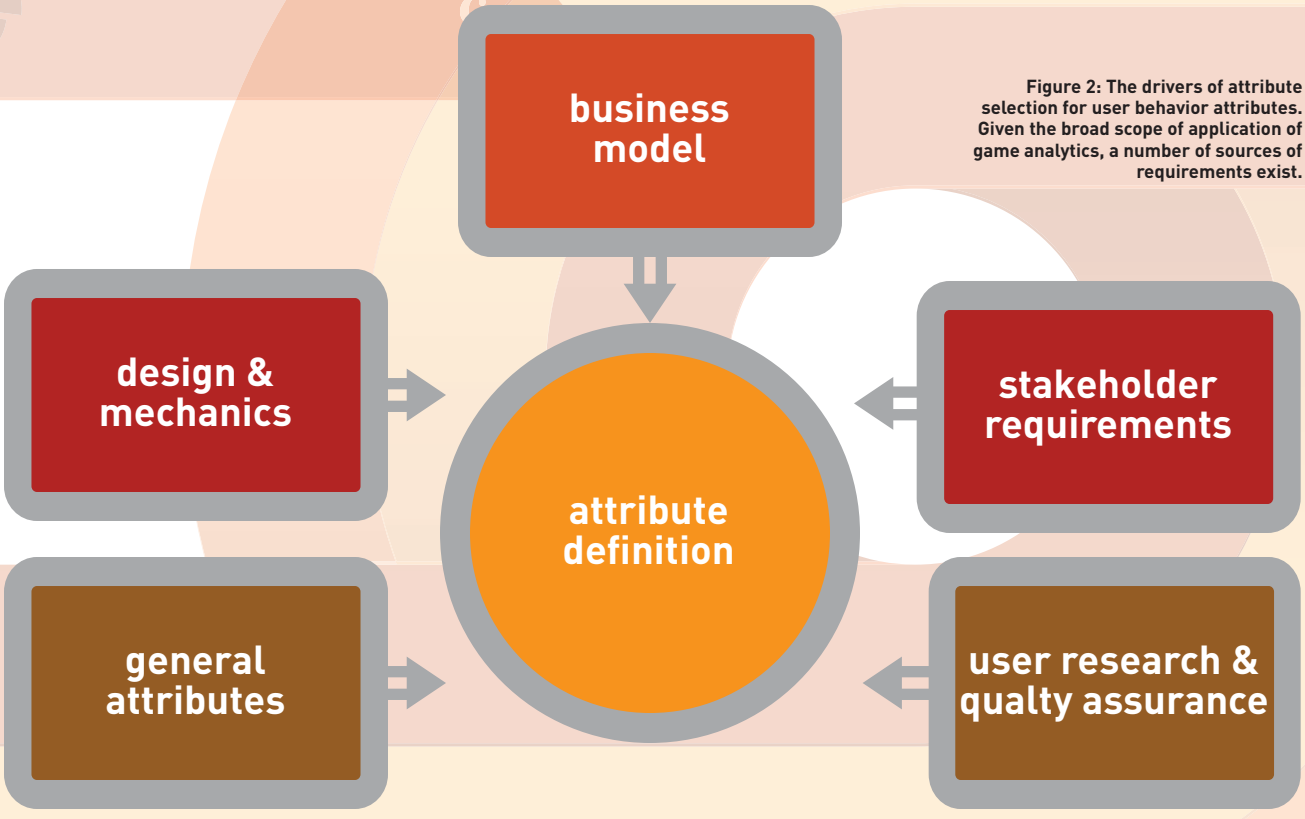


Figure 2: The drivers of attribute selection for user behavior attributes. Given the broad scope of application of game analytics, a number of sources of requirements exist.

predefined—for example, relying on a set of Key Performance Indicators (such as DAU, MAU, ARPU, LTV, etc.) can eliminate your chance of finding any patterns in the behavioral data not detectable via the predefined metrics and analyses. In general, striking a balance between the two situations is the best solution, depending on available analytics resource. For example, focusing exclusively on KPIs will not tell you about in-game behavior, e.g., why 35% of the players drop out on level 8—for that we need to look at metrics related to design and performance.

It is worth noting that when it comes to user analytics, we are working with human behavior, which is notoriously unpredictable. This means that predicting user analytics requirements can be challenging. This emphasizes the need for the use of both explorative (we look at the user data to see what patterns they contain) and hypothesis-driven methods (we know what we want to measure and know the possible results, not just which one is correct).

STRATEGIES DRIVEN BY DESIGNERS' KNOWLEDGE

During gameplay, a user creates a continual loop of actions and responses that keep the game state changing. This means that at any given moment, there can be many features of user behavior that change value. A first step toward isolating which features to employ during the analytical process could be a comprehensive and detailed list of all possible interactions between the game and its players. Designers are extremely knowledgeable about all possible interactions between the game and players; it's beneficial to harness that knowledge and involve designers from the beginning by asking them to compile such lists. Secondly, considering the sheer number of variables involved even in the simplest game, it is necessary to reduce the complexity through a knowledge-driven factor reduction: Designers can easily identify isomorphic interactions. These are groups of similar interactions, behaviors, and state changes that are essentially similar even if formally slightly different. For example "restoring 5 HP with a bandage" or "healing 50 HP with a potion" are formally different but essentially similar behaviors. The isomorphic interactions are then grouped into larger domains. Lastly, it's required to identify measures that capture all isomorphic interactions belonging to each domain. For example, for the domain "healing," it's not necessary to track the number of potions and bandages used, but just record every state change to the variable "health."

These domains have not been derived through objective factor reduction; there is a clear interpretive bias any time humans are asked to group elements in categories, even if designers have exhaustive expert knowledge. These larger domains can potentially contain all the possible behaviors that players can express in a game and at the same time help select which game variables should be monitored, and how.

STRATEGIES DRIVEN BY MACHINE LEARNING

Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed. More than an alternative to designer-driven strategies, automated feature selection is a complementary approach to reducing the complexity of the hundreds of state changes generated by player-game interactions. Traditionally, automated approaches are applied to existing datasets, relational databases, or data warehouses, meaning that the process of analyzing game systems, defining variables, and establishing measures for such variables, falls outside of the scope of automated strategies; humans already have defined which variables to track and how. Therefore, automated approaches individuate only the most relevant and the most discriminating features out of all the variables monitored.

Automated feature selection relies on algorithms to search the attribute space and drop features that are highly correlated to others; algorithms can range from simple to complex. Methods include approaches such as clustering,

classification, prediction, and sequence mining. These can be applied to find the most relevant features, since the presence of features that are not relevant for the definition of types affects the similarity measure, degrading the quality of the clusters found by the algorithm.

DIMINISHING RETURNS

In a situation with infinite resources, it is possible to track, store, and analyze every user-initiated action—all the server-side system information, every fraction of a move of an avatar, every purchase, every chat message, every button press, even every keystroke. Doing so will likely cause bandwidth issues, and will require substantial resources to add the message hooks into the game code, but in theory, this brute-force approach to game analytics is possible.

However, it leads to very large datasets, which in turn leads to huge resource requirements in order to transform and analyze them. For example, tracking weapon type, weapon modifications, range, damage, target, kills, player and target positions, bullet trajectory, and so on, will enable a very in-depth analysis of weapon use in an FPS. However, the key metrics to evaluate weapon balancing could just be range, damage done, and the frequency of use of each weapon. Adding a number of additional variables/features may not add any new relevant insights, or may even add noise or confusion to the analysis. Similarly, it may not be necessary to log behavioral telemetry from all players of a game, but only a percentage (this is of course not the case when it comes to sales records, because you will need to track all revenue).

In general, if selected correctly, the first variables/features that are tracked, collected, and analyzed will provide a lot of insight into user behavior. As more and more detailed aspects of user behavior are tracked, costs of storage, processing, and analysis increase, but the rate of added value from the information contained in the telemetry data diminishes.

What this means is that there is a cost-benefit relationship in game telemetry, which basically describes a simplified *theory of diminishing returns:* Increasing the amount of one source of data in an analysis process will yield a lower per-unit return.

A classic example in economic literature is adding fertilizer to a field. In an unbalanced system (underfertilized), adding fertilizer will increase the crop size, but after a certain point this increase diminishes, stops, and may even reduce the crop size. Adding fertilizer to an already-balanced system does not increase crop size, or may reduce it.

Fundamentally, game analytics follow a similar principle. An analysis can be optimized up to a specific point given a particular set of input features/variables, before additional (new) features are necessary. Additionally, increasing the amount of data into an analysis process may reduce the return, or in extreme cases lead to a situation of negative return due to noise and confusion added by the additional data. There can of course be exceptions—for example, the cause of a problematic behavioral pattern, which decreases retention in a social online game, can rest in a single small design flaw, which can be hard to identify if the specific behavioral variables related to the flaw are not tracked.

GOALS OF USER-ORIENTED ANALYTICS

User-oriented game analytics typically have a variety of purposes, but we can broadly divide them into the following:

***Strategic analytics**, which target the global view on how a game should evolve based on analysis of user behavior and the business model.*

***Tactical analytics**, which aim to inform game design at the short-term, for example an A/B test of a new game feature.*

***Operational analytics**, which target analysis and evaluation of the immediate, current situation in the game. For example, informing what changes you should make to a persistent game to match user behavior in real-time*

To an extent, operational and tactical analytics inform technical and infrastructure issues, whereas strategic analytics focuses on merging user telemetry data with other user data and/or market research.

When you're plotting a strategy for approaching your user telemetry, the first factors you should concern yourself with are the existence of these three types of user-oriented game analytics, the kinds of input data they require, and what you need to do to ensure that all three are performed, and the resulting data reported to the relevant stakeholder.

The second factor to consider is to clarify how to satisfy both the needs of the company and the needs of the users. The fundamental goal of game design is to create games that provide a good user experience. However, the fundamental goal of running a game development company is to make money (at least from the perspective of the investors). Ensuring that the analytics process generates output supporting decision-making toward both of these goals is vital. Essentially, the underlying drivers for game analytics are twofold: 1) ensuring a quality user experience, in order to acquire and retain customers; 2) ensuring that the monetization cycle generates revenue—irrespective of the business model in question. User-oriented game analytics should inform both design and monetization at the same time. This approach is exemplified by companies that have been successful in the F2P marketplace who use analysis methods like A/B testing to evaluate whether a specific design change increases both user experience (retention is sometimes used as a proxy) and monetization.

SUMMING UP

Up to this point, the discussion about feature selection has been at a somewhat abstract level, attempting to generate categories guiding selection, ensuring comprehensiveness in coverage rather than generating lists of concrete metrics (shots fired/minute per weapon, kill/death ratio, jump success ratio). This because it is nigh-on impossible to develop generic guidelines for metrics across all types of games and usage situations. This is not just because games do not fall within neat design classes (games share a vast design space and do not cluster at specific areas of it), but also because the rate of innovation in design is high, which would rapidly render recommendations invalid. Therefore, the best advice we can give on user analytics is to develop models from the top down, so you can ensure comprehensive coverage in data collection, and from the core out, starting from the main mechanics driving the user experience (for helping designers) and monetization (for helping making sure designers get paid). Additional detail can be added as resources permit. Finally, try to keep your decisions and process fluent and adaptable; it's necessary in an industry as competitive and exciting as ours.

***

Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa are the editors of **Game Analytics - Maximizing the Value of Player Data**, *a recently published compendium of insights from more than 50 experts in industry and research. This article is based on selected content from the book. Dr. Anders Drachen, lead game analyst at Game Analytics, is a veteran game analyst and game researcher. He curates the GA development blog (blogs.gameanalytics.com). Dr. Alessandro Canossa is an associate professor at the PLAIT lab at Northeastern University, working with the interplay between in-game behavior and psychology of personality and motivation. Dr. Magy Seif El-Nasr is an associate professor at the PLAIT lab and directs Game Educational Programs and Research at Northeastern. Finally, Janus Rau Moeller Soerensen (Crystal Dynamics) contributed to some of the chapters this article draws material from.*

INTERNAL INDIES

WHAT CAN A LARGE, ESTABLISHED STUDIO LEARN FROM SMALL, INDEPENDENT STUDIOS?

BY STEVE STOPPS

Someone far more clever than me once said, "Creative endeavors are abandoned, not finished." Anyone who has worked on a large game knows that game development can often be filled with both conflict and compromise. I've worked on my share of commissioned games, and I know that these projects are tough. Big games are expensive to make, and that raises the already-high stakes for everyone involved—which, in turn, makes everyone involved far more risk-averse. Were the stakes substantially lower, most devs would want to take on more creatively ambitious projects, with no boundaries or restrictions getting in the way of making what we want to make.

In other words, we'd want to be indies—but without the terrifying risk of failure. So, six months ago, I asked this question: Are there lessons from indie development that can be utilized in a larger, established studio setting? In order to answer this question, we set out a simple studio experiment. We changed our contracts of employment, invited five of the most senior developers to form a new team, and gave that team complete creative freedom.

CHANGING THE CONTRACT

The first step was a relatively small but significant change to our contracts, so that people could develop their own projects in their own time and retain the IP themselves—that is, own their own game designs. Our aim was to free our staff to be more creative and for both them and the company to potentially profit from that. The reason many companies seek to control their developers' IP is an understandable fear that some of them will go on to make great games and leave the studio to pursue their own projects full time—and in fact, this has already happened to some degree to us.

Allow me to let you in on a little secret: If you work for a reasonably large-sized studio, there will be people who are working on cool stuff on their own time whether their contracts allow it or not, because people who make games are inherently creative. They can't stop themselves. We just formalized this and turned it into a positive.

We found that this risk is worth it; when developers feel like they own their work, they feel more creative and more motivated to develop new skills. For example, one of the programmers on our team learned new shader techniques on his own time,

while another learned a lot about Facebook integration through developing their own game. Our current game directly benefited from the new skills and knowledge gained through outside-of-work creativity, as will future games developed by the team.

BUILDING A NEW TEAM By chance, a small number of our most senior developers had finished their company projects and had around eight weeks before their next projects began. As in many studios, people between projects are often asked to undertake research into various areas, and this group had been tasked with learning more about the mobile market. They decided to do this in a more practical, hands-on sense than the company had initially envisioned—that is, they wanted to make a game to prove out the issues about which they were learning. Because they were some of our most senior devs and we trusted their abilities, we gave them free rein to make a game in that eight weeks. They repaid this trust with a completed game, KUMO LUMO, which has since been published by Chillingo and received over one million downloads just 10 days after launch.

THE GOALS FOR KUMO LUMO This team was very commercially minded during the whole process; they not only wanted to make a game that had its own voice and personality, but also create something that would help us understand the mobile market much better. They also wanted to learn as many practical lessons about the process as possible.

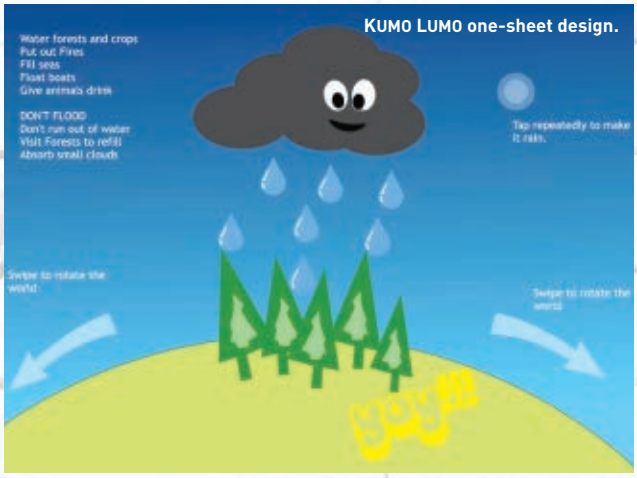
Their research showed that the majority of money spent on iOS comes from men ages 25-34 (70% more than any other group). Having looked at the games that were selling, they quickly realized that this did not necessarily mean the game had to be traditionally hardcore. Also, the rise of freemium games over the past year was clearly a key area our studio needed to learn more about, so they decided that monetization should be part of the game design from the outset.

Since the team had limited time and resources, they knew that the game had to be simple in design and execution to secure the best chance of success. Their focus on this simplicity ruled out many early game concepts. For example, they had a great puzzle-game idea based on the paper toy movement, but quickly realized they had neither the time nor expertise required for this concept. To appeal to App Store shoppers bombarded by options, the visual direction of the game was as important as the gameplay. Put simply: Screenshots sell your game, the gameplay keeps people playing.

Finally, the team knew from the studio's previous experience that a game lives or dies by PR and marketing. Once you've made a good game, people need to know that it exists. These days, that means finding ways of engaging directly with the audience. Consequently, they decided to do their own social marketing alongside development, so that they could learn about this process, too.

DESIGNING KUMO LUMO The design process was simple. The design lead presented several concepts, all as one-sheet visual designs. They were assessed on suitability for the audience, possibility for monetization, scope for visual appeal, and feasibility of delivery. In the end, the team chose the design shown below.

Once the team had decided on a design, the art team explored many potential visual treatments before choosing one that was both appropriate for the audience, felt suited to mobile, but also had an individual identity. For inspiration, they looked to the sticker bomb/street art movement. This process was part research and part instinct. The team started by identifying a large addressable market on iOS, then researched what this audience liked and didn't like. In parallel they looked at the art direction of many of the best-selling mobile games. Finally, they struck a careful balance between what they thought the audience would like, what seemed to sell on mobile, and something unique that would stand out on the App Store.



KUMO LUMO one-sheet design.

Some of the early concepts skewed too young, or were too niche, or looked too much like something that had been done before. It took lots of iteration before the team settled on a direction that felt right.

The result was KUMO LUMO. By the end of the first week, the team had its trademarks and copyright checked, URLs secured, T-shirts and mugs ordered. Eight weeks later the team had a fully formed game ready for its first submission to Apple. In addition, during the same period, they had created and maintained a blog at kumolumo.com, and engaged with various forms of social media to drive traffic and create interest.

DEVS ON MARKETING
Of course, part of being an independent developer isn't just making what you want to make—it's being able to sell it, too. Indie devs are responsible for their own marketing and PR, so we handed that responsibility off to the team as well. Here's what they learned:

Marketing is harder than it looks. Developers frequently make disparaging remarks about the "dark arts" of marketing, but having been there and tried it ourselves, it is really more difficult than we initially thought.

Twitter is massively useful—if you make it personal. Remember that social media really is all about human interaction. Using bots to add users to your Twitter account isn't useful; it increases your followers but does not necessarily help your engagement with people. Click-throughs count, and bots won't help you with that. The team members who carefully curated their followers and built personal relationships with people got much better engagement. This certainly takes time, but is well worth it. But what do we mean when we say it was worth it? I don't believe we generated many direct sales from our work on Twitter, but it did enable us to create a real buzz around the game, which was picked up by many key players. This sense of buzz added a lot of credibility to our early press outreach.

