



AUGUST 2012

gd

GAME DEVELOPER MAGAZINE



FIND THE
RIGHT AI
FOR YOUR
GAME

DEAR

ESTH

ER

GO

B

BEYOND

SPATIAL

VOL

19

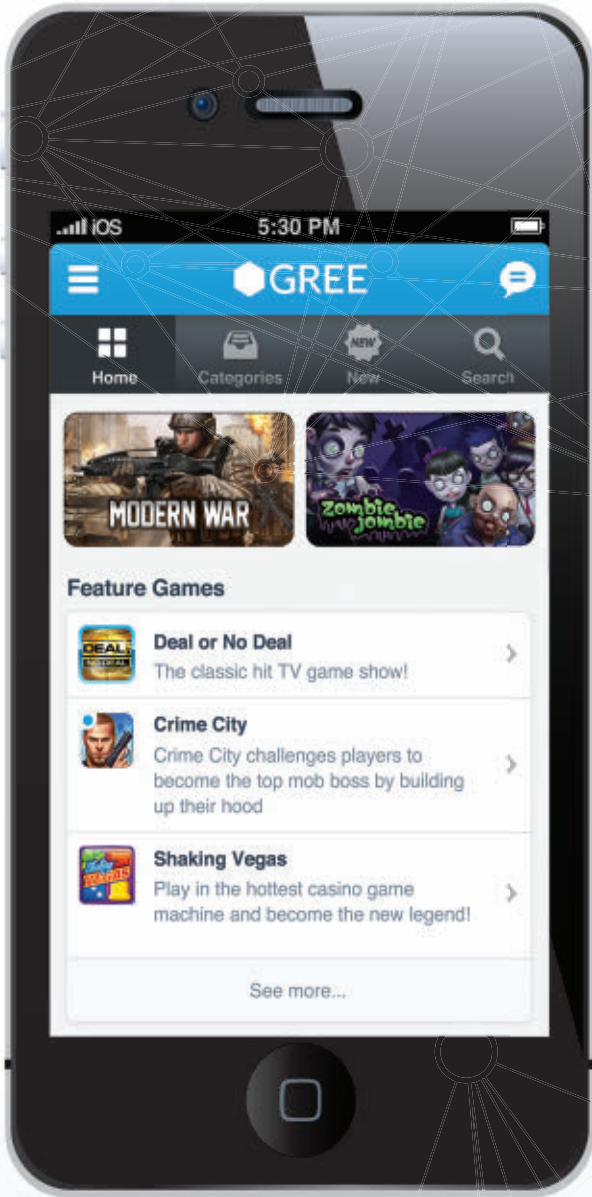
NO 8

INDUST

THE L



POSTMORTEM



Join the World's Largest Mobile Social Gaming Network

Get the SDK at
developer.gree.net

Come Play on GREE Platform:



gAMENAUTS



INFINIDY



CONTENTS.0812

VOLUME 19 NUMBER 8

SPATIAL ANALYTICS

postmortem

22 DEAR ESTHER

DEAR ESTHER started as a narrative-driven Source Engine mod made as part of a research project, and turned into a standalone cult hit that made back its Indie Fund investment within six hours of its release. In this postmortem from thechineseroom, writer and project lead Dan Pinchbeck explains how the team created a powerful game that took the "S" out of "FPS."

By Dan Pinchbeck

features

7 AI ARCHITECTURES: WHAT'S ON THE MENU?

How do you choose the right artificial intelligence architecture for your game? Even if you do your homework, it's not always easy to know which architecture best fits your game genre, design, and development workflow until you dive in. AI consultant Dave Mark walks you through the most prevalent AI structures to help find the one that best complements your game.

By Dave Mark

15 BEYOND THE HEAT MAP

Get better game analytics, and you'll be better able to understand what makes your players tick. Heat maps are only the tip of the iceberg, though. You can use telemetry data, geographic information systems, and trajectory analyses to fine-tune your game even further, and with greater control.