Analytics are important. By using tools like Google Analytics and link-tracking services such as bit.ly, the team could understand which activity was generating the most hits to our blog. Once you find that out, just do more of it. For example, some forums picked up on our "undercover development" angle. There was some real strength of feeling, and this in turn generated traffic and interest in our game. So we helped stoke this controversy a little.

Facebook is massive. No, really. It sounds obvious, but the team was so excited by Twitter and Pinterest that they didn't do anything to promote the Facebook page. However, Facebook still generated more traffic than pretty much all the other activity put together, and was still our second-biggest driver of traffic

Journalists need stories. They have space to fill with interesting and engaging content. Therefore, if you make your story interesting, you are more likely to get their attention. Think about their readership, and what they might find interesting. For us, we created our story around our "undercover development." At the point of starting the project, it was true: Our bosses knew we were doing something iOS related, but had no idea what. So we played on this to create a more interesting story. A large studio moving into iOS development isn't unique or interesting—a rogue element within a studio developing their own game has much more of an edge.

Keep up the momentum. Our team activity had to stop for three months when we moved into contract negotiation with a publisher, which ultimately undid much of their hard work in building an audience. I am sure there are people out there who believed for a while that KUMO LUMO was vaporware.

KUMO LUMO first concept.

LUCK, SKILL, AND TIMING

After about eight weeks, the game was made and the marketing was in progress. The team unveiled the game to the executives, who had not been previously involved. The team presented everything they had learned, which was warmly received.

This is when some luck, good timing, and the team's excellent decisions paid off: The game was finished just before one of our regular American business trips. Our business development executives set up a meeting with Chillingo, and they agreed that we would show them KUMO LUMO. Chillingo really liked it! They understood the visual direction and the style of game and felt it had great commercial promise. Of course, they had some feedback on how to make the game better, and this is what the team has been working on for the past few months.

We have often been asked why we used a publisher when we could have self-published. For us the answer was simple. We felt that we had met a partner who was as passionate about the game as we were and who wanted to share with us their vast knowledge and experience; it just seemed too good an opportunity to pass up. It's been a real joy working with them.

MAKING A NOTE HERE: HUGE SUCCESS We would consider the results of our indie experiment a success; we learned that giving people autonomy, creative freedom, and trust is motivating, and motivating and trusting experienced and knowledgeable developers delivers great results. Also, we learned that social media marketing is worthwhile, but only if you do it the right way—no shortcuts!

And probably more than anything, trust your vision. Make the game you believe in, and make it fun! If you take nothing more from indie games, take this: An original idea and a purity of vision will make your game feel fresh and alive. IN

***

Steve Stopps is project director of Team Lumo for Blitz Games Studios. [Team Lumo is hard at work on our next project, PAPER TITANS, and we've been applying all the lessons we learned from KUMO LUMO.]

Sticker bomb branding sheet.

BALDUR'S GATE ENHANCED EDITION

POSTMORTEM

BALDUR'S GATE: ENHANCED EDITION launched on PC on November 28 and on iPad on December 8 to great fan interest, rocketing to the #2 spot on iTunes in the U.S. and #1 on many other app stores around the world. Our initial plan was to launch all platforms as close to each other as possible. In retrospect, this was very naive given the size of our team and the volume of work required. We're working steadily now to build versions for the remaining platforms, and to roll out concurrent versions so cross-platform multiplayer can be a reality. We've had great success on the PC and iPad, with great sales and positive feedback from the fans. We're very anxious to roll out versions for the remaining platforms as fast as we can. From the trenches, the development of BALDUR'S GATE: ENHANCED EDITION was an interesting journey. We had early moments of exuberance as we played the first tablet version and proved our theory that BALDUR'S GATE did indeed kick ass on tablets. We had great moments of insight when we brought in people from the modding community and shared work-in-progress versions with them. We had moments of despair, such as lost source art and subsequent sacrifices to salvage the publishing deal. Along the way we had the opportunity to work with some great talents such as Mark Meer, John Gallagher, and Sam Hulick. BALDUR'S GATE: ENHANCED EDITION was a challenge, but we've all come out with some sanity remaining and a great understanding of the game, the engine technology, and what makes Baldur's Gate the legend it is.

WHAT WENT RIGHT

THE IPAD AND THE TOUCH INTERFACE We proved you could produce an epic, massive RPG in the tablet space and that the market was starving for such a project. The success has been awesome, and the feedback from the fine fellows at respected media outlets like Touch Arcade and Pocket Tactics has been a real positive. Beamdog cofounder and veteran developer Cameron Tofer went crazy on the tablet interface early on—we had an iPad version running and semifunctional almost a year before we shipped.

Every new build brought radical changes, as we explored the logical means to perform actions in BALDUR'S GATE using a touch design paradigm. In the end, Cameron's approach broke down to favoring common actions and giving them the least-complex control mechanism. For example, the most common player action is to scroll the screen, so we used the photo app UI standard of a touch-drag for scrolling. The second most common action is moving/interacting, which we mapped to a single touch. We initially favored group selection, but after a number of clumsy iterations we realized the vast majority of actions in BG are performed with either a single character selected or the entire group, and as such, group selection could be moved to a modal button.

We've added to this level of polish recently with a further enhancement we call smart radius. Smart radius allows the user to have the precision of a mouse with the less-accurate nature of a touch interface, making it easier to enter buildings, pick up items from the ground, and so on. We'll continue to iterate as we go, making the game better as we move along. From an overall UI design, we wanted to keep the original feel, but base it around the new resolution of 1024x768. My priority in UI was to make the user portraits as large as possible. We listed all the required UI elements and slowly reconstructed all the art and panels around the new portrait sizes, all while trying to embrace some UI guidelines from Apple for the iPad, such as keeping buttons over 44 pixels in height (which we only failed to do in a few spots, like class selection). The overall process was challenging, as we tried to balance rewriting the entire UI code while keeping the existing complex (and overly integrated—damn you, Scott Greig) systems working.

BETA TESTING We beta tested the game for over six months, during which we received tremendously helpful feedback on how to improve it. We found and fixed bugs at a rapid pace, in part because experienced BALDUR'S GATE modders in the beta were able to suggest fixes that cut our development time. In our mind, the beta was a core element to the success of the project. The feedback on our new content was excellent, as we found bizarre edge cases that caused the quests to break in unexpected ways. Due to the complexity of BALDUR'S GATE, we needed a lot of testing to find all the possible ways a quest could be broken and act quickly to fix the problem and retest. By sticking with the original game structures and asset formats, we were able to leverage the knowledge of a community of developers and not just the direct efforts of our team. As I mentioned, this beta-testing group was volunteers recruited from our forums. We chose people with strong skills and a good background with the original game, so the feedback was fast and on target. The beta team had access to our bug database and beta bugs went straight into the system.

We also hired an external testing company named iBeta to help us final the iPad versions and to ensure testing coverage. They were able to run us through a "precertification"-style test to catch the differences between the different iPad hardware devices.

OUR OWN DIGITAL DISTRIBUTION SERVICE Beamdog has two parts: an online digital distribution service, and our game development team, which we call Overhaul Games. We developed the Beamdog Store with the goal of selling our games directly to our users, no middlemen, no confusion on support, just a developer and a customer. The direct-sales option and

a preorder program allowed us to receive payment many months in advance of a typical royalty-advance deal, which was a great benefit for a small, self-funded developer.

The other great benefit was how our digital-distribution service allowed us to push out beta builds of the game six months in advance and update testers to new versions without a hitch. We were able to push PC fixes to the game in record time, doing four major updates in under two weeks as we found issues and corrected them. Sure, we had a few hiccups with the client software and out-of-date SSL root certificates on user systems, but the system performed exceptionally well. We were able to add servers in Germany, Singapore, and on the East and West coasts of the U.S. without a hitch as we approached launch, and we handled a huge surge in demand as we pushed hundreds of terabytes of data to our users. We use Beamdog daily internally for all our build distribution, and without it we couldn't have been nearly as agile in our response, effective at version control, or as fast at build distribution. We even use our service to send around Mac OSX, iPad, and Android development builds of the game. We've been very happy with our service to date and we'll be expanding on what it can do going forward.

WHAT WENT WRONG

INTRICATE CODE-DATA DEPENDENCY I have a development joke I often drop: "When a programmer believes they're being clever is when they create the greatest atrocities." BioWare's Infinity Engine is chock-full of clever. For example: To render a character to the screen, the correct frame and orientation of a character sprite is first loaded out of a resource file using a very heavy resource-management system. The sprite is then color-mapped using a 256-color palette swap to enable player colors. Following the remapping, any number of further palette-manipulation code (stoneskin, anyone?) can step in and further change the actual data. Then the sprite is rendered against whatever potentially covering elements are nearby. The final result is sent to the screen in 64x64 pixel tiles to be rendered.

The entire system runs under a dynamic update system that flags 64x64 tiles as updated and renders them or, with no change, leaves the tile from the previous buffer. The volume of clever shifts and tweaks along the way make it nearly impossible to track down all the ways in which a simple sprite can be manipulated. In some cases, the data can reference many different .2da data files on how it can be manipulated, from equipment-changing animation frames to a data redirection to render a dwarf sprite instead of the default human. Complexity is par for the course in an RPG of this magnitude, but the intricate linking of assets and code really limited our ability to make the architectural improvements we wanted to make.

THE LOST SOURCE ART Originally, we had planned to do a BALDUR'S GATE: HD. Our plan was simple: Grab the original artwork, clean it up, and rerender it at higher resolutions and with better materials, which would give us stunning versions of the areas everyone remembers. We planned to take the character models and rerender them with many more frames of animation and add new orientations to the movement to make the game smoother. We nailed down the core terms, got everyone on the same page, and we got our first drop of the assets from BioWare. A few days later we noticed a large hole where the source art should be—stuff like 3DS Max files and texture images. "No problem," I said; I contacted Derek French over at BioWare, and he dug further and sent us more data. We again dug through and failed to find the source art, so I made arrangements to visit BioWare with a removable drive and work with Derek and the IS department to find the assets.

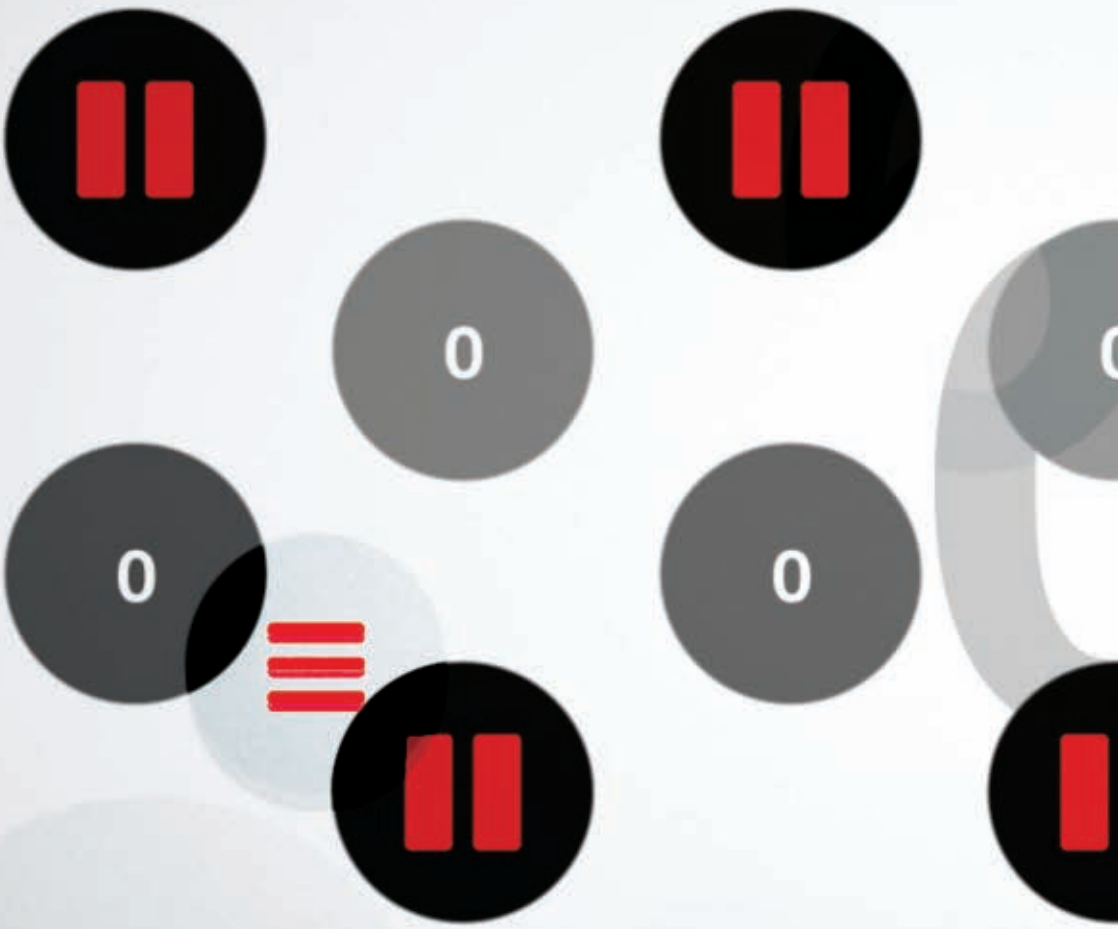
After two days of searching we came to the horrible realization that the source artwork was stored on a departmental drive and not a project drive, and as such was not frequently backed up. We dug through tape backups to no avail. The source art was lost. At this point we brought the information back to Atari and the deal was dead in the water. Cameron and I spent a few weeks rethinking the concept and we repitched Atari with an "Enhanced Edition" as opposed to an "HD" version. We discussed the implications of a non-HD version and we had to renegotiate royalty terms to get the deal back on track. We basically had to completely discard our plans and start anew after almost a year of negotiation. The end result is a little less graphical flash for launch than we had initially planned, but we actually had a bit more time to make more widespread improvements as a result, so the project is different; in some ways larger, but still pretty awesome. The key learning here is: Don't panic!

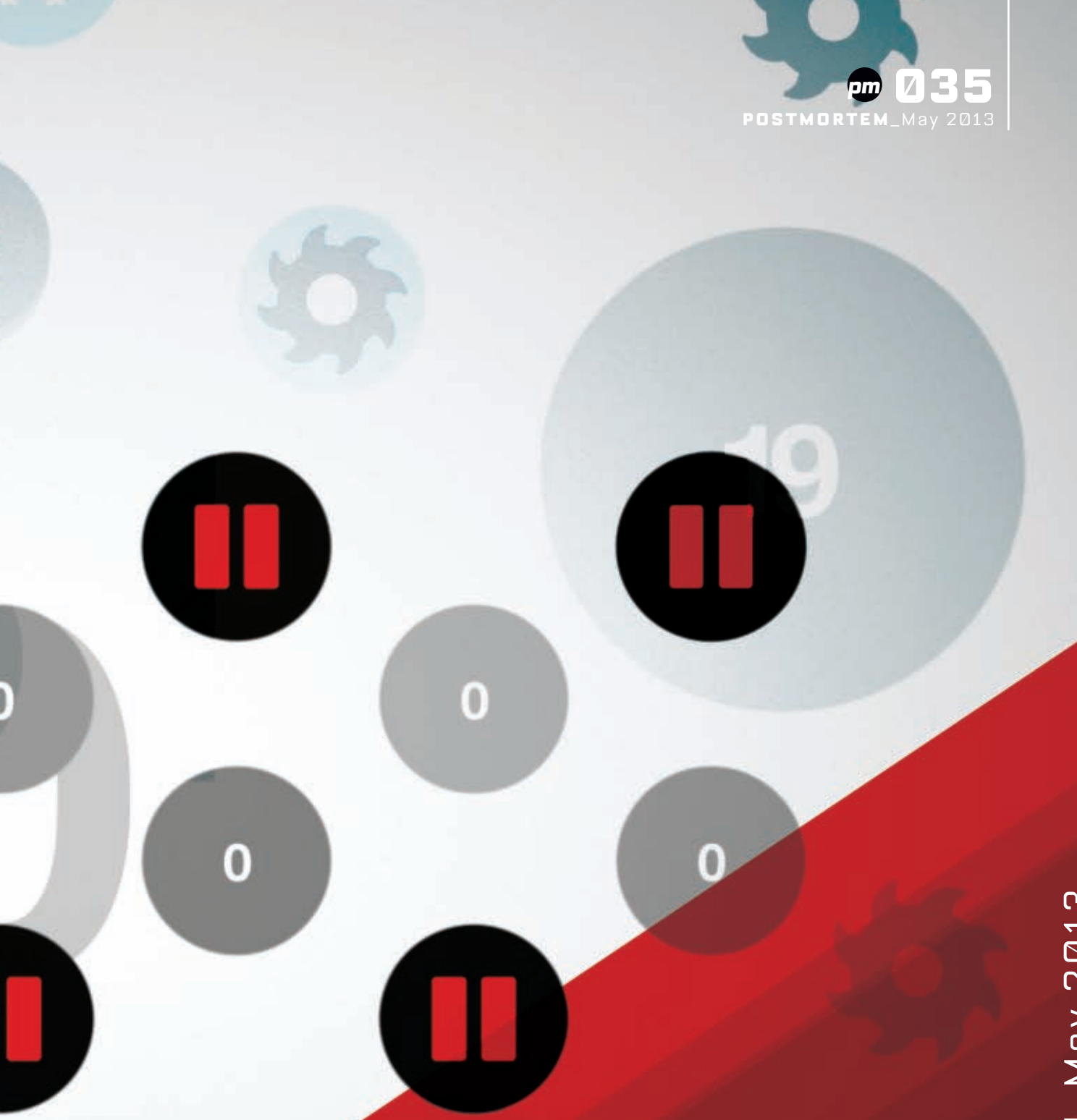
iOS6 Late in development, we started testing on the upcoming iOS6 beta build. We're not 100% sure what was changed in the core iOS graphics code, but we took a massive hit in framerate on all iPads. The iPad 1 and iPad 3 became completely unplayable, less than a month from our contractually amended ship date. We quickly diverted most of our key programming effort to fixing the performance concerns. We fought a war between hurting the visual upscaling quality and improving the framerate. In the end, we lost a lot of key developer time to a completely unplanned task. The end result was that a great deal of our bug-fixing time was spent not fixing the bugs we wanted, and the quality of the shipping build suffered. The lesson learned here is to build in some slack in terms of performance for when you need it due to an unplanned change, and to budget some extra time for periodic performance testing on current and beta OS releases.