By Anders Drachen

departments

2 GAMEPLAN *By Brandon Sheffield*

The Aftermarket

4 HEADS UP DISPLAY *By Staff*

Quick looks at a new interactive fiction development tool, updates to Commodore and Amiga preservation packages, and the Seven-Day FPS Game Jam

31 TOOLBOX *By Alexander Brandon*

RAD Game Tools's Miles Sound System

35 THE INNER PRODUCT *By John Szczepaniak*

A Basic History of BASIC

41 AURAL FIXATION *By Alexander Brandon*

Audio in Unity 101

44 DESIGN OF THE TIMES *By Soren Johnson*

When Digital Meets Physical

47 BUSINESS *By Paul Taylor*

In Defense of Paying Once

48 PIXEL PUSHER *By Steve Theodore*

Tread Lightly

50 GDC NEWS *By Staff*

DC Europe adds Journey, Goldeneye 007 postmortems

51 GOOD JOB *By Patrick Miller*

Q&A with Taiyoung Ryu, new studios, and who went where

53 EDUCATED PLAY *By Patrick Miller*

LOVE PUNKS

56 ARRESTED DEVELOPMENT *By Matthew Wasteland*

Vox Populi

gd

GAME DEVELOPER MAGAZINE

EDITORIAL
NEWS
REVIEW
PROGRAMMING
SOUND
DESIGN
BUSINESS
ART
NEWS
CAREER
EDUCATION
HUMOR





THE AFTERMARKET

FOR SMALLER TITLES, YOU CAN NO LONGER PLAY THE WAITING GAME

I recently watched a trailer for a game by Diego Garcia and Emmett Butler called HEADS UP! HOT DOGS. It's an amusing iOS game about dropping hot dogs on people's heads as they walk by at various speeds. Some of them bounce up and down, making it tougher for the dog to settle. Cops try to shoot your hot dog out of the sky. The game has a nice art style, good music, and seems like quirky fun, if a bit light. As soon as the trailer finished playing, I thought "I reckon that's about 99 cents. I'll go buy it." I went to the App Store and found...nothing.

Turns out the game isn't due out until fall 2012. The trailer got coverage on a few major blogs, and for a game of this size, that's about all it's going to get. By fall, who will remember the little game about dropping hot dogs? Even if I do remember it, will I still want it then? Will Kotaku want to write about a game of this size a second time, no matter how quirky? With the press, your first shot is when you convey the excitement of a New Thing. After that, the thrill is gone, and at best you'll get a "remember that game? It's out now," if that.

The pace of the game world is speeding up, and the window of opportunity for promotion is changing accordingly. I think that for indie games on PC, iOS, and Android, the bulk of your marketing should thus be after the game is already on the market.

A SMALL CASE STUDY

» My friend Tim Rogers of Action Button Entertainment released a game called ZIGGURAT on iOS about four months ago. Before the game's release, he kept putting out hints about it—releasing trailers, images, and the like—and he would always have people asking where the game was. After all, he had been talking about it, so clearly it must be out, but the people looking for it couldn't find

it, so there must be some mistake.

99-cent App Store games have become the impulse purchase of the game market. I don't want a Reese's Peanut Butter Cup right now, but when I'm in the checkout line waiting for the pink-haired old lady in front of me to take out her checkbook, it starts to look pretty compelling. When I'm avoiding work for five minutes to look at a video of a new game, that's my checkout line. I want to buy it right away, and I don't want to be told to come back later.

ZIGGURAT had two spikes in sales, which were related to two favorable reviews from larger publications, and a postrelease YouTube mock infomercial Tim created. The most important lesson is that while Tim did a good job promoting the game before its release, the major sales came when promotions hit postrelease—not when the game first went on sale. In his case, releasing too much good information to the press before the game was out might actually have hurt sales.

It's been well established that any barrier between the game and the player is a significant loss of revenue. In the case of HEADS UP! HOT DOGS, I thought it was interesting enough to buy. But how will I remember to check when it's released? If industry blogs never talk about it again, or nobody links it to me once it's out, I won't know. What if the developers used up all their goodwill with that first trailer push? Will potential players even remember the name?

A NOTABLE EXCEPTION

» SUPERBROTHERS: SWORD & SWORCERY EP is an exception in a lot of ways, but we'll just talk about the trailer here. The team released a teaser trailer well before the game came out, and it got people excited. The difference is that S:S&S EP has a very defined

visual and sonic style, and the video was all about tone, not gameplay (in fact, it barely even told you what the game was). Unlike the HEADS UP! trailer, the S:S&S EP trailer had a voice and a direction, but no detail.

Here, then, are my ideas for the modern world of iOS and PC indie game marketing.

If your game is high-concept:

Tease it with video. Establish a tone and authorial voice for your game, and present developer diaries in that voice. Don't be too specific, and keep all communication with the outside world in the game's voice. After the game is out, promote further in this manner, but also discuss your game frankly in your own voice as well. When you're releasing game updates, return to the "game voice."