In summary, BG:EE was defined by the product it is built upon, and the hard, respectful work of a small and caring team. We could have done faster development work, and we could have had fewer bugs by making sweeping changes, but early on we decided to adopt the role of curator. We didn't want to change BALDUR'S GATE just to show off our development skills and leave our mark; we wanted to make a great game even better and in the process, bring it to new platforms. With the great commercial and critical success so far, we're very anxious to continue to improve the game even more and bring our newly leveled-up skills to bear on BALDUR'S GATE 2: ENHANCED EDITION.

***

Trent Oster is the president of Beamdog and Overhaul Games.

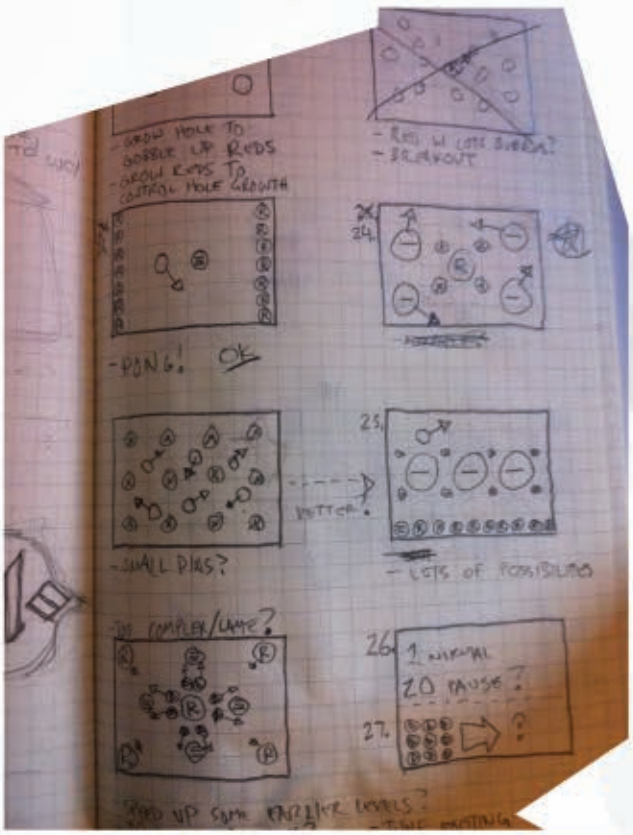

0
0
0

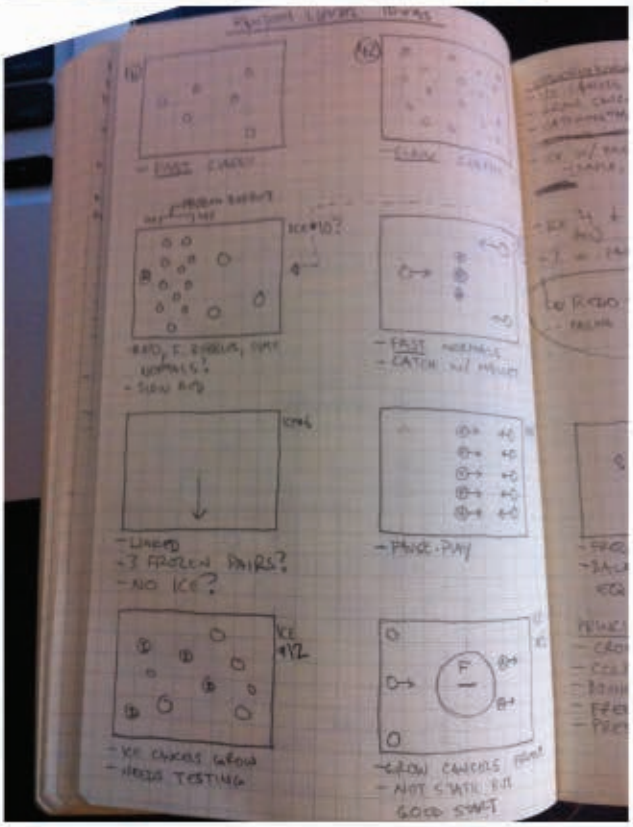


Hundreds

BY ADAM SALTSMAN

FOR THOSE OF YOU WHO AREN'T FAMILIAR WITH IT, HUNDREDS IS A SUCCESSFUL ARCADE-PUZZLE GAME DESIGNED PRIMARILY FOR THE IPAD. ERIC JOHNSON (MY PARTNER HERE AT SEMI SECRET) AND I WERE INSPIRED BY ONE OF GREG WOHLWEND'S FLASH GAMES, SO WE STARTED COLLABORATING WITH HIM ON A LARGER AND MUCH-IMPROVED VERSION BACK IN AUGUST OR SEPTEMBER OF 2011, AND WE FINALLY SHIPPED IT ON THE ITUNES APP STORE IN JANUARY 2013. IN HUNDREDS, PLAYERS TOUCH MOVING, BOUNCING CIRCLES. TOUCH A CIRCLE, AND IT WILL PHYSICALLY EXPAND AND CONTRIBUTE POINTS TO THE LEVEL TOTAL. REACH 100 POINTS, AND ADVANCE TO THE NEXT PUZZLE. THE CATCH: WHILE INFLATING, CIRCLES TURN RED. WHEN A RED CIRCLE TOUCHES ANOTHER CIRCLE, IT'S GAME OVER, MAN. IT MAY SOUND LIKE A SIMPLE MECHANIC, BUT OVER THE COURSE OF 100 LEVELS WE CONTINUOUSLY INTRODUCE NEW TOUCH-BASED MECHANICS AND OBSTACLES.





We opted for a modern, autodidactic approach to our puzzle design, allowing us to introduce new elements without requiring or relying on wordy tutorials. Players learn by experimenting and discovering things on their own, through what are essentially series of controlled experiments. We were very much inspired by games like BRAID, PORTAL, and SPACECHEM, though the barrier to entry for Hundreds is much lower in some ways than these more "hardcore" games, thanks to its simplified interface.

Our team of four—a designer/artist, a programmer, a level designer, and a sound designer—took more than 15 months to create HUNDREDS. I am very, very proud of the work we did; making inclusive and humanist games is my game design passion, and I am completely satisfied with our efforts. We hadn't shipped a profitable iOS game since CANABALT back in 2009, so building HUNDREDS was a huge experiment and a learning experience. Hopefully, this postmortem will let me clarify some of what we learned—both for my own sake, but also to lend some assistance or assurance to others in a similar situation.

Finally, I'm normally a bit opposed to the traditional what-went-right/what-went-wrong approach to postmortems, but I promise I have done my best to include real, unfiltered pros and cons here!

WHAT WENT RIGHT

1/ HARDWARE-NATIVE DESIGN A few years ago, I was interviewing my friend and game designer Miah Slaczka (LOCK'S QUEST, SCRIBBLENAUTS) for the now-defunct TIGRadio podcast, and we were talking about his design approach to the Nintendo DS specifically. He stressed the importance he personally placed on doing hardware-native design. That is, designing games really specifically based around the hardware interface and leveraging it, rather than simply compromising for it. A game like RIDICULOUS FISHING or BIT PILOT has really solid hardware-native controls and design, for example. A classic example of extremely non-native design, especially for touchscreens, is the virtual D-pad found in many ports and action games. PixelJunk's DSi games for Nintendo (especially TRAJECTILE) are great examples of hardware-native design too.

Anyway, I was interviewing Miah just a few months after releasing CANABALT, which was sort of miraculously interfaceless. Anything with a touchscreen or a button can run CANABALT just fine. For me, at least, that was a design fluke; I tried designing more games with that much flexibility in their controls, and I was really struggling. Talking with Miah about hardware-native designs really changed my ideas about commercial game design, and I began thinking about designing games very specifically for touchscreens, rather than for multiple platforms.

I designed a few different prototypes, but we didn't end up building any of them. Getting into that headspace ended up being a huge win for HUNDREDS, though. With CANABALT, I'd been mulling over one-button action game designs for a few years, and I think that paid off in a big way. By the time the opportunity to work on HUNDREDS came along, I'd already been thinking about touchscreen-centric game design for a few years, and was eager to put that time to good use. HUNDREDS, to my mind anyway, is almost the perfect touchscreen game; it is very immediate and 1:1 in its interactions. There is no interface. We use multitouch in an intuitive way. We avoided contrived and complicated gestures, and focused on getting the basics just right.

2/ MINIMALIST DESIGN The thing that stands out the most about HUNDREDS is the way it looks. Greg Wohlwend's striking minimalist graphic design permeates the entire game, with nearly everything important communicated by subtly different shades of gray, punctuated by bold, red user-interface elements.

The minimalist design of HUNDREDS had two big benefits. The first (and maybe most important) benefit is what I (somewhat hyperbolically) refer to as the "#Sworcery Doctrine." In this age of app stores and the proliferation of indie studios and indie games, it's not enough to just make a pretty good game and release it.

You could probably argue that was never enough! Right now it feels like creating something that stands out is an integral part of making indie games commercially. There aren't very many games out there that look like HUNDREDS, and that made a very big difference for us.

There are many different ways to make something that stands out, of course. We knew from the start of this project that one of the things we wanted to build was a game that wasn't embarrassing to play on your iPad in public. (That sounds snobby, and please keep in mind that I love cartoony games, and I enjoy violent games, and this is not a low-culture/high-culture thing.)

There is a stigma about games being "for kids," and I do think a lot of that has to do with their presentation, whether these games are aping Dreamworks cartoons, comic books, or R-rated action movies (which we all know are really marketed to kids). How hard is it to convince people that *How to Train Your Dragon* is totally amazing? We really wanted to make a game that didn't unnecessarily exclude anyone, and that definitely includes people who are turned off by stuff that is obviously "for kids." Games researcher Ian Bogost went so far as describing the HUNDREDS aesthetic as "haute couture" in an article for *The Atlantic*. The great thing is that kids still love Hundreds; it turns out they don't really need cartoon birds in order to enjoy something.

The more pragmatic benefit we got from pursuing a minimalist design is that we didn't have to spend months or even years drawing thousands of frames of animation, modeling backgrounds, or whatever. Nearly everything in HUNDREDS is procedurally generated, from the circles to the icons, and even the texture atlases for the fonts. We didn't even have an art pipeline; no retina-resolution assets required.

3/ LEGACY HARDWARE SUPPORT Another not-terribly-romantic aspect of HUNDREDS that I'm pretty proud of is our robust support for older iOS hardware. While there are some devices out there that HUNDREDS can't realistically support, we did push hard to get really good performance out of devices as old as the iPhone 3GS and especially the iPad 1. Even though the numbers indicate that these are not the most popular devices by a long shot, it was enormously satisfying to run at 60 frames per second even on these grouchy old gadgets. It also meant that our performance on newer devices was exceptional. Ironically, I'm not 100% sure I would recommend this to other developers, but more on that later.

4/ DISTRIBUTED DEVELOPMENT Like some other small indie studios, Semi Secret is a completely distributed development studio. We have no central office and we don't even meet face to face on a regular basis. Eric and I both live near Austin, but Greg was working from Baltimore, Chicago, and New York, and our incredible audio collaborator Scott Morgan worked from his home studio in Vancouver, CA. The huge pro here is that in spite of our lack of physical proximity, we got to work with exactly the right people. Scott (aka Loscil) was our first choice for audio, and HUNDREDS was all Greg's idea in the first place. Without long-distance collaboration, this project would not have been possible; it never would have existed.

We relied on a lot of common tools to accomplish this; Google Mail and Docs are basically invaluable. We rely exclusively on Git for source control, but for the first time we made heavy use of Dropbox to coordinate with Greg and Scott. Dropbox is great for folks who don't need Git's versioning and (for many new users) frustrating level of complexity.

5/ SLOW DEVELOPMENT HUNDREDS took us a pretty long time to make, by my standards anyway: 15 months at least; maybe a bit more. The longest I ever worked on the same game before this was a few months. CANABALT took five days. 15 months was a lot longer than I expected, and this had some serious ramifications for the project (more on this later). But it had a big positive effect on the project too. As level designer, I had about 90% of the game's mechanics and levels designed by March 2012, roughly nine months before we shipped the game. At the time I remember being frustrated that we were still working on menus and technology. But those "delays" ended up making the level designs much, much better.

On past projects, I was perfectly happy just sending game content out the door as long as it felt pretty fun to me. This isn't necessarily a horrible idea, but our goal was to build a game that would be able to teach itself to a very wide audience. We needed more than a few days to put the game in front of strangers and see if the game worked for them. I can't remember if it was Kellee Santiago or Robin Hunicke who originally told me about the idea of the Silver Rule. The Golden Rule, as we all know, says something like "Do unto others as you would like them to do to you." The Silver Rule says "Do unto others as they would do unto themselves." This tiny change makes a crucial and enormous difference in all manner of disciplines, but I really like the way it applies to game design and playtesting.

I want to reiterate that when I write "playtesting," this is not at all the same as "focus testing." When we playtest our games, we're not making sure players never get stuck. We're making sure that most players only get stuck on what we want them to get stuck on. Maybe other folks have different approaches, but for us, for our goal of building what we hope are intelligent games for a wide audience, we want people to get stuck sometimes. We want people to have little epiphanies and victories. What we don't want is for them to get stuck on a confusing menu item or grossly misunderstand a critical part of the game's basic controls or something. Over the course of the game's longer-than-expected development time, we were able to put it in front of a lot of different players in a lot of different settings, and we greatly benefitted from these playtesting experiences.

The other thing the long dev time allowed us to do was to mercilessly edit our designs. A lot of designers are familiar with the idea of "killing your darlings"—the necessity of editing to create a good, cohesive work. Sometimes there are elements of a design that we love but that don't fit in the design,

or don't make sense to our playtesters. For this project, it was very important to us that we strip out anything that wasn't functioning properly.

The hardest darling to kill in HUNDREDS was one of the very first ideas I contributed to the game: holes. This was a game about circles, and holes seemed like a really interesting obstacle to put in the mix. They would be circular, of course, but they could remove objects from the level, which was interesting, and had a kind of pinball vibe that I loved. In practice, though, holes were extremely problematic. They introduced some interesting dynamics, but they were shallow, and there was a huge downside: Puzzles could reach unwinnable states, and we were adamantly opposed to having a reset or retry button in the game. So, we killed the holes. Eventually.

The thing is, at least for me, it was (and is) really hard to muster the discipline required to really kill off some darlings if they're fresh out of the oven. "Who knows, right? Maybe people just aren't used to this new design yet. Maybe people just don't get it yet. They'll come around, they just need to get used to the way it—hang on. They're right! This idea sucks!"

This is how it goes in my head most of the time. I need some time, some editorial distance, before I can objectively judge an idea. There is some span of time, usually somewhere in between a week and a month, where an idea seems to cease to be my own, and starts to feel like someone else's idea, even if that someone else is just a past me that didn't know any better. And as soon as that idea isn't "mine" anymore, I find it very easy to judge its worth, especially when I'm trying to figure out why playtesters got stuck so long on a level that used that idea.

While developing HUNDREDS, we threw out a lot of stuff—at least half a dozen core mechanics or game objects, and dozens of puzzle designs that were fundamentally flawed in some way or other. We were throwing out level designs and replacing them with improved concepts even during the last month of development. This makes me even more certain that this long development period was crucial for achieving the necessary editorial distance.

WHAT WENT WRONG

1/ HARDWARE-NATIVE DESIGN (Oh ho, see what I did there? This is a thing I'm doing in this postmortem about double-edged swords! So now you're in on it too.)

As I said before, hardware-native design was one of the best things about the design of HUNDREDS. Unfortunately, from a business perspective, it's also been one of the most difficult aspects. If you look at the most successful independent games of the last few years, there is a bit of a pattern: Release game on some interesting native hardware or platform, then sell it on PC/Mac both directly and through Steam, then buy Teslas. The thing is, we are absolutely, completely, and intentionally tethered to touchscreens by the design of HUNDREDS. This is still a big market, which is great, but it's also a market where games have very, very low prices. Even at over 100,000 copies sold in the first month, we barely broke even, if that, on our year-plus development cycle. On Steam, well, we wouldn't have been buying Teslas, but we would have at least joked about buying Teslas.

2/ MINIMALIST DESIGN Minimalist game design is a passion of mine, but it can put enormous pressure on weird or unexpected parts of the design when you are trying to produce work that is competitive with less minimalist offerings. A lot of the game-buying audience has a sensitivity to polish. It's not a requirement, by any means, but it is part of a promise that we make as game designers, to our audience. To me, polish says, "Look at this game; look how much effort we put into it. Look how much we believed in

it! It's worth taking a second look at this game." It doesn't mean the game is good, of course, but it is one way of conveying a creator's confidence in the work. As an indie game studio with no marketing budget, there aren't a lot of ways for us to tell people this. So, we're pro-polish, as it were.

When your game is just circles, though, part of making the work really shine means drawing really, really good circles. I don't have room to go into detail here about how we did it, but I can tell you that drawing perfectly anti-aliased circles of any imaginable size, under a full-screen vignette effect, at 60 frames per second, on an iPad 1 or iPhone 4, is... difficult. The tech behind HUNDREDS was rebuilt from scratch at least three times during development. Which leads me to the next problem...

3/ LEGACY HARDWARE SUPPORT As almost any iOS developer will tell you, one of the great advantages of the platform is not having to deal with the intimidating degree of fragmentation of PC or Android. However, there is still a big difference between the low-end and high-end hardware that's out there. There are multiple screen resolutions, huge differences in CPU power, and large disparities in available memory to deal with. If we'd only targeted the most powerful iOS hardware, I think it we could have saved months of development time. For us, in retrospect, this was a much less trivial decision than I thought it was.

4/ DISTRIBUTED DEVELOPMENT Issues like technological delays were exacerbated by our physical distance from each other. Email can be a disastrous way of communicating about important issues, and as development time runs over the targeted deadlines, it is harder to stay focused and energized when you're half a country apart (or farther). Distance can hamper your ability to (I'm gonna say it!) synergize, too. There is undeniably something really powerful about being in a shared space with other creative people with a shared goal, where your voice, and body language, and outside interests create a kind of intoxicating creative soup that is a very fertile environment for lateral thinking and solving problems in counterintuitive ways.

But as someone more experienced and more intelligent than me once said, I'd much rather clarify confusing emails to the perfect person than have a crystal-clear face-to-face meeting with the wrong person. In an ideal world, you get to work with the perfect people in perfect physical proximity, but I don't know of any bootstrapped indie studios that can afford this luxury. We simply do the best we can with what we have!

5/ SLOW DEVELOPMENT This was brutal and almost destroyed the company. One of the reasons that we decided to work on HUNDREDS was because we thought we could get it to market in six months or less, which was only slighter longer than our financial runway at that moment. I thought we could do it in three months, so I doubled it and thought we had a great schedule. I see now that I should have quadrupled my estimate. Man, that was a bad estimate.

If CANABALT hadn't been included in a Humble Android Bundle back in early 2012, we would have had to face a very difficult choice: Release HUNDREDS in an unstable and unfinished state, or put it on hiatus until we could afford to finish it, by taking on other work individually, or aiming the studio at contract jobs.

Development actually ran long enough that we faced exactly that decision anyway, about six months later. I am very glad that we were able to persevere and release it, and obviously I am thankful, in hindsight, that we had this extra time, so that we could reflect on and edit the game and get it into its current state. Time to reflect has its own downsides: You end up with a lot of time to indulge in crippling self-doubt about your design. Near the end of the project, Greg and I panicked and started seriously considering reskinning the game to prominently feature cartoon blowfish characters. FishPop would save us all! We ended up staying the course, but given enough time to overanalyze my work, I am often prone to these sorts of concerns.

Why did it take so long to make a game about bouncing circles? Legacy hardware support was part of it, as was, I think, our distributed development arrangement. I think our skills were not spread optimally; we ended up having maybe five times more programming work than art or level design, which put a huge strain on Eric. We also had a lot of feature creep, though I still don't view that as a mistake in the case of HUNDREDS specifically. Our goal was to produce something that was competitive on the App Store, and we only added what we felt was necessary to create a full experience. We spent so long in development that our ideas about what a "full experience" was ended up changing along with the rising quality bar of other games and apps. It wasn't a DUKE NUKEM FOREVER arms race, but it is important to recognize this tendency of markets to be moving targets, especially at the start of a project. It's not enough to have a thorough understanding of what the market is like right now for our next game. What will the market be like when we finish our next game? That's the mark we have to aim for.

Looking back, I see our long development time as a catch-22 that I just didn't understand at the time. We couldn't afford not to spend that much time on HUNDREDS. I don't think people would have enjoyed it if we shipped it with fewer features or less polish. At the same time, we literally couldn't afford to spend that much time making it. We didn't have the runway, and we didn't seek outside investment in order to fix that. I think that was a big mistake.

LAUNCHING HUNDREDS

HUNDREDS launched January 2013 to a dozen or more perfect scores from reviewers all over the world. The design appeals to fans of all ages, genders, and ethnicities, and we sold over 100,000 copies in our first month on the App Store, with a modest (read: nonexistent) marketing budget.

That's not to say we didn't do any marketing; we started talking to Apple about HUNDREDS over a year before it launched. We were profiled on major mobile gaming websites as far back as March 2012. We took HUNDREDS to IndieCade in September 2013, and did a two-hour keynote talk about it at GameCity in October 2013. Greg designed an achingly gorgeous website for the game, as well as a teaser trailer, and we collaborated with indie game trailer guru Kert Gartner to create our beautiful and mysterious launch trailer. All 50 of our App Store promo codes had been sent out to journalists weeks before the game actually launched. Polygon was very generous in their coverage. Kotaku (admittedly somewhat tongue-in-cheek) named HUNDREDS their best game of 2013.

Looking back, I'm very glad that we took our marketing very seriously. Marketing, for me, is just the way you choose to communicate with the public. Thinking about how to talk to your audience about your game can be an enormous challenge, but forgetting to do it can be catastrophic. There is too much competition. Not in the classical sense of industry titans vying for the same wallet contents, but as we said earlier, the bar for indie games is rising steadily. This is both inspiring and frustrating; it gets harder and harder to get noticed, and when your game sells for $4.99 or less, and takes over a year for a handful of people to make, well, you need a lot of people to notice you if you are going to be able to afford to make another game.

"Postmortem" is almost a misnomer. HUNDREDS is a mobile game, which means it may never be truly done. We've been working on updates and an Android port of the game since launch, which will hopefully have been announced by the time this article goes to print. But, by and large, the game is "out." I'm more proud of HUNDREDS than anything else I've ever worked on. We all learned a lot, and I hope that I've been able to share at least some of that with you here. pm

***

Adam "Atomic" Saltsman makes games in Austin, Texas, where he lives with his family, and is best known for making CANABALT.

THE TOOLS OF GDC 2013

TOOLS AND MIDDLEWARE, STRAIGHT FROM THE SHOW FLOOR

The Game Developers Conference is a great opportunity for tools and middleware developers to show off their latest and greatest to dev studios large and small. Here's a taste of some of this year's offerings.

Project Anarchy

HAVOK // www.havok.com

Havok's middleware technologies, particularly Havok Physics, have been a mainstay in video games for years. The company is determined to take its technology further still, by giving it away to smaller teams interested in developing for mobile platforms.

Havok's calling its new initiative "Project Anarchy"; It's a full-fledged game engine that will be available for download in the spring. It's capable of deploying games to iOS and Android devices, completely free of any cost or commercial restrictions. The tool offers access to Havok's litany of middleware applications, including physics, animation, and AI tools. Complete game samples (and their source code) will be included to give budding developers a taste of the tool's capabilities, and tips on how to get started.

Project Anarchy has a few features meant to ease the pain of cross-platform mobile game development, including viewports that mimic the size and resolutions of a number of different kinds of hardware, and remote input so devs can push their project to a mobile device and test it with a real-time debugger. Havok also plans to launch a Project Anarchy community site, complete with Anarchy's source code, and will encourage the community to develop extensions and share their customizations with other developers.

Gameware

AUTODESK // autodesk.com

Autodesk's Gameware division (the middleware side of the business) had a few upgrades to announce at GDC: AI middleware package Kynapse has been rebranded as "Gameware Navigation 2014" and promises devs full source code access and some neat visual debugging tools, UI middleware Scaleform 4.3 has extended its ActionScript 3 API classes to help its compatibility with Adobe AIR, lighting middleware Beast 2014 adds support for physics-based rendering and support for Open Shading Language, and animation middleware HumanIK 2014 offers a few new mobile-specific features.

Retargeter 4.0

FACEWARE // www.facewaretech.com

Great facial motion capture can add life to a game, but the added time and expense involved can be daunting. Faceware's latest update to its Retargeter application adds a few features aimed at speeding up an animator's workflow.

The Faceware facial-capture process starts with Analyzer, the company's tracking and analyzing software. Load a video into the application, assign virtual tracking points to key elements of the face—spots like the eyes, eyebrows, and mouth—and Analyzer will create a file with motion-capture data that you can use in Retargeter to tweak your character rig.

New to Retargeter 4.0 are the Expression Sets and Autosolve features. Expression Sets are "default" facial animation poses—looking left and right, or frowning, for example. You'll need to mimic these default expressions on the character rig you've assembled in an animation studio like Maya. Here's where the magic happens: The new Autosolve feature combines that data collected from Analyzer and the Expression Set you create, and automatically creates an animation that matches your actor's performance. From this new "starting point," animators can make tweaks or determine what needs to be redone with minimal fuss and loads of time saved—the entire process should only take a few minutes.

Faceware believes that its tech will ease some of the pain and expense of implementing facial motion capture, by requiring less effort than typical rigging situations and allowing for much faster iteration should you need to make corrections. The company also claims that its software will work with footage from any video capture source, though it does sell head-mounted cameras. It will also work within your workflow: Faceware's Analyzer and Retargeter can be downloaded for free from the company's site, so you can take it for a spin yourself.

modo 701

LUXOLOGY // www.luxology.com

Luxology's latest update to modo 3D modeling and animation suite is all about speed. There's less in the way of raw new tools, but Luxology is convinced that modo fans and neophytes alike will be smitten by performance bumps, new, cleaner layouts, and refined workflows.

Modo 701 does offer a few new tricks. The revamped dynamics simulation layer allows animators to create realistic animations with accurate physics modeling. At the show, an artist emulated rubble masonry by creating individual stones, and "pouring" them into a door frame by starting the animation and letting gravity do the work. Modo now also offers the ability to sculpt and direct particles at will, creating some rather impressive effects with little effort.

Performance has been improved dramatically over modo 601, from the selection tools and animation playback down to interacting with schematics or firing up the preview renderer window. It's all aimed at saving time; all of those seconds spent waiting for objects and scenes to load add up, after all. Modo 701 also serves up a revamped interface focused on click reduction: The hundreds of potential keybinds users are already familiar with have been bolstered with new layout-selection tools, pop-up menus that allow you to collapse most of the UI for increased viewport space, and a few customizable workspace options borrowed from applications like Photoshop.

Simplygon 5.0

DONYA LABS // www.simplygon.com

Donya Labs's Simplygon aims to take some of the grunt work out of building level-of-detail (LOD) models out of textured, detailed assets. Its automated LOD-building tools cut and simplify polygons, creating entirely new textured, low-resolution meshes without requiring hours of an artist's valuable time. And Simplygon 5.0 is all about speed and accuracy, introducing new tools that improve Simplygon's automation processes. For example, the new Vertex Reposition function adjusts an LOD model's vertices to get a tighter silhouette and improved texture reproduction, resulting in less jarring differences between LOD models and the original assets. The new Smart Improve feature compares original assets to the LOD Simplygon creates, and automatically tweaks the geometry to reduce visible differences between the two. LOD models created in Simplygon 5.0 can also take advantage of symmetry-aware polygon reduction, which aims to maintain symmetry in a model across a user-defined axis.

Simplygon's LOD tools have also been updated to offer skinning support, preserving animation and skinning data from applications like Maya and 3DS Studio Max when creating low-poly LODs. Donya Labs believes this will be especially useful for maintaining an animated asset's fidelity when optimizing models for mobile devices, or rendering large crowds.

Hansoft 7.0

HANSOFT AB // www.hansoft.se

Hansoft has gone social. The project management system's 7.0 update (released in November 2012) is the company's biggest yet, revamping the user interface and adding a slew of features that will seem familiar to anyone who's spent time on a social networking site. These social features are designed to improve efficiency: As teams grow, they lose the ability to "get together" in a single shared space. A lack of effective communication can lead to headaches, and Hansoft hopes the new update will make coordinating and communicating more efficient,

without forcing large teams to rely on a multitude of tools.

Collaboration is central to Hansoft 7.0, and the application now features a news feed (not unlike Facebook's). Teams using Hansoft can keep track of projects and one another by "subscribing" to the users and groups they're interested in, which essentially filters updates. Here's an example: An executive producer might subscribe to updates from all of her team leaders, getting a glance at and commenting on their progress without necessarily being inundated with updates on every project their teams are embroiled in. The application has also added chat functionality (with optional conversation logging). All of the application's modules can also be popped out of the main interface and placed anywhere on your PC's desktop should you prefer to have chat conversations or just the news feed visible at all times while you work in other applications.

Up next for Hansoft are Dashboards, which will offer a bird's-eye view for project leads and the like. Dashboards will be capable of monitoring all the data that's entered into Hansoft, and should offer an idea of a project's total progress and milestones in simple, digestible chunks. Also coming down the pipe are Mac and Linux versions of the software, so developers who'd rather not use Windows systems can also get on board.

Enlighten

GEOMERICS // [website?]

Geomerics might not be a household name, but there's a chance you're familiar with its work; the company's dynamic lighting technology, dubbed Enlighten, was used in EA DICE's Battlefield 3. At this year's Game Developers Conference, Geomerics announced that Enlighten would be integrated into both the Frostbite 3 engine (for Battlefield 4) and Unreal Engine 4. As of the latest Enlighten update, Geomerics has extended runtime support to include just about everything under the sun, including the upcoming Sony PlayStation 4, Windows, Mac, and Linux PCs, current-generation consoles, and both iOS and Android mobile devices.

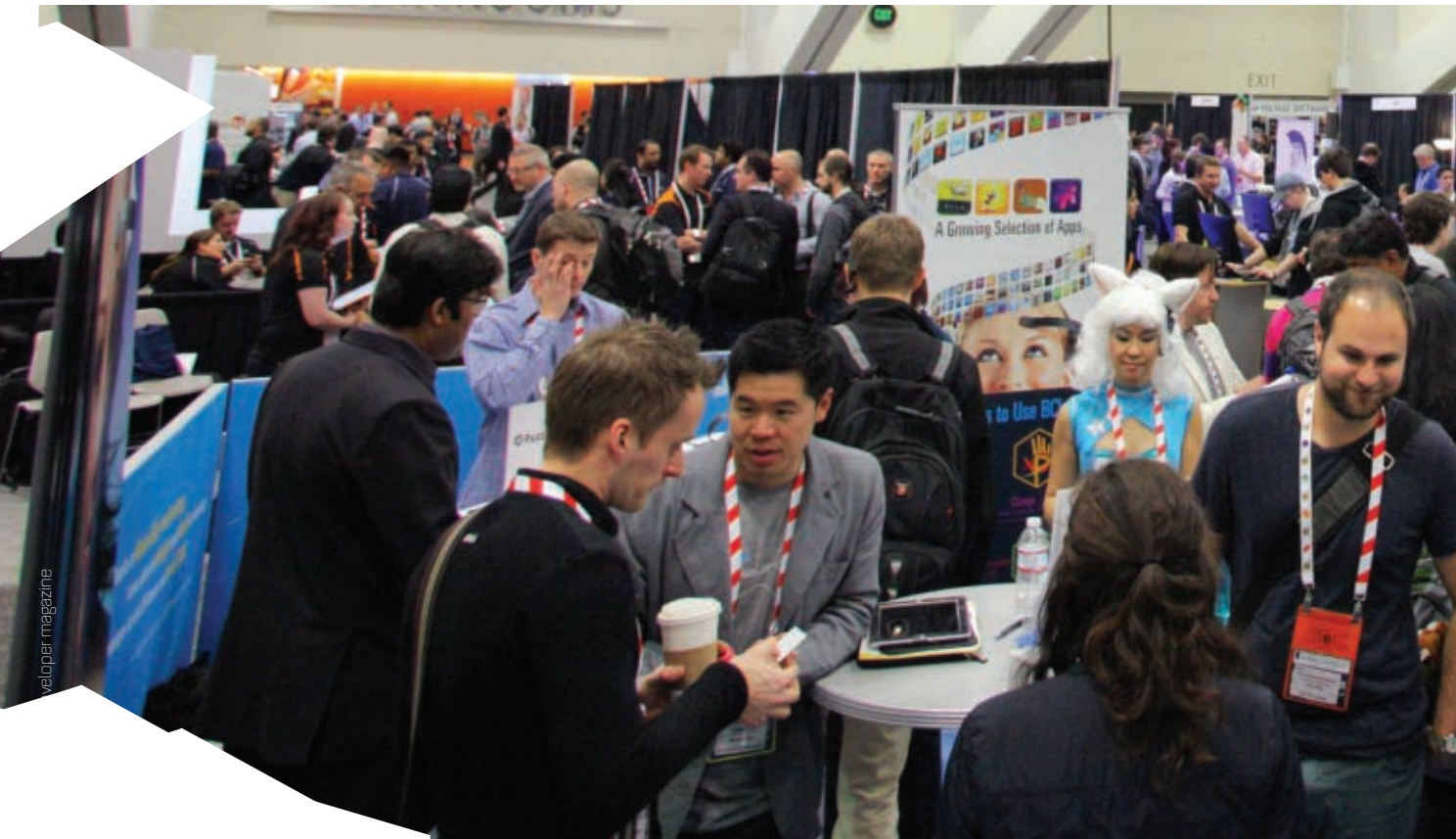
Also new is support for a broader range of lighting models. We were given a tour through an artist's rendition of an ancient ruin to see the effects firsthand: Light spilled into rooms and caverns as the sun rolled across the sky, and crept back out casting shadows all the while—an ample demonstration of how Enlighten tackles static and dynamically lit environments. As befitting a video game-inspired ruin, brightly lit torches burned seemingly without purpose; knock them about, and cobblestones glowed from the reflected light, bouncing indirectly off cavern walls. This served as a case study for the new dynamic specular effects, a computationally expensive process that Geomerics claims it's been able to achieve in real time on mobile devices. On the production side of things, a new plug-in for Maya adds real-time previews of Enlighten's lighting tools into Maya's viewport, so designers can experiment and get a feel for how their models will appear in-game before making any commitments.

Geomerics has also expressed interest in working with more mobile developers to bring Enlighten-powered lighting effects to small screens, but remarked that it isn't ready to make its technology and support staff available to smaller independent developers quite yet, as some training is require to implement Enlighten into a game developer's tool chain. We were told that the first licensees creating content for mobile devices are actually developers already creating content for consoles and PCs, since they're already familiar with the tools.

Perceptual Computing SDK

INTEL // www.intel.com

Intel's Perceptual Computing initiative went into beta in fall 2012, with an SDK release that called on developers to create applications and games that made use of voice recognition, close-range hand and finger



tracking, and facial analysis. During this year's Game Developers Conference, Intel announced the release of the Perceptual Computing Software Development 2013. The crux of the update is that there's no longer a beta tag, and software can now be developed for commercial purposes.

The Perceptual Computing SDK intends to change the way we interact with our PCs, but roadblocks abound: The SDK only works with Intel CPUs on Windows 7 and Windows 8 PCs, and the gesture-tracking controls are currently only supported by a $150 gesture-tracking camera (sold separately). That said, the ability to develop applications using the SDK for commercial use could prove promising. The SDK provides a few sample-use case scenarios and tools to develop applications that are capable of monitoring speech, or tracking faces and hand gestures, and can be downloaded for free. The standalone camera will be available to consumers in stores later this year, and Intel plans to implement the technology from the standalone camera into Ultrabooks at a later date. That $150 could prove to be a sound investment, should consumers be quick to embrace Intel's vision of a finger- and face-friendly future of computing.

Prime 17W

OPTITRACK // www.naturalpoint.com/optitrack/

Motion-capture specialist OptiTrack's latest offering, the Prime 17W motion-capture camera, offers a 70- by 51-degree field of view, with a generous 50-foot camera-to-marker range. It can capture 360 frames per second at a 1.7 megapixel resolution, but costs $3,700. That's a hefty sum, and the costs will only continue to climb as you add more cameras for capturing useful motion data in 3D spaces, or if you opt to use OptiTrack's Motive:Body software.

Should you choose to take the plunge, OptiTrack's powerful hardware and software could potentially streamline your mocap workflow. Setup is simple: Actors strap on their markers and assume a neutral pose,

and the OptiTrack system will create and calibrate a trackable skeleton. On the GDC show floor, the company demonstrated the Prime 17W's precision with a steady stream of actors dancing for the cameras, capturing precise footwork and finger tracking that uses a minimal number of markers to approximate hand gestures and the like.