If your game is anything else:

Do not release an expository trailer before your game's release. Tease it with images and sounds, but don't show a trailer. Keep your public info to just your loyal followers, fans, or friends. Do accept feedback, and discuss your game publicly and answer questions if asked—it's always good to let people feel involved in your work—but don't release a trailer or do a real press push until the game is out. After that, go wild.

In the world of triple-A games, building buzz is still the name of the game, but that's because you have to have a big brick-and-mortar launch, and you need to be at the front of players' minds when they enter that GameStop. But for smartphone, PC indie, and other smaller-scale platforms, you won't have the budget to get tons of trailers into their brains. For most games, you've got one shot to get people excited. Make sure your game is out when it comes.

—Brandon Sheffield
twitter: @necrosofity



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
e: gamedeveloper@halldata.com
www.gdmag.com/contactus

EDITORIAL

PUBLISHER

Simon Carless e: scarless@gdmag.com

EDITOR-IN-CHIEF

Brandon Sheffield e: bshffield@gdmag.com

EDITOR

Patrick Miller e: pmiller@gdmag.com

MANAGER, PRODUCTION

Dan Mallory e: dmallory@gdmag.com

ART DIRECTOR

Joseph Mitch e: jmitch@gdmag.com

CONTRIBUTING WRITERS

Anders Drachen, Dan Pinchbeck, Alexander Brandon, John Szczepaniak, Steve Theodore, Soren Johnson, Paul Taylor, Matthew Wasteland, Mike Rose, Leigh Alexander

ADVISORY BOARD

Mick West Independent
Brad Bulkley Microsoft
Clinton Keith Independent
Brenda Brathwaite Loot Drop
Bijan Forutanpour Sony Online Entertainment
Mark DeLoura THQ
Carey Chico Globex Studios
Mike Acton Insomniac

ADVERTISING SALES

GLOBAL SALES DIRECTOR

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

MEDIA ACCOUNT MANAGER

Jennifer Sulik e: jennifer.sulik@ubm.com
t: 415.947.6227

GLOBAL ACCOUNT MANAGER, RECRUITMENT

Gina Gross e: gina.gross@ubm.com
t: 415.947.6241

GLOBAL ACCOUNT MANAGER, EDUCATION

Rafael Vallin e: rafael.vallin@ubm.com
t: 415.947.6223

ADVERTISING PRODUCTION

PRODUCTION MANAGER

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

REPRINTS

WRIGHT'S MEDIA

Jason Pampell e: jpampell@wrightsmedia.com
t: 877-652-5295

AUDIENCE DEVELOPMENT

AUDIENCE DEVELOPMENT MANAGER

Nancy Grant e: nancy.grant@ubm.com

LIST RENTAL

Peter Candito
Specialist Marketing Services
t: 631-787-3008 x 3020
e: petercan@SMS-Inc.com
ubm.sms-inc.com



UBM

WWW.UBM.COM

with 180 million Arabs under the age of 25*...



...the gaming industry is set to boom in the Arab world.

twofour54° Abu Dhabi – the tax-free gateway to a new world of gamers

The MENA region is one of the world's fastest growing media and entertainment markets with 19% growth in recent years. And with 80% of under-25s owning mobile phones*, strong broadband take-up and new gaming innovations, it's a prime opportunity for gaming businesses. Over 100 leading media companies are already capitalising on the opportunity at **twofour54° Abu Dhabi**.

- 100% company ownership in a stable, tax-free environment
- Unique campus environment with facilitated business networking
- The region's only stereoscopic 3D Lab
- **twofour54°** gaming academy in partnership with Ubisoft®
- Easy licensing and business set-up services
- Guidance and liaison with UAE content regulatory bodies
- Dedicated fund for mobile apps development via Apps Arabia™
- Full on-site HD production and post-production facilities

Find out how we could help grow your business today.

twofour54.com/gaming
+9712 401 2454



twofour54
Abu Dhabi

media & entertainment hub

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



INTERACTIVE FICTION FOR ALL INKLEWRITER PUSHES INTERACTIVE FICTION PAST THE PARSER

The age of accessible platforms coupled with a hunger for deeper stories have set the stage for interactive fiction games to flourish, but longtime IF writer and game industry veteran Jon Ingold (FAIL-SAFE, ALL ROADS) believes the tools to create storytelling games have to be accessible, too.

That's why he and his colleagues at Cambridge, U.K.'s Inkle Studios, a software company founded by game developers, created Inklewriter (www.inklestudios.com/inklewriter), a

especially those unaccustomed to text adventures, might be frustrated in their attempts to get the game to understand what they're trying to do. It's a challenge for designers, too, who might find themselves limited by the steep challenge of creating affordances for every option a player might want to execute.