Extreme Motion

EXTREME REALITY // xtr3d.com

Extreme Reality aims to slash the cost of developing and releasing motion-friendly games by eliminating one of the most expensive components: hardware. The company's Extreme Motion software uses 2D cameras with as low as 0.3-megapixel resolution counts to create accurate skeletal models of players, and insert them into games that utilize Extreme Reality's tech. At GDC, the company gave demonstrations of the software's speed and accuracy—even in a setting with subpar lighting—on meager laptop webcams. Assume a neutral pose with your arms raised in the air, and the software will quickly recognize your skeleton and map your movement data to whatever developers see fit—from games to adding simple gesture controls to user interfaces. Extreme Reality's software also works with the front-facing camera of Apple's iPad, with plug-in support for Unity3D on iOS. The software currently supports Windows and iOS devices; Android support will be coming later this year.

***

Nate Ralph is an aspiring wordsmith fascinated by games, hardware, and most everything in between. You can find more of his musings in 140-character chunks at @nateralph.

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

FROM XNA TO MONOGAME

TIPS FOR MIGRATING YOUR XNA SKILLSET TO MONOGAME

For a long time, the idea of making "real" games using managed languages such as C# was considered lunacy. But things have changed, and managed languages are now proving to be quite viable for making games, thanks in large part to XNA and Unity 3D, with both using C# as their main development language or scripting engine. Many new game devs learned to use XNA in order to get their games on the Xbox 360 and Windows PCs, and even though Microsoft recently announced that XNA will neither see any more active development nor be supported in Microsoft's Metro interface, those devs don't have to start over from scratch. Enter MonoGame, an open-source implementation of the XNA 4 API.

WHAT IS MONOGAME? MonoGame's project goal was initially to allow XNA developers to publish their games on iPhone, but the project has grown a lot since those humble beginnings. It now supports a number of platforms including Mac OS X, Linux, Windows 8, Windows Phone 8, Android, and iOS. For the most part, it is free to use (though you'll need to pay for a license for the Xamarin iOS and Android frameworks) and modify, as all the code is covered by the MS-PL (Microsoft Permissive License). Now, MonoGame's goal is first to produce an XNA-compatible cross-platform API, which can be extended to new platforms as they appear. For example, because MonoGame already supported Android, we were able to add Ouya support to MonoGame in a matter of days.

Once the XNA API is stable, we're planning to extend the API with new features that people would have liked to see in XNA, and extend the API to make it even easier to do certain tasks. For most of the team involved, some of whom have been working on MonoGame for a few years now, this is a long-term project.

GETTING STARTED WITH MONOGAME First off, head over to www.monogame.net/downloads and grab the latest stable release. At the moment, MonoGame supports Visual Studio and MonoDevelop/Xamarin Studio.

If you want to use Visual Studio: The Windows installer will install project templates for all editions of Visual Studio 2010 and 2012. Particular requirements for your target platform are as follows:

- For Windows desktop, you can use VS 2010 Express or higher, or VS 2012 Express for Desktop or higher on Windows 7 or 8.
- For Windows Store, you will need VS 2012 Express for Windows 8 or higher on Windows 8.
- For Windows Phone 8, you will need Windows 8 64-bit and the Windows Phone 8 SDK. This will install VS 2012 Express for Windows Phone, and can also work with VS 2012 Professional or higher. To use the Windows Phone 8 emulator, your PC needs to meet specific hardware requirements; see www.microsoft.com/en-GB/download/details.aspx?id=35471 for details.
- For Android and/or iOS, you will need VS 2010 or 2012 Professional or higher and Xamarin Business or higher on Windows 7 or 8.

To build content to xnb format, you will also need XNA Game Studio 4.0 or the Windows Phone 7.11 SDK installed. This is a temporary measure until our content pipeline replacement is completed. To install either of these SDKs on Windows 8, you need to have the Windows Games Live client first. A full list of the prerequisites and download links can be found here: https://github.com/mono/MonoGame/wiki/Required-SDKs

If you want to use MonoDevelop/Xamarin Studio: MonoDevelop is a free open-source IDE that is available for Windows, Linux, and Mac OS X (www.monodevelop.org). MonoGame is available as a package, which includes the runtime for each platform. You can download this from Codeplex as an .mpack file, and install it via the AddIn Manager within MonoDevelop. Recently, Xamarin has released Xamarin Studio, which is an updated version of MonoDevelop for people wanting to use this IDE. There is also a .mpack file available for that version.

Once you have installed the required package into your chosen IDE, you will be able to create a new MonoGame project. There are a number to choose from, but to get started, you should pick the one that is native to your platform—for Windows, create a new "MonoGame for Windows GL" application; for Mac OS X, create a "MonoGame for MacOS," and so on. Once you have a new project you will see the familiar Game class that renders the customary CornflowerBlue screen.

Before you start coding, there are a few things you should watch out for. For Windows, all of the templates will work out of the box, but if you want to work on an Android project, you will need Xamarin.Android from Xamarin installed. MonoDevelop/Xamarin Studio does include templates for Mac OS X and iOS, but these will currently work only on an Apple Mac because of the underlying development requirements. For iOS, you'll need to install Xamarin.iOS, and for Mac you'll need to grab MonoMac. Also, due to some issues with the AddIn for MonoDevelop and Xamarin Studio, you will need to download the MonoGame source code and add references to the required MonoGame projects for MacOS, iOS, and Android. (The team is working on getting this fixed.)

GETTING THE CODE If you want to get the code for MonoGame, you will need to install git. There are nice user interfaces for git, but most of the team use the command line. So here are a list of the commands you need to get up and running:

```
git clone git://github.com/
mono/MonoGame.git MonoGame
cd MonoGame
git submodule update
init ThirdParty
```

These commands will checkout a read-only copy of MonoGame and the required third-party library to get you started. If you want to make changes and contribute back, you will need to fork the repository. Details can be found here: https://github.com/mono/MonoGame/wiki/GIT'ing-Started-With-Git

CONTENT AND ASSETS MonoGame, like XNA, can make use of .xnb compiled content files. These .xnb files are currently created by the XNA content pipeline; as of this writing, the MonoGame team is working on a cross-platform implementation that will work on Windows, Mac OS X, and Linux, but for now we'll need to use the XNA Content Pipeline. Also it is possible to load some native assets through the ContentManager, so if you have a .png file you want to load directly you can use the same code as you would normally:

```
texture = Content.
Load<Texture2D>
("character");
```

Listing 1: Bloom extract.

```
// Pixel shader extracts the brighter areas of an image.
// This is the first step in applying a bloom
postprocess.
sampler TextureSampler : register(s0);
float BloomThreshold;

float4 PixelShaderFunction(
float2 texCoord : TEXCOORD0
) : COLOR0
{
// Look up the original image color.
float4 c = tex2D(TextureSampler, texCoord);
// Adjust it to keep only values brighter than the
specified threshold.
return saturate((c - BloomThreshold) / (1 -
BloomThreshold));
}

technique BloomExtract
{
pass Pass1
{
PixelShader = compile ps_2_0 PixelShaderFunction();
}
}
```

CLASS	WINDOWS	LINUX	MACOS	IOS	ANDROID	WINDOWS
Texture2D	.xnb .png .jpg .tiff	.xnb .png .jpg .tiff	.xnb .png .jpg .tiff	.xnb* .png .jpg .tiff	.xnb .png .jpg .tiff	".xnb, .png, .jpg"
SoundEffect	.xnb .wav	.xnb .wav	.xnb	.xnb	.wav .mp3 .ogg	.xnb
Song	.xnb .wav	.xnb .wav	.xnb	.xnb	.mp3	.xnb
Model	.xnb	.xnb	.xnb	.xnb	.xnb	.xnb
Effect	.xnb** .	.xnb**	.xnb**	.xnb**	.xnb**	.xnb**
SpriteFont	.xnb	.xnb	.xnb	.xnb**	.xnb	.xnb

Figure 1: Available formats for class types.

** - PVRTC compressed only*
*** - Must be generated by the MonoGame Content Processor*

MonoGame will attempt to load an .xnb file first, but if one does not exist, it will fall back onto known native types for that type of object. This feature was originally added to help people developing for iOS and Mac who might not have access to a Windows machine to build their content. (See **Figure 1** for a table of available formats for class types.)

As you can see from **Figure 1**, on Android you cannot make use of the .xnb files for SoundEffect or Song types; you must use the native type for that platform. This is due to the limitations of the SoundPool and MediaPlayer classes on Android. (Some of

SKULLS OF THE SHOGUN dev 17-BIT used MonoGame to build a Windows 8 version of the game.

these restrictions will be removed as time goes on and the framework matures.) Also, some of the platforms only support .xnb files for certain types; this is because it is more efficient to use the compiled and optimized content for those types. For example, it's more efficient to pre-process a Model at build time into an optimized format than it is to load an .fbx file at run time and decode it on a mobile device.

Fortunately, the MonoGame framework comes with some tooling that we use to extend the existing XNA content pipeline to produce the content we need—though since it relies on the XNA framework, this will only work in Windows. You can create a "MonoGame Content Project" within Visual Studio 2010, which will create a normal XNA Content project and an XNA project that will in turn build the content project. Both have all the required references and MSBuild target imports to allow it to use the MonoGame Processors.

Now, when you add a new asset to the content project and select the type of Processor you want to use, you will see a number of MonoGame-related processors: "MonoGame – Effect," "MonoGame – Texture," and so on. These processors will do some custom processing on the asset to optimize it for the platform you are targeting (see **Figure 2**).

The new Builder project has a number of build configurations for each platform, so if you are building for iPhone you can choose the iOS configuration. If you are building for the Windows Store choose the Windows 8 configuration. The resulting output files will be placed in bin\<Configuration>\Release, so if you are building for multiple platforms, you will end up with a directory for each platform you are targeting. The last step is to add these files to the Content directory of your project—you can do this easily by using the "Add as Link" feature in Visual Studio and MonoDevelop. This way, you can just link to the file without copying it to the project directory, which is great because if you change your content and compile it using the Content Builder, it will be automatically picked up the next time you compile your application. Note that depending on the platform you are targeting you need to set the BuildAction correctly for your assets file to be included in the final application package.

USING CUSTOM EFFECTS If you want to use Effect files from a previous XNA project or an XNA sample, you'll need to process them with the MonoGame Effect processor to compile them for that specific platform. Some of these use OpenGL rather than DirectX as their graphics API, so the Effect file from XNA will need to be converted to OpenGL shader language for it to work.

Rather than have developers rewrite their shaders to GLSL, MonoGame installs some tooling to automatically convert the HLSL in the Effect file to the appropriate shader language for the target platform. For the DirectX-based platforms, MonoGame uses the DirectX 11 tool chain to compile your Effect into a shader optimized for that platform. For the OpenGL-based platforms, the Effect is processed by a tool called MojoShader, which does a low-level conversion from HLSL to GLSL that allows the Effect to work on that platform.

Of course, this conversion process can occasionally introduce errors or unsupported features that the target platform does not support. For example, with OpenGL Shader Model 3.0, you cannot do a texture lookup in a Vertex Shader, so if your Effect uses that feature it probably won't work. Let's take a sample effect from one of the XNA samples available on the Xbox Creators Club website—the Bloom Extract Effect is a good example (http://xbox.create.msdn.com/en-US/education/catalog/sample/bloom). Although it is only a pixel shader, it will give you a good idea of what kind of things we need to look at.

If you look at the code for the effect in **Listing 1**, one of the first things to note is that this effect is using pixel shader 2.0 (ps_2_0). This is fine for DirectX 9, but for OpenGL and DirectX 11 this needs to change. Before we get into the code changes, it is worth noting that you can use conditional defines within effect files to change the behavior depending on the shader model you are targeting. If you are targeting shader model 4, you can use this:

```
#if SM4
// code
#endif
```

to handle that special case. These defines are still valid when using the MonoGame

Content processor, so in order to make this shader support all the platforms we want, we need to update the technique section to handle all the platforms you want to support. In this case, we need to support Shader Model 4 for DirectX 11 and Shader Model 3 for OpenGL/GLES. With that in mind, the Pass becomes:

```
pass Pass1
{
#if SM4
PixelShader = compile
ps_4_0_level_9_1
PixelShaderFunction();
#elif SM3
PixelShader =
compile ps_3_0
PixelShaderFunction();
#else
PixelShader =
compile ps_2_0
PixelShaderFunction();
#endif
}
```

For each type of Shader Model, we define a different Pixel Shader; you can also do this with Vertex Shaders, so a declaration of vs_2_0 would become vs_3_0 for SM3 and vs_4_0_level_9_1 for SM4.

Next, let's look at the parameters passed into PixelShaderFunction. In the sample effect, it only takes one parameter—but in order for this effect to work correctly with the SpriteBatch Effect within MonoGame, we'll need to add the additional parameters. (This is a limitation with MonoGame at the moment that will probably be resolved in the future.) The good news is that adding these extra parameters does not break the effect in normal XNA, so in this case, we can just add the missing parameters like so:

```
void PixelShaderFunction(
float3 position: POSITION0,
float4 color : COLOR0,
float2 texCoord :
TEXCOORD0)
```

Watch out: In XNA 3.1, the PixelShader and VertexShader functions were called PixelShader and VertexShader. This is no longer valid in XNA 4.0 or MonoGame, which is why in **Listing 1** we renamed the vertex and pixel shader functions to VertexShaderFunction and PixelShaderFunction. Of course, you can call them anything you like.

If you apply this technique to all of the effects that are included in the Bloom sample, porting that project over to Windows 8, Android, and iOS simply becomes a matter of using the Content Builder to produce the .xnb files for each platform you are targeting, and then linking to those files for your project.

Listing 2: Optimizing update calls for mobile devices.

```
public class Player
{
  Vector2 position;
  Vector2 scale;
  Matrix matrix;
 public Vector2 Position {
   get { return position;}
   set {
       if (position != value) {
          position = value;
          UpdateMatrix();
       }
   }
 }
public Vector2 Scale {
   get { return scale;}
   set {
       if (scale != value) {
          scale = value;
          UpdateMatrix();
       }
   }
 }
 public void UpdateMatrix() {
   matrix = Matrix.CreateTransform(Position) * Matrix.
CreateScale(Scale);
 }
  public void Update(GameTime gameTime) {  }
 public void Draw(GameTime gameTime) {  }
}
```

TWEAKING MONOGAME FOR MOBILE

So you have a great Xbox 360 or Windows game that you want to port over to a mobile platform. What kind of things do you need to worry about? There are some obvious ones that will come to mind first: screen size, input, and performance. Here's how to handle those factors with MonoGame.

Screen Resolution With certain platforms, like Windows Phone 7 and iOS, you can almost guarantee the screen sizes of the devices you are going to be working on, but with some devices and platforms it's not so easy. Windows Phone 7/8 support hardware scaling, but with Android, Ouya, Windows 8, and the Retina displays on iPad and Mac, MonoGame developers need to take into account the various resolutions they might come up against. Sometimes you'll find yourself navigating a minefield of screen resolutions ranging from 320x200 all the way up to full HD.

One technique that has been used quite well in the past is to use SpriteBatch scaling. With this method, you can design your game to run at a particular resolution, then at run time figure out a scale matrix that will resize your game to match the actual screen size of the device you are running on. This matrix can then be passed to the SpriteBatch, which will scale your graphics. In the following code, we can see some sample code for calculating the scale matrix; in this case our virtual resolution is 800x600.

```
var virtualWidth = 800;
var virtualHeight = 600;
var scale = Matrix.
CreateScale(
(float)GraphicsDevice.
Viewport.Width /
virtualWidth,
(float)GraphicsDevice.
Viewport.Height /
virtualHeight,
1f);
```

With that matrix calculated, we can now call spriteBatch with the extra matrix parameter like so:

```
spriteBatch.
Begin(SpriteSortMode.
Immediate, null, null,
null, null, null, scale);
// draw stuff
spriteBatch.End();
```

This will apply the scale matrix to the items being drawn in that batch. There are other considerations like aspect ratio and letterboxing that you need to take into account, but there are plenty of resources out there, which were originally written for XNA that can be reused in MonoGame and applied to new platforms. (For more on this topic, read this: www.david-amador.com/2010/03/xna-2d-independent-

resolution-rendering/)

At a certain point, however, scaling won't solve your problems. Scale down too much and your graphics will become pixelated; scale up too much and they become blurry. At this point, you need to start looking at the assets. One technique you can use on iOS is have two sets of assets—one for normal displays, and one for Retina displays. You can do this by adding a @2x suffix to the name of the larger asset and including that asset in your game package. The MonoGame content pipeline has been coded to make use of this trick on iOS, so if you have a raw asset or .xnb file which has a @2x version and you are running on a Retina display device, that higher-resolution asset will get loaded. All the developer needs to do it include those assets in the application. (This does make your app package larger, unfortunately.)

The other option is to have two versions of the game, one for normal and one for HD, each with different scales of assets. This way, people have the choice of downloading a larger high-definition game or the smaller normal one.

Handling Input Devices Moving between platforms means you need to support various input devices. On the desktop platforms (Windows, Linux, and Mac), MonoGame uses the Simple Direct Media Library (SDL) to interface with the various joysticks and gamepads that might be available. The inputs from these devices are routed through the GamePad class. SDL does seem to have a problem reading the USB description of some of the Xbox 360 controllers, so on occasion it will fail to apply the correct configuration.

Mouse and touchpad input is one area where MonoGame has diverged from the normal XNA. On Windows Phone 7 and 8, if you use the XNA implementation provided by Microsoft, all of your touch events will be routed through to the mouse as well. This can make it very easy to port games over from Windows and Xbox 360 to those platforms. However, with the introduction of Windows 8, you now have devices which can have a mouse and a touch screen, and it is possible that game devices will want to use these separately. With this in mind, if you have a Windows game that you are porting to a Windows Store App, you will need to add support for touchscreens as your existing mouse code will not be used.

PUSHING PERFORMANCE Both the iOS and Android versions of MonoGame run on top of the mono platform. Mono has a great compiler, and these two mobile platforms also include a linker, which is used to reduce the package size of your application by removing code that is not needed. It also has a really good garbage collector, but even though there have been some major advances in the mono GC in the last few years, you still have to remember you are running on mobile devices which vary dramatically in terms of power (especially when it comes to Android devices). Make sure to look at your Update methods; you might be making work for the GC by creating lots of temporary objects that will just be disposed of. You should also think about whether you actually need to do a particular calculation in a certain way—after all, the fastest code is the code that doesn't get executed. Consider the following class definition:

```
public class Player
{
  public Vector2 Position;
  public Vector2 Scale;
  public void
Update(GameTime gameTime)
{  }
 public void Draw(GameTime
gameTime) {  }
}
```

In this segment, we are defining a player that has a position and a scale. Now suppose we are using the Matrix property of the sprite batch to render this player using the position and scale data, and also we are using that information for collision detection. One easy way to create the matrix is to do it in the update method like so:

```
var matrix = Matrix.
CreateTransform(Position) *
Matrix.CreateScale(Scale);
```

On desktop machines, this would probably work just fine, but on lower-powered mobile devices it could be a problem. Even though the Matrix class is a struct, and cheap to create, we are still making this calculation every call to Update. If your game is running at 30 frames per second, that alone is many matrix calculations that you might not have had to do. Instead, you could declare a field for the Matrix and just update it in the Update method. Or, even better, you could just update the Matrix only when the Position or Scale properties are changed (see **Listing 2**).

I know this is a simple example, but if it gets you thinking about what tiny improvements you can make, then it has done its job. Those of you who know your math will know that you can probably remove the scale and position fields from this class, and just store the matrix.

Another example is handling firing bullets from a gun or ship. One way of doing this would be to have a List<Bullet> where we add new bullets as we need them, and remove them as they go offscreen or out of the playing area. When you add items to a list that cause the list to expand its capacity, you are creating new instances of Bullet each time. A more efficient system would be to have a cache of Bullet instances that we can use to populate an "active" list of bullets that are in use. As bullets go out of play, we can just put them back in the cache.

Of course, you probably don't need an infinite amount of bullets; most games tend to limit the number of bullets or missiles you can fire. In that case, you can keep your cache small so as not to use up too much memory, and you can set a capacity on the active bullets list in advance so that the list is not expanding during game play.

If you are having to create lots of temporary variables while loading levels or doing some other sort of major processing, you can try to call GC.Collect(0) as often as you can. This will ensure that the Garbage Collector collects those temporary items as soon as possible, rather than waiting for the next automatic collection. Also, you will want to try to avoid calling any kind of collection during game play; if the GC does have to do a major collection during an update loop you will no doubt see a definite pause in your game. It might only be for a few milliseconds, but it will be noticeable. Note that unlike GC.Collect(0), GC.Collect() does a full garbage collection across your application's heap. It can take up to 100-200 milliseconds (several frames) so you will want to avoid making calls to GC.Collect during actual gameplay, otherwise it will introduce a definite pause or lag to your game.

Figure 2: Example of optimized internal .xnb formats

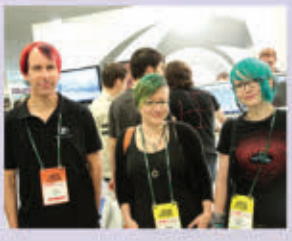
TYPE	WINDOWS	LINUX	MACOS	IOS	ANDROID	WINDOWS 8
Texture2D	DXT	DXT	DXT	PVRTC	DXT	DXT
SpriteFont	DXT	DXT	DXT	PVRTC	DXT	DXT
Song	MP3	MP3	MP3	MP3	N/A	MP3
SoundEffect	PCM	PCM	PCM	PCM	N/A	PCM

GDC 13 EUROPE

CO-LOCATED WITH gamescom

GAME DEVELOPERS CONFERENCE™ EUROPE 2013

COLONGE, GERMANY | AUGUST 19–21, 2013 | EXPO DATES: AUGUST 19–20, 2013

THE GAME DEVELOPERS CONFERENCE™ EUROPE (GDC EUROPE) IS THE LARGEST PROFESSIONALS-ONLY GAMES INDUSTRY EVENT IN EUROPE. JOIN US AT GDC EUROPE, AUGUST 19-21, 2013, FOR THREE DAYS OF LEARNING LED BY TOP INDUSTRY EXPERTS.

MAIN CONFERENCE SESSIONS

MONDAY–WEDNESDAY

- Business, Marketing & Management
- Design
- Production
- Programming
- Visual Arts

SUMMITS

MONDAY–WEDNESDAY

- Independent Games
- Smartphone & Tablet Games
- Free to Play Design & Business

EXPO FLOOR

MONDAY & TUESDAY

Explore the latest in game innovations and network with industry experts

VISIT WWW.GDCEUROPE.COM TO REGISTER

Supported by: European Games Developer Federation | BIU | gamescom | Stadt Köln

UBM Tech

Supergiant Games used a fork of MonoGame code to port BASTION to iPad.

Lastly, don't use too many instances of SpriteBatch. Generally speaking, in XNA games you will want to keep the number of SpriteBatch instances to a minimum; this is true for MonoGame as well. MonoGame's implementation has an internal cache of SpriteBatchItems that we recycle to keep the number of object allocations to a minimum. By default, we create 1000 items in the cache, so the more instances you have the less system memory you will have to play with.

ANDROID AND IOS LINKER When targeting Android and iOS using the Xamarin tool chain, you will need to be aware of the linker that is run against your code during the build process. This linker reduces the code size of your packages by removing unused code in both your code and the framework depending on the linker settings. The result is your final package will contain an optimized mono framework, rather than the entire framework, and any unused code in your game will have been removed as well.

If you are dynamically loading objects through the content pipeline, you'll sometimes get into the situation where the linker doesn't know you need the types you are using, and it removes that code. This will usually result in a MissingMethodException when trying to construct or call methods on these linked away types. The good news is that there are ways in which you can inform the linker that you wish to preserve certain objects as they are and not remove them. One method is to use PreserveAttribute, which is provided in both iOS and Android:

```
#if ANDROID
  [Android.Runtime.
Preserve(AllMembers=true)]
#elif IOS
  [MonoTouch.Foundation.
Preserve(AllMembers=true)]
#endif
public class Example {
  public Example ()
  {
  }
}
```

This will make sure that on both Android and iOS this entire class is not linked away.

There are other ways to control the linker behavior; if you are interested in learning more the documentation for both iOS and Android is available on Xamarin's docs site.

- http://docs.xamarin.com/guides/ios/advanced_topics/linker
- http://docs.xamarin.com/guides/android/advanced_topics/linking

HELP US OUT! This article has only touched on the surface of MonoGame, but hopefully it will motivate you to try it out—and perhaps even make it better! We've got a few things on the horizon that we want people to help out with, like building a fully cross-platform content pipeline, adding new platforms like Windows Phone 8 and Raspberry Pi, making use of DirectX 11, and extending the XNA API even further. So if you want to help out, head over to www.monogame.net and join in. ip

***

Dean Ellis has been a software engineer for the last 16 years. He currently works for Xamarin on their Android platform. Prior to that, he was working on cross-platform C#-based face-recognition technologies using .Net/Mono WPF and Moonlight. In his spare time. he is one of the project coordinators and a contributor to MonoGame.

ART ON THE RUN

TABLET APPS FOR GAME ART

When we talked about stagnation in artist salaries in last month's Pixel Pusher, we noted that one of the reasons that game artists have been treading water economically is that the barriers to entry to our field have fallen pretty dramatically. A decade ago, it was hard to scare up a license for a professional 3D application plus the hardware needed to run it efficiently. There were few schools teaching skills like 3D modeling and animation. To much of the world, what we do for a living was a high-tech mystery.

The democratization of 3D art over the last decade is pretty remarkable: Free tools like Source Filmmaker and Blender have become the foundation of a vibrant hobbyist generation. This hasn't been great for our paychecks, but it is certainly great for our art form—the YouTube stars of today are tomorrow's great game artists. Of course, there are still fairly significant hurdles to overcome if you're an aspiring outsider. Even free software needs a fairly beefy computer and a big monitor—and nowadays, many households don't keep up with the cutting edge of PC technology as mobile and handheld devices take up more and more of people's time and attention. There's a marketing truism to the effect that mobile devices are for consuming information, not creating it. Does that mean that today's iPad kids will have a harder time growing up to hatch the ANGRY BIRDS of tomorrow?

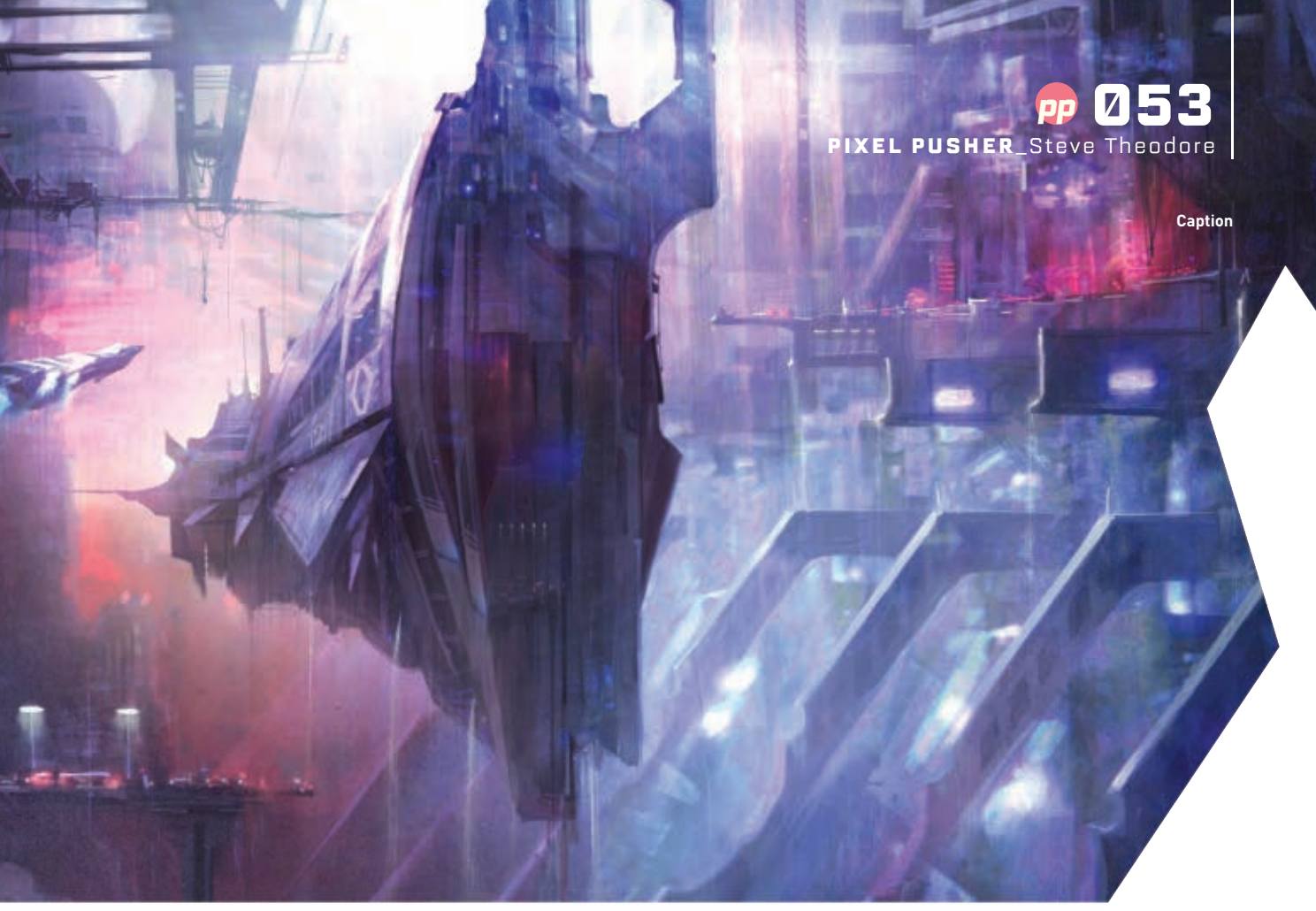
Maybe not. Some of the most interesting recent arrivals in the App Store suggest that the idea that tablets, in particular, are only good for watching content is outdated. So, in honor of the mobile focus of this month's *Game Developer*, we're going to take a look at some of the new crop of apps for painting, modeling, and animating using mobile devices—and of course, we'll round out our survey with some pontifications on the meaning of the new mobile art environment.

2D TOOLS There have been a variety of painting apps for touch devices since the earliest days of the iPhone. Few of them, however, were useful for much beyond casual sketching. The limited processing power of earlier devices made it hard to track subtle gestures and update the screen at the same time, leading to apps that lagged or drew obviously segmented curves where they should have delivered graceful strokes. Worse, the finger is a big, clumsy tool for serious artwork, unless you're still in kindergarten.

Nowadays, though, it's easy to find a conductive stylus that provides a pen- or marker-like grip. Artists are of course notoriously finicky about their brushes, but the stylus market has grown very fast—you can find anything from a $2 plastic job that you won't worry about losing to a fancy Wacom Bamboo stylus that doubles as a real pen and costs around $25. There are even paintbrush styluses such as the Sensu Brush (www.sensubrush.com) with conductive wires mixed in among traditional bristles. These provide pleasant feedback to the digital painter and offer a kind of natural interface to the imprecision of tablet painting. (Note that unlike the hybrid pen-styluses, you really don't want to try using them with traditional paints in between tablet sessions.)

If you're expecting the stylus to completely recreate the experience of a pen, marker, or brush, you'll be disappointed. The smooth glass of a tablet screen provides much less tactile feedback than paper or canvas. Most styli also have pretty large nibs—no needle points here—which makes fine work somewhat tedious and results in a lot of close zooming. Most importantly, tablet touch screens are not pressure sensitive—it's up to the software to simulate the fine control you're accustomed to from using a Wacom tablet. Despite these drawbacks, though, using a stylus still beats finger-painting.

The biggest irritant for many artists who attempt to sketch on a tablet is the fact that a tablet can't tell the difference between a deliberate stroke of a finger and an accidental brush with the heel of your hand. It's the inverse of the charcoal-smudged pinkie that has been the traditional hallmark of art students since the Renaissance: Instead of getting your hand dirty, you end up leaving lots of random splotches on your



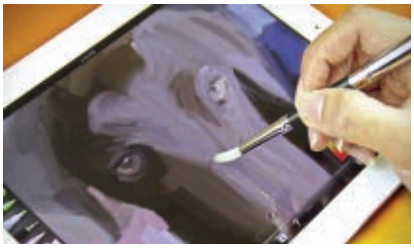
Caption

pristine digital canvas.

Human ingenuity marches on. "Palm rejection gloves" that prevent accidental touches can be had for about $25 from Handglider (www.thehandglider.com). The glove may look familiar to Wacom veterans: It's very similar to the Smudgeguard (www.smudgeguard.com) that many Cintiq users wear to prevent mis-touches or friction on their big tablets. Some users swear the regular Smudgeguard glove can block accidental touches on iPads and Android tablets, but others find the material too thin to stop the touch screen from reacting. In any case, the makers of the Smudgeguard promise an upcoming model dedicated to mobile sketchers.

Of course, the biggest developments in tablet art are driven by growing screens, more powerful processors, and better software. The enormous growth in the tablet user base has encouraged the growth of a number of successful painting programs. Tablet painting apps include many of the common features of their PC cousins: layers, adjustable brushes, and undo, to name a few. Generally these features are somewhat simpler than the PC versions—for example, few tablet programs offer anything like Photoshop's complex blending modes or layer masks—but they offer the intuitive immediacy of painting on a Cintiq tablet at a fraction of the price.

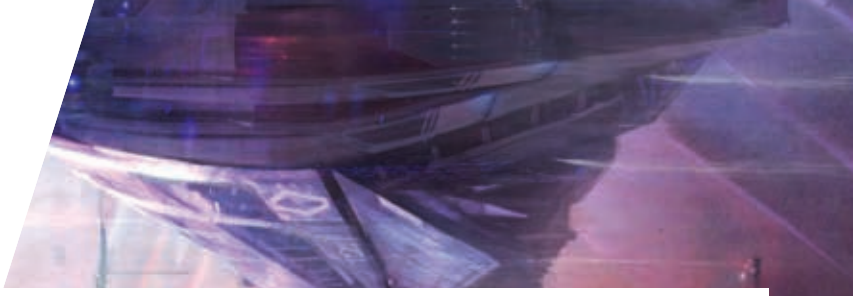
The most venerable painting app is Sketchbook Pro from Autodesk, which is available on both iOS and Android. Sketchbook is one of the oldest art apps,



dating all the way back to the hoary days of 2009. Sketchbook offers much of what you'd expect from a painting app on the PC: layers, multiple brushes, and multistep undo. The app is focused on sketching and digital marker comps, rather than photo editing, so it doesn't offer Photoshop-style filters or adjustments. Its paint and splatter brushes offer some basic natural media feel, but the strength of the package is in industrial design, concept sketches, and roughouts, rather than in painterly rendering. Earlier versions of the program could only create images of the same size as the tablet screen; the current version allows images up to 2500 by 2500 pixels, although the number of available drawing layers is reduced as the images scale up. The interface will be fairly easy for any professional artist to pick up, and most of the time it's just out of the way—which is exactly what you want on a small screen.

The hot (iOS-only) alternative to Sketchbook is Procreate from Savage Interactive. Procreate offers a similar feature set, but with a stronger emphasis on painterly rendering. The selection of paint-style brushes is wider, and the softening effects like smudging or watercolor overlays are more complex and controllable than their equivalents in Sketchbook—though they are still fairly simple compared to the highly tweaked-out specialty brushes of Corel Painter on the PC. Procreate layers offer some Photoshop-style blending modes, and the program also allows users to rotate the canvas—which might seem unnecessary on a device that can be physically rotated, but is quite handy when sketching with the tablet in one hand and the stylus in the other. However, the most appealing aspect of Procreate is its powerful brushing engine, which makes effective use of the iPad's graphics hardware to keep the brushing action silky smooth. To top it off, Procreate can produce canvases as large as 4k, although that's not very practical except on high-res Retina displays. Unfortunately, Procreate is not available on Android tablets yet, but it's a very capable app with a lot of appeal to concept artists and sketchers.

There are a wide variety of other 2D applications on tablets as well. There's a port of Photoshop called Photoshop Touch, which does a fairly good job of porting Photoshop to the tablet. It's not a real substitute for a big monitor and a Wacom tablet, since fingers and styluses are still too crude for really detailed pixel work, but it can be a useful way to review an existing file and—via Adobe's cloud service—even do a little bit of useful work on the bus.



Better suited to the tablet form factor is the large group of sketching apps, such as Paper By Fifty Three, which emphasize quick sketching accessibility over the full painting feature set—handy for capturing an inspiration or recording an impression on the go, but not really suited to what we'd ordinarily call "content creation." Of course, the fact that most tablets come with cameras makes for an interesting set of options for the wandering artist: An app like Paper Camera (www.jfdplabs.com/papercamera/) can turn a chance photo op into the start of a striking concept drawing.

3D ART ON THE GO In a way, drawing on a tablet seems sort of obvious; it's so immediate and direct, and of course we've all seen something similar on the Cintiq. It seems a bit much, though, to expect tablet artists to create fully realized 3D models. Or at least, that's what you might think.