Ingold believes it's the steep barrier to entry posed by the parser interface that has kept text games confined to their niche status. A year and a half ago, he

FRANKENSTEIN app was go full circle back to something that puts choices in front of you, but with the idea we give you no undo, no stats, so everything is risk... It's a pretty extreme take, though, and we're looking at playing with that a little in the next thing we make," Ingold said.

The biggest difference between this approach to interactive story and the "choose your own adventure" books loved by children is that "CYOA books can't remember much, and can't reflow

much they'd enjoy it. That's where the Inklewriter tool comes in.

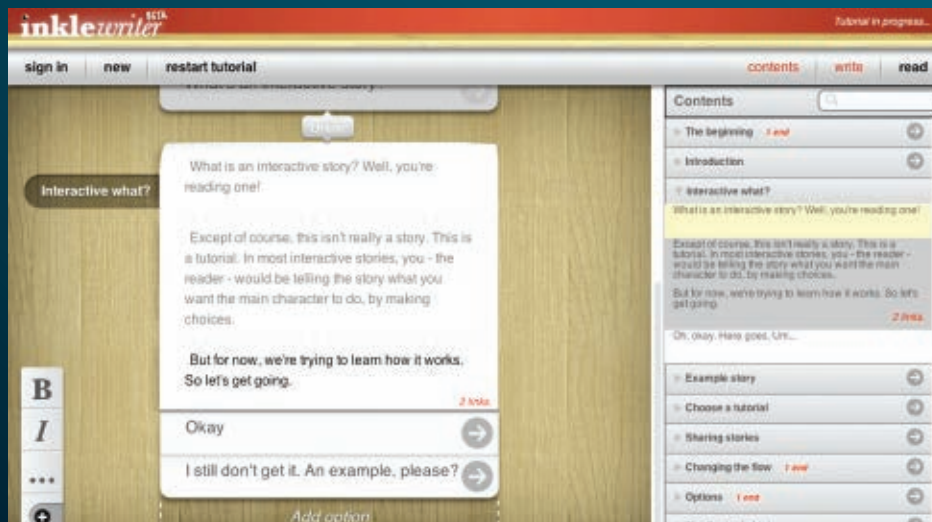
With the hope of encouraging loads of people to experiment with Inklewriter, the studio has just announced its Future Voices Competition, where the 10 best stories made with the tool will be published worldwide as part of an anthology.

"We're hoping we'll get a few great stories out of Inklewriter, and that'll get other writers inspired enough, or annoyed enough, to come along and try to one-up them and do better," Ingold said. "Then if people start talking about what they're writing, then I think we could see some really good stuff."

"One day I would like to come back to the parser thing and really fix it, somehow," he added. "I'd love to make a version that was playable on a tablet or a phone, without typing, and without presenting the user with huge menus of verbs, but that somehow captured that same freedom...but I'm not sure how popular it would be. It might just annoy the purists and confuse the more casual players!"

For now, Ingold's dream is just to see people having fun making easy-to-read interactive stories through Inklewriter. Already he's seen someone make a birthday adventure as a present for a friend, and someone else developed a story about George Osborne's appearance at the Levinson enquiry in the U.K. "We've also been seeing it used by schools for doing creative writing with kids, and that's really nice, too," he said. Its facility for easily managing narrative branches may lend Inklewriter to good use by other kinds of game developers doing story plotting ahead of development. "That's the kind of thing I want it to be—a sort of 'WordPad for interactive fiction,'" Ingold said. "Simple, but useful."

—Leigh Alexander



new tool for making interactive stories. It's free and designed for anyone to use.

"I spent a long time in the full-on parser-based hobbyist niche, and a lot of that was spent doing experiments trying to make things more accessible," Ingold told *Game Developer* sister publication Gamasutra. "You know, less puzzle-y and more story-y, and then trying to do things to make the parser clever."

Interactive text games have long depended on the parser interface, which relies on players typing simple commands that the game can understand. But the more complex games in the genre get, the more likely it is that players,

showed one of his games to a colleague he describes as "really keen" to explore the genre, and the colleague couldn't get into it. "That was when we started thinking, 'Okay, this isn't about the content—it's about the interface,'" he said. "The minute you type something wrong or can't work out how to phrase it, [the narrative momentum] is pretty quickly shattered."