Limited processing power and lack of screen real estate are very significant challenges for creating 3D content on a tablet. These difficulties, however, can be overcome. After all, an iPad 3 is as powerful as the Silicon Graphics workstations that created Jurassic Park and Terminator 2—and, amazingly, it pushes more pixels. Moreover, they can be carried in one hand instead of requiring a 50-pound, 24-inch CRT display.

Verto Studio (http://VertoStudio.com/ipad) shows both sides of the equation. It's a fairly complete 3D modeling program, with primitives and basic mesh editing. Any professional artist will spot the familiar tropes of 3D modeling right away; just drop a cube into the 3D view, grab some verts, and start modeling. It's true that you'll find a lot of more advanced functionality missing—asking an app with only 15 buttons to compete with Max and Maya is a little much. However, if you're long enough in the tooth to remember 3DStudio R4 or the early days of StrataVision, you'll have to admit that Verto Studio offers at least as much power in a far more elegant and accessible package.

The tablet works remarkably well for displaying models—multitouch is a more natural match for typical pan, zoom, and orbit operations than a mouse. As with drawing applications, however, fingers are just too clumsy for really fine work on a smallish screen. The app includes a type-in window for precise control, but it's not a viable replacement for more traditional tools when it comes to modeling convenience and control. It's still a pretty impressive achievement, and a harbinger of things to come. After all, it's not hard to imagine a larger tablet with a more precise stylus being a real portable modeling platform in a couple of years.

The looseness and informality of tablets are a better match for a more expressive, less precise medium like sculpting. ZBrush and Mudbox users will instantly recognize the workflow of Autodesk 123D Sculpt, which is a remarkably competent, if somewhat simple, implementation of brush-based sculpting for iOS. You start with common templates, ranging from geometric primitives to a complete human, and then sculpt with the tools we've all gotten used to since the ZBrush revolution: push, pull, inflate, pinch, and smooth brushes with adjustable size and strength. Notably missing are alpha brushes, so it's hard to achieve effects like the classic rake-pull or instant rock texture. However, it is possible, particularly when working with a brush stylus, to bang out a decent concept sculpt or study quickly. The app even includes integrated 3D paint, allowing for simple 3D airbrushing and even texture projection.

The most pleasant aspect of using Sculpt is that UI flows naturally for the tasks. Any beginning ZBrush artist will suffer through a lot more confusion and frustration than a first-time user of 123D Sculpt, which is an impressive achievement. Performance and level of detail can't compare with what you'd get on beefier machines with more memory, but the results are still quite respectable. As with painting apps, the lack of true pressure sensitivity means the experience is less precise; nonetheless it flows nicely. They might not win many contests on deviantART today, but even five years ago they would have been pretty cutting edge. Here's the real kicker, though: It's free. You do have to pony up $10 if you want to be able to save the sculpts as OBJ files, but it's still a decent 3D sculpting tool you can run for free while riding the bus.

123D Sculpt is part of a raft of new Autodesk apps that seem to be designed to push 3D modeling into the hobbyist mainstream. The family also includes 123D Creature, which extends 123D Sculpt's functionality with a skeleton reminiscent of ZBrush's ZSphere skeletons. This allows for the quick roughout of articulated creatures, followed by a sculp. [should that say sculpt?] There's also 123D Design, a SketchUp-like modeler intended for people who want to create designs for 3D printing.

The most intriguing member of the 123D family, however, is 123D Catch, an app that uses the iPhone camera to create 3D models by stitching together lots of photographs. As you can see from, the results are not what we'd consider shippable art. An iPhone camera doesn't have a depth sensor, after all, so the 3D models are based on image analysis and can easily be fooled by strong colors or odd lighting.

Nonetheless, 123D Catch is a real indicator of what the future will bring. In a few years, it should be possible to whip out a mobile device, spend a few minutes snapping views, and end up with a decent 3D model that can be beamed back to the office through the cloud and turned into a real game asset. Art directors and environment modelers everywhere will have an incredible new tool—and modelers will have another round of that sinking feeling that accompanies every new capture device, from motion capture to 3D scanners. The blog at Makerbot, a popular 3D printing website, ran a piece on 123D Catch under an image of a (3D-captured) Donald Trump announcing "Professional 3D modelers: You're fired!"

Which brings us around to those pontifications and prognostications promised in the first few paragraphs. As you can see, tablet art tools are not—so far—serious competition for the highly evolved tools we use on our big powerful desktops. None of us—except, maybe, for a few globe-trotting concept artists—are going to ditch our workstations for tablets any time soon; the real nitty-gritty work demands a level of precision and control that is still beyond the abilities of small screens and fat fingers. However the rise of tablet apps is pretty startling: Three years after the release of the first iPad there are a number of very useful, flexible semipro tools—and they're incredibly, unbelievably cheap by the standards we're used to: Every app mentioned in this article, plus a brand-new iPad, costs less than single copy of Photoshop.

On the plus side, this means that the somewhat shopworn art tools market will get a much-needed kick in the pants. The interface conventions and toolsets we all take for granted have hardly changed in 20 years; the tools are more polished and powerful, but they work in more or less the same way they did for the pioneers who made *Toy Story* and *The Abyss*. As tablet tools expand and democratize the community of users, we can expect to see some real innovations and fresh approaches to old problems.

Of course, the downside is that our already-fragile technical mystique is further eroded. As hundreds of thousands or even millions of new users get access to the same kinds of tools we use, we'll face even more pressure to find ways to prove ourselves indispensable to our masters. One way or another, these are the tools that tomorrow's artists will be learning on. At least, that's the lesson I draw from trying to keep my kids away from 123D Creatures on my iPad. Dang kids.

***

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.

BREAKING THE RULES

CHALLENGING THE GOSPEL OF SIM-SHIP IN A MULTIPLATFORM WORLD

The 2013 DICE conference was an interesting one. Normally, the show is seen as catering to the big developers and publishers in the triple-A console and PC space, but it has been trying to adapt content and speakers to begin to encompass the broader markets of mobile and social, inviting some smaller independent developers to attend and speak. One of my favorite talks came from this "new" category of developer: Amir Rao of Supergiant Games, developer of BASTION. He spoke about "multiplatformism"—specifically, Supergiant's approach to shipping BASTION across a wide range of platforms and their experience doing so. (You can see the talk yourself on YouTube: http://www.youtube.com/watch?v=NMlMIlX5-pc.)

Rao really got me thinking when he discussed how, during his time at Electronic Arts, it was a default assumption that the optimal launch strategy was to simultaneously launch all SKUs of the game; this is done to maximize the buzz about the game, and to amortize the cost of the marketing blitz across all SKUs.

Rao said that with BASTION, Supergiant serialized launches of different versions of the games. This allowed them to do the port themselves rather than outsource it to a studio that may care less about the title. It also allowed them to do custom tuning for each platform to make the game shine, and in doing so, capture the attention of each platform on which they shipped. This let them get better marketing and promotional assistance with each, which in turn helped sales.

SIM-SHIPPING STRATEGY It was this last item that provoked some thought about the idea of sim-shipping as an assumed optimal strategy. Because they were doing the ports of BASTION themselves, Supergiant wasn't able to sim-ship all those versions, but shouldn't they have preferred to? Even small developers have marketing costs and work on building a buzz about their games. Shouldn't they also want to maximize impact of that effort with one big bang at launch?

Not necessarily—that call depends on a few key differences between EA and Supergiant's respective situations. First, consider the size of the overall marketing "budget" (in both dollars and effort/attention—the latter being perhaps more relevant to an indie developer) especially in relation to the overall game. For a large console title, if spinning up a second, serialized, launch blitz for a second platform costs more than the port itself did, then that blitz will have a material negative impact on the P&L for that SKU. However, for a small title, if it's a matter of working relationships and grassroots PR for a particular platform community, that effort may serialize well, and may need to anyway, as the team has limited manpower.

An even more important difference though, is in *who* is footing the bill for the marketing. If a small title is counting on promotional opportunities on the platform or service's digital storefront, then effectively someone else is picking up the bill for that portion of the marketing. If serializing to tune games better to a platform also allows for someone else to foot the bill for a portion of your marketing, then serialization makes far more sense. (Note that there's room for an entirely separate discussion about how putting all your eggs in a basket of someone who really doesn't care whose eggs he sells is a risky strategy, but I digress.)

DEVELOPING TO AUDIENCE VS. DEVELOPING TO DEVICE A third reason to question the sim-ship "rule" is that the amortization of those marketing costs makes the most sense when you are marketing to a single audience that may choose to access the game across any choice of platform. With the broad spectrum of gamers we see across phones, tablets, consoles, and PCs, this may make less sense than, say, marketing a football title to NFL fans, who then can self-select to the SKU for their console.

So one could imagine a spectrum of marketing-cost sources ranging from "entirely self-funded" to "paid for by platform owner," and another spectrum of marketing budget ranging from "very small portion of total development cost" to "very large portion." Determining where your game falls along these spectra will help you determine whether the sim-ship rule applies.

SIMSHIPPER 2000 There are, of course, other factors that play into the decision to sim-ship—whether the dev team has the time and bandwidth to handle another port without distracting from other titles or sequels, for example. The point is that as the market changes rapidly around us, and as new ways of reaching market emerge, it pays to question all assumptions about how things need to get done.

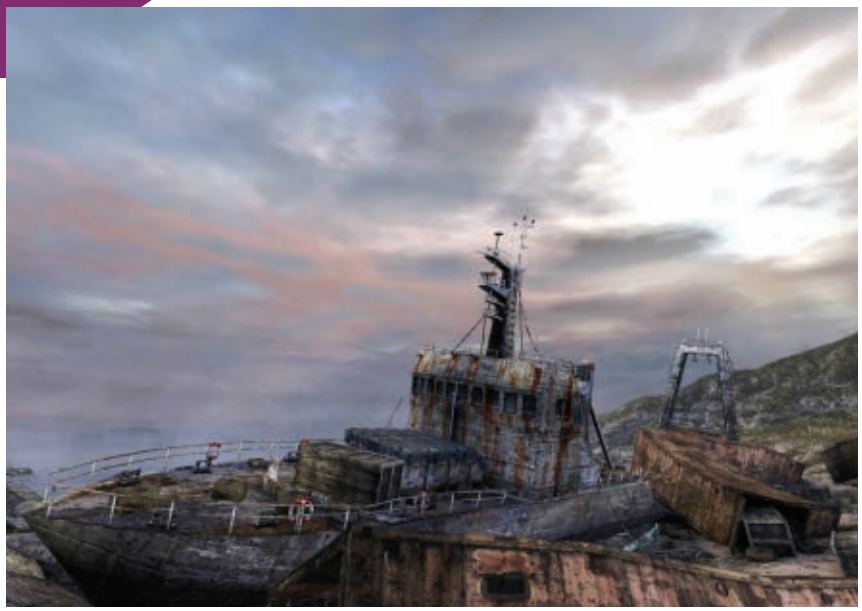
We all have lots of "rules baggage"—sim-shipping, price points, business models, demographics—and we should question all of them. With BASTION, Supergiant showed how by questioning one assumption and deciding that the old rules no longer applied, they were able to define an approach for taking their game to market in a more effective and profitable manner. Which other rules of the game have changed, and how have you prepared to change your approach along with them? b

***

Kim Pallister works at Intel doing game industry forecasting and requirements planning. When not prepping the world for super-cool hardware, he blogs at www.kimpallister.com. His views in this column are his and do not reflect those of his employer.

WHEN CHOICE IS BAD

FINDING THE SWEET SPOT FOR PLAYER AGENCY



Nothing defines video games more than player choice. Interactivity is what separates games from static entertainment forms like film and literature, and when critics accuse a digital experience like DEAR ESTHER of being "not really a game," it's usually due to its lack of meaningful player choice.

However, because game designers hold choice as such an ideal—with phrases like "enabling player agency" and "abdicating authorship"—its downside is often ignored during development, hiding in a designer's blind spot. In fact, every time a designer adds more choices to a game, they make a tradeoff.

With every new option added, your game will gain a degree of player engagement, but at the cost of something else. These costs—too much time, too much complexity, or too much repetition—all can far outweigh the positive qualities of the extra choice.

TOO TIME-CONSUMING If games can be reduced to a simple equation, a possible formula would be (total fun) = (meaningful decisions) / (time played). In other words, for two games with similar levels of player choice, the one that takes less time to play will be more fun. Of course, usually the comparison will not be so obvious; a new feature will add a meaningful decision, but is it worth the extra time added to the play session?

As an example, DICE WARS and *Risk* are similar games of territorial conquest that answer this question differently. In both games, players attack each other by rolling dice, and victors are rewarded with extra armies at the start of their next turn. In RISK, the player decides where to place these armies, which can be a meaningful decision depending on the situation. In DICE WARS, however, the armies are placed randomly by the game, and the result is a much faster game.

Which design is right? While the answer is subjective, the relevant question to ask is whether the combat decisions become more meaningful if the player takes the time to arrange their new armies—or, as is likely, how much more meaningful they become. After all, the player can pursue a more intentional strategy in *Risk*, but is that aspect worth the not-insignificant extra time taken by the army-placement phase?

The answer may depend on the audience (DICE WARS is a casual Flash game, while *Risk* is a traditional board game), but designers should understand the ramifications of their decisions. Sometimes, army placement in *Risk* can be a rote decision, and sometimes, reacting to an unexpected arrangement in DICE WARS can lead to a new, more dynamic type of fun. Ultimately, the aspects of *Risk* that lengthen the play session must justify the time they cost to the audience.

TOO COMPLICATED Besides its cost in time, each choice presented to the player also carries a cognitive load in added complexity that must be weighed. More options mean more indecision; deciding between researching five different technologies feels much different than choosing from 50. Players worry not just about what they are choosing but also about what they are not choosing, and the more options they decline, the more they have to worry about.

Each type of game has a sweet spot for the number of options that keep play manageable—you need enough options to make each decision interesting, but not so many that they overwhelm the player. Blizzard RTS games have maintained a constant number of units per race for decades; STARCRAFT, WARCRAFT 3, and STARCRAFT II all average 12 units per faction. For the third game, the designers explicitly stated that they removed old units to make room for new ones.

Indeed, RTS games as a genre are under assault from their more popular upstart progeny, the MOBA genre, best exemplified by LEAGUE OF LEGENDS and DOTA 2. The original MOBA was a WARCRAFT 3 mod called Defense of the Ancients, which played out like an RTS

except that the player controlled a single hero instead of an entire army.

This twist broadened the potential audience by radically reducing the complexity and, thus, the cognitive demands placed on the player. Instead of needing to manage a vast collection of mines and barracks and peons and soldiers as in a typical RTS, the player needed to only worry about a single character. Consider the UI simplifications made possible by allowing the camera to lock onto the player's hero instead of roaming freely across the map, which forced the player to make stressful decisions about managing their attention.

Of course, this change did take away many of the meaningful choices found in an RTS. Players no longer decide where to place buildings or what technologies to research or what units to train or even where to send them; all these choices were either abstracted away or managed by the game instead. Again, the relevant question is whether these lost decisions were worth the massive amount of complexity they added to a typical RTS.

The success of MOBAs demonstrate that although players enjoy the thrill and spectacle of the large-scale real-time battles pioneered by RTS games, they do not necessarily enjoy the intense demands of trying to control every aspect of the game. Designer Cliff Harris discussed a similar point for his successful alt-RTS GRATUITOUS SPACE BATTLES, which does not allow the player any control of units during combat: "GSB does not pretend you can control 300 starships in a complex battle. It admits you can't, and thus doesn't make it an option. Some people hate it. Over 100,000 enjoyed it enough to buy it, so I can't be the only person with this point of view."

TOO REPETITIVE The final way that too much player choice can negatively affect the game experience is perhaps a bit surprising: Games with too much freedom can suffer from becoming repetitive. A typical example is when a game presents the player with an extensive but ultimately static menu of choices session after session; players often develop a set of favorite choices and get stuck in that small corner of the game space.

Sometimes, a fixed set of options can work if the player needs to react to a variety of environments; the random maps in a CIVILIZATION game can prod the player down different parts of the technology tree. However, almost all games could probably benefit from reducing some player choice to increase overall variety.

Consider ATOM ZOMBIE SMASHER, a game in which players use up to three special weapons (such as snipers or mortars or blockades) to help rescue

"Indeed, would DIABLO be more or less fun if players couldn't actually choose their skills?"

civilians from a city overrun by zombies. However, these three weapons are randomly chosen before each mission from a set of eight, which means the player reacts as much to the current selection of weapons as to the city layout or zombie behavior. Instead of relying on a particular favorite combination, the player must learn to make unusual combinations work, which means the gameplay is constantly shifting.

Similarly, in FTL, the crew members and weapons and upgrades available change from game to game, depending on what the randomly generated shops provide. Thus, the game is not about discovering and perfecting a single strategy but about finding the best path based on the tools available. Put simply, the variety of gameplay in ATOM ZOMBIE SMASHER and FTL emerges because the designers limited player choice.

At the opposite end of the spectrum, games with hefty customization systems usually devolve into a few ideal choices, robbing the flexible systems of their relevance. In ALPHA CENTAURI, players used the Unit Workshop to create units with different values and abilities. However, the most effective combinations soon became obvious, marginalizing this feature.

Thus, giving the player too much control—by offering too many options and too much agency—can reduce a game's replayability. The game would certainly feel different as the loss of intentional progression would turn off many veterans, but the new variety might attract others looking for a more dynamic experience. Randomly distributed skills might force players to explore sections of the tree they would have never experienced otherwise. The important fact is that this loss of meaningful player choice would not necessarily hurt the game.

Ultimately, game design is a series of tradeoffs, and designers should recognize that choice itself is just one more factor that must be balanced with everything else. Even though player control is core to the power of games, it does not necessarily trump all the other factors, such as brevity, elegance, and variety.

***

Soren Johnson was the co-designer of CIVILIZATION III and the lead designer of CIVILIZATION IV. He is a member of the GDC Advisory Board, and his thoughts on game design can be found at www.designer-notes.com.

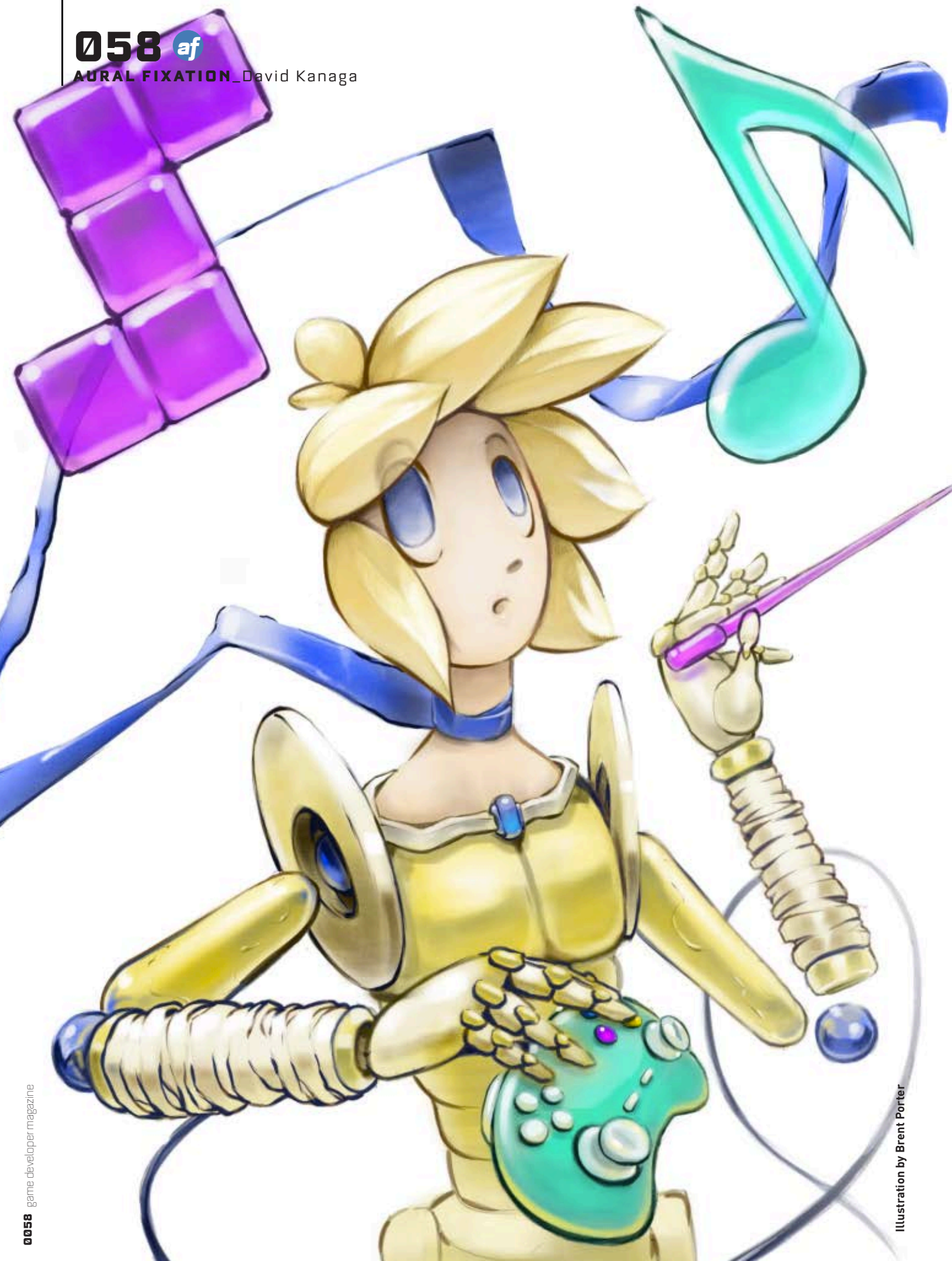


Illustration by Brent Porter

MUSIC > GAMES > MUSIC

FINDING MUSIC IN GAMES AND GAMES IN MUSIC

An "isomorphism" is a 1:1 structural relationship between different forms. As a simple example, we could say that the equations "2x - 2 = 0" and "3y - 3 = 0" are isomorphic, because they both define a variable with equivalent solutions, x = y. Douglas Hofstadter famously hypothesized a relationship between perception of isomorphisms and our experience of meaning (see *Gödel, Escher, Bach*). There has been loads of recent interest, of course, in what it might mean to make games more meaningful.

As events and processes in time, musical structures and game structures can be described isomorphically. That is to say: *games are musics, musics are games.*

I have a theory that the vague concept of "musical meaning" is relevant to finding a new kind of played meaning in games. Designing games as music spaces ("music designing" any game), we can tap into existing structural relationships (isomorphisms, harmonics) in the system/unit operations—and by "hugging"/"skinning" these structures with composed sound materials variably manipulated in ways true to the space and the time-structures of play, we can make their presence sensuously felt. It is in this presence that gameplay becomes musical.

Naturally, we ask: "Is it really true that games are music?"

How do we begin to explore the possibility that they are, or might be?

The following is a practice/method/game/process I've been experimenting with to test the idea that games are music—in two parts, looped:

1/ READING GAMES AS SCORES When designing (composing) music for games—*any game*—try to "read" the game mechanics and structures and feels as a score to be realized sonically, a compositional map. The musical movement exists ready-made in the game's structure, its rhythms, all of its motion. *Be true to these time-structures*. Video games are dense with mechanical variation. This is musical variation.

For every process or change of state in a game, there should be a corresponding process or change of state in its soundtrack. So, don't think in terms of background music and sound effects, but rather events, states, processes, textures, rhythms, and forms. There is no sound design or composition, only improvisation and music design—improvisation plays, and music design builds.

Now, if you embrace this method, you will have to compose spaces that tend toward fluidity and disorder (because they're being played), built of modules and processes as opposed to full "pieces." Depending on the game, many modules could be required, perhaps as many as there are sprites or 3D models in the visual design, and producing all of these might be a lot of work. To make this tolerable, *play all of it*! Improvise the music, don't get caught up in how things sound. It doesn't matter, everything is music. Music doesn't need sound, only movement.

Algorithmic techniques are useful, but don't lose track of the touch of musical play (we'll touch on this in the next section). It might be helpful to practice compositional techniques on video footage. For example, take the "Mickey Mousing" technique (the process of scoring music events to coincide with motion-image events); while it's perceived humorously in movies, the fact is that games are music, and as played music, they are effectively musical instruments. In other words, we can think of Mickey Mousing in games as the process of designing a musical instrument such that it can produce sound.

Think of your music-design process as adding vibrational effects (real perceptual experience) to informational architectures (music spaces as virtual compositions/scores). Graphics (lights, images) do this too; they are information sensualizers—but graphics are only seen in front of us while music surrounds us.

2/ PLAYING MUSIC When you're playing music outside of video games, try to find spaces that you love independently of video games. Continue to explore new territories (play spaces): Improvise, "de-quantize" (an aesthetic-ethical tactic of turning constants into variables described in Adam Harper's excellent book of music space theory, *Infinite Music*), find music in all things. We don't know what kinds of play spaces are possible, so finding new spaces in play is essential research.

We can too easily grow accustomed to all sorts of value-qualifiers as to what music "should be." If we are unable to push past these boundaries, to feel the infinite potential and presence of music in all situations (project of John Cage's 4'33"), we will be hesitant to read a game structure as musical (it might be arrhythmic, for instance). Find new music, new movements—we cannot consider motion to be separate from music.

/\/ Music design is quantitative manipulation of qualitative affect—experience becoming numerical value, becoming variable, becoming experience. Numbers, harmonics, vibrations. Our conception of systemic design should not be opposed to sensuousness or irrationality. On the contrary, music design will need to be totally irrational in its pursuit of real subjective experience, while at the same time highly disciplined in its engagement with quantity.

Imagine game music design as a similar process to following music notation or data visualization's leads in making raw quantitative data intuitively knowable, tangible, playable. It is in transforming material presence that we experience the nonrational, qualitative time flows of differences, repetitions, harmonies, textures—these qualities that describe the lived space of musical playing. af

***

David Kanaga is an improviser and music designer. He has produced award-winning dynamic scores for the indie titles DYAD and PROTEUS. He is currently researching smooth and shifting dimensionality in improvised music spaces with Ilinx Group and working on music designs for morphing landscapes in Panoramical.

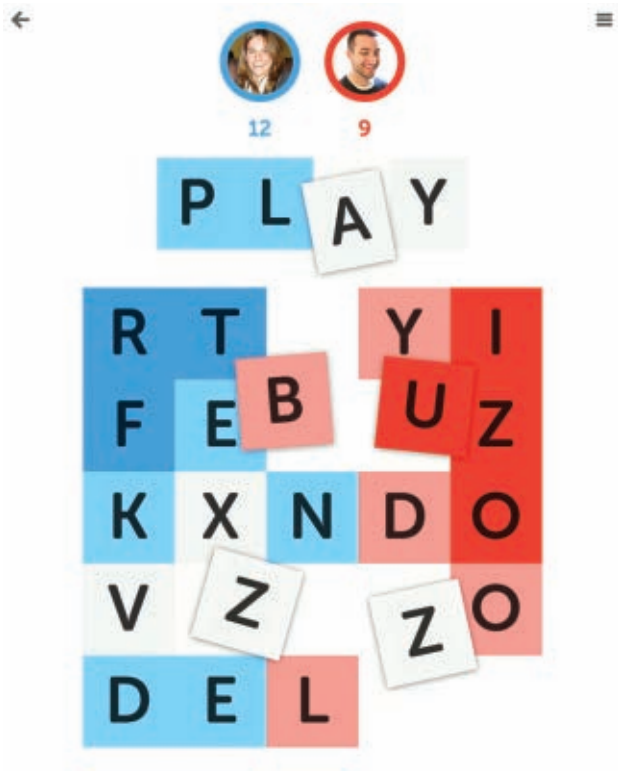
SIMPLE BEAUTY

LESSONS FROM LETTERPRESS'S INTUITIVE UI AND GAME DESIGN

Atebits's LETTERPRESS was one of the more popular iOS apps of late 2012, with 60,000 downloads on its first day alone. It was one of those games that everybody suddenly seemed to be playing, and it's been in constant play on my iPad since release. This success is warranted—on top of being a fun, competitive word game, it has a fair amount to teach us about intuitive UI and game design on touch devices.

For those who haven't played, LETTERPRESS is a turn-based word game about claiming territory. You have a grid of letters, and you're supposed to make the longest words you can with them. The letters needn't connect—you can use any on the grid, and gain a point for each letter you use. But when your opponent uses your letters in their words, they steal the points back (and vice versa). Any letters you've surrounded with other letters you've captured become "saved." Your opponent can still use these letters, but they won't get a point for them. They can use the letters surrounding your "saved" letters though, and thereby make them available for capture next round, if you don't save them again.

The game fixes one of the problems I've had with word games in the past, because it encourages you to make the largest, most impressive word at any given time. Rarely can you find an impressive word and not be rewarded for its use, unless strategy suggests you wait. Now let's get into these UI and design lessons a bit more.



LEARNING FROM *LETTERPRESS*

Start the game = play the game. When you start the game, you see an option for "new game," and one for "about," which has information about how to play and who made it. That's all there is. This initial area becomes your game screen—all the games you've ever played (or are playing) are stored here. Importantly, your current games are on top, and "new game" is relegated to the very bottom, because you often rematch from the win or lose screen. It's very cohesive, and makes a lot of sense!

Many console games still have a "start game" splash screen, when you really only need to use that once. LETTERPRESS is more efficient; the starting screen launches games, and is never useless.

Show, don't tell. The game board is quite spare in its design, but everything is utilitarian, and nothing leaves you guessing. There's a grid of letters, and two icons. One icon represents you and your Game Center ID, one represents your opponent's. An arrow under the icon shows whose turn it is. Captured letters are a light colored shade, saved letters are a darker shade of that same color (and your opponent has a different color). You instantly know how well you're doing by looking at the game board, even if you don't look at the score.

Even better, this is represented by chunky pixels, "zoomed out," in the main game screen, where you can see all your games. A micro version of your board is displayed there, which gives you an idea of what's happening in games you're playing before you even enter them. Everything is subtly and clearly laid out, and obvious.

Meet logical expectations. Spare though it may be, everything in the game does something, and trial and error is rewarded. Touch an avatar icon, and see the word that you or your opponent played last. Touch an empty space in the game board, and see where that letter is in the current word you're spelling (which is helpful when strategizing), and you can easily rearrange your letters in the playfield. It even tells you if a word you're trying to play would be acceptable if you hit the submit button when it's not your turn. If you hold the "clear" button, you can save the current word you've spelled, even if it's not your turn. Anything you might try to do has been anticipated and integrated into the UI design.

Teach through interactivity. On the main game screen, you can move the whole screen around if you choose to, because nearly everything moves when you touch it. This made me wonder whether touching and dragging one game from my game list would do something, and indeed, moving it all the way to the right reveals a prompt to remove the game from your list. It was something I thought would be nice to do, I

tried it, and there it was. That kind of movement happens throughout the game—you can slide around any pop-ups or notifications, or toss them off the screen. That's intuitive design! This didn't need to be explained to me—I was able to figure out everything I needed to know by simply touching and sliding things, as is natural on a touch platform.

Details matter. LETTERPRESS is very simple, but it has a few excellent little design flourishes. If you pick up a letter, it jiggles with life, and tends to tilt in the direction you're swiping. As you bring it up to the live area where you build words, the player avatars jump pleasantly out of the way, using a classic animation bounce. It makes just interacting with letters feel good, which is important, since that's all you really do in the game!

Monetize intelligently. LETTERPRESS is free, but the monetization scheme is smart. If you pay 99 cents, you get more color themes for your board, the ability to start more than two games at a time, and the ability to see all prior words played. None of these things grant a real advantage over a free player, but they do make things a bit more convenient, without making the free game any less convenient, or feeling like something truly valuable was hidden.

LETTERPRESS basically asks you "How much do you want to play this game?" If the answer is "a lot," you'll spend those 99 cents. Nothing forces you to buy anything, nothing compels you to tweet—the game's success was natural, simply because it was good. And the social networking elements of this game are all hidden in the "about" section (surprisingly subdued, considering LETTERPRESS was made by Loren Brichter, the same dev who made the Tweetie iPad app which got snapped up by Twitter itself).

MAKE IT WHOLE The lessons LETTERPRESS teaches are obvious, yet somewhat subtle. It has a holistic design, and everything you expect to be able to do can be done. With such a simple game, it is very possible to meet players' expectations, and yet we so rarely see this kind of well-thought-out design. The extra touches to the menus, the way you can instantly get a grip on how your game is going, and the seamless integration of everything into the core experience all show that the UI is an actual part of the game—it's not a skin, it's not a barrier, it *is* the game.

UI should not be a gatekeeper of content or an afterthought. It should be an integrated part of the game's experiential design. UI should be part of the game's core. While LETTERPRESS may not explicitly teach you how to do this, it's a great example of thinking about your game as a holistic product. Try the game for yourself and see if you don't agree! 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. Follow him on Twitter via @necrosofty.*

CONTRACTOR CORNER

ADVERTISER INDEX

COMPANY NAME	PAGE #
ART BULLY PRODUCTIONS	62
CENTURION ART DEVELOPMENT	62
COSTA IMC	C5
EPIC GAMES	05
HAVOC	02
THE MIT PRESS	57
REASEARCH IN MOTION CORPORATION	C2,C3,C4,001
RAD GAME TOOLS	C6
SHARP SHADOW STUDIO	62
SUPERGENIUS	62
TECHNICOLOR DIGITAL PRODUCTION	62

IN ASSOCIATION WITH UBM TECH	PAGE #
APP DEVELOPERS CONFERENCE	20
GAMASUTRA	30
GAMASUTRA JOBS	63
GAME CAREER NETWORK	44
GDC EUROPE	50
GDC NEXT	10
GDC VAULT	61

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

ALL SYSTEMS GO READY FOR THE LAUNCH OF OUR NEW ONLINE GAME!

07-16 20:09:20
Revision: 12412

* fixed localization text-wrap bug (nice try, Jeff)
* locking repo for final launch deploy!!!

07-16 22:01:01
Revision: 12413

* fixed polling ping-time bug in matching players across regions that might have caused small delays before game launches. There might have been a hiccup on launch day if we hadn't caught that... good job, QA
* finally the work we've been doing for the last three years is gonna pay off!
* relocking repo for final, final, final deploy
* come at me, bros!!

07-17 00:17:24
Revision: 12414

* emergency fix to address "Unable to Join" error

07-17 05:20:49
Revision: 12415

okay, this should address most of the launch volume issues that players have been experiencing:

* more error checking added to "Region Unavailable" crashes
* another fix for "Unable to Join" error seen by players in Pacific time zone due to a floating-point error reporting that the game doesn't actually unlock for 277 years (Jeff, I am looking at you very sternly right now)
* dynamically switch a region's hosted server if the ping is too long
* set available servers to the actual number of servers we have available (whoops, that's mine, haha)

07-17 07:52:41
Revision: 12416

rolling back change #2416

07-17 11:30:10
Revision: 12417

* removed rendering of all citizens' pants to decrease server sync load

07-17 11:33:55
Revision: 12418

* re-added pants. something about ESRB? whatever
* removed rendering trees instead

07-17 14:11:12
Revision: 12419

* disable high-level spells that use too many server-side particles (can't believe I just used the phrase "server-side particles"...JEFF!!!!)
* boost power of low-level spells to take the place of the spells we just removed
* add free spells for players so they aren't mad that we took away the high-level spells
* I did something else with spells that I forgot
* I would really like to go home
* Jeff is a smelly doofus

07-17 15:18:40
Revision: 12420

* HEY, did you know that APPARENTLY, to cast a spell in our game you have to query the player authentication database, query the player database, query the spell database, AND query the player action table?
* and did you know that if you're a mage, it does all those EVERY TIME you press the attack button?
* I am laughing and crying right now
* I don't know how to fix this
* I'm actually just crying

07-17 17:11:12
Revision: 12421

* reduced queue size and added ghost entries to buffer queuing more
* I'm not sure that's a thing. well, it's a thing now

07-17 23:51:12
Revision: 12422

* more queues, less syncs, god I don't even know
* come make games, they said. lots of fun, they said
* I have two degrees in computer science and am imagining the utter shame on my professor's face right now
* drifting, I'm just drifting through infinite black space
* I just wanted to make something fun that people would enjoy... did I ruin kids' childhoods? like, is this going to be a terrible memory that haunts them for the rest of their lives?
* it appears that I am out of whisky
* not sure what this commit is

ad

***

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.

ILLUSTRATION: JUAN RAMIREZ

There is a country that has the softest coffee and also the best offer in video games.

Would you like to know which one?

THE ANSWER IS CO COLOMBIA

Colombian talent, high level technology, and competitive prices are just a few reasons why our industry has been growing exponentially in the last few years.

Take advantage of everything we have to offer. Our companies have developed award-winning apps and they are ready to start providing you with their best work.

For more information please visit:
www.proexport.com.co/eng/ITservices
or contact one of our representatives in your area.

www.
PROEXPORT
.com.co

Government of COLOMBIA

PROEXPORT COLOMBIA
TOURISM, FOREIGN INVESTMENT AND EXPORTS PROMOTION