The team aimed to avoid that with FRANKENSTEIN, a new mobile app written by Dave Morris that offers an alternate take on Mary Shelley's classic tale of a doctor creating nightmarish new life. "What we finally did with the

text," Ingold explained. "Our engine has a lot of power geared toward altering the text within paragraphs depending on what you've done and seen, so while it's not auto-generating any prose, it's certainly customizing right down to the level of the individual words. When that works, you get this seamless flow that's a little bit like the writer is there inside the app, like some kind of goblin making up the story on the fly."

That can be an extremely appealing feeling for game designers, and it's also an approach that should allow interactive stories to be accessible to more players who might not even know how

SEVEN-DAY SHOOTER JAM

HOW A GAME JAM TRIED TO CHANGE THE FPS GENRE IN SEVEN DAYS

"FPSes are a horribly oversaturated genre, indies can easily do amazing new stuff. Who's up for it?" tweeted Vlambeer co-founder Jan Willem Nijman in April. What started out as a random thought quickly snowballed into one of the most interesting recent game jams: The 7 Day First Person Shooter Challenge (7dfps.org), which took place in early June. The goal was to create weird and wonderful first-person shooter concepts—a genre that independent developers tend to avoid—in just a week. 7DFPS went on to see hundreds of entries, with the likes of Wolfire Games (LUGARU) and Cryptic Sea (GISH) taking part.

The issue with the FPS space, Nijman said, is that players have no idea of the potential innovation that can occur, and instead choose to throw money at publishers who churn out the same dreary titles over and over again. "Most players



don't know what they want, so we have to give it to them!" he said. "I wish people would look at shooters from the start, where it all started, and work from there instead of iterating on the stuff from two years ago."

Wolfire Games's David Rosen was excited for the chance to experiment. "The 7DFPS challenge seems like a great event for creating this kind of experimental game prototype," he said. "If any large

company divided into teams and participated in it, they would end up with more design ideas than they know what to do with, along with immediate evidence of which ones work and which ones don't."

Wolfire ended up releasing its jam entry, RECEIVER (<http://www.wolfire.com/receiver>), as a paid game after the jam had ended. RECEIVER explores gun-handling mechanics, randomized levels, and unordered storytelling as the player attempts to uncover a variety of secrets in a dangerous building complex.

"I've always found guns to be fascinating in their simplicity, in contrast to their world-changing power, so a key design motif of RECEIVER is a focus on machines that are simple but deadly," Rosen said. "We expressed this by exposing every single component of each machine to

the player. For example, there's a key for every possible function of the gun, from the safety to the slide lock, and players can independently disable every component of the enemy robots."

Cryptic Sea's Alex Austin took a different approach to his 7DFPS entry; SUB ROSA (<http://7dfps.org/?projects=sub-rosa>) has two teams of players try to swap cash for important documents, while a third team, armed with guns and cars, wants to roll in and cause a stir, taking both the documents and cash in the process. "What needs to happen is gamers need to support games that try something new. DAYZ's popularity and increase in ARMA II sales show that players are looking for new experiences. Even though it's a mod for a fairly obtuse game, it's been the best selling game on Steam," Austin said.

—Mike Rose

RETRO FOREVER

EMULATION PACKAGES FOR AMIGA, C64 UPDATED

Classic computing enthusiasts will be glad to hear that longtime Amiga/Commodore software developer Cloanto has updated its C64 Forever and Amiga Forever 2012 "preservation packages" for the Commodore 64 and Amiga computers.

Both packages now support the RP9 format, a special kind of ZIP archive that lets you package multiple disk image files together and open them with the emulators so you can more easily distribute games and demoscene productions in one simple file. Other new features include social networking functionality, faster loading times, and Unicode support, as well as the new inclusion of GEOS, the 8-bit desktop suite, for C64 Forever.

R2 updates are free for existing users, and prices for the two packages range from free for a feature-limited version up to \$50 for special editions that include DVDs with five hours of interviews with "Father of the Amiga" Jay Milner and others. Check it out yourself at amigaforever.com and c64forever.com.

—Patrick Miller



Careers at Audi

Advanced Graphical User Interface Designer (m/f)

We have a clear goal: by 2020, we want to become the most successful premium car manufacturer in the world with our emotional and innovative cars. We would therefore like to strengthen our workforce by recruiting competent new employees.

Stimulating and varied – your working environment:

Vehicle interiors are increasingly characterised by graphical user interfaces. Audi enjoys a leading position when it comes to visual quality and aesthetics. The interfaces designed by Audi Design for series production vehicles and show cars impress with their intuitive and emotional interaction and displays.

Challenging and diverse – your tasks:

As an Advanced Graphical User Interface Designer, your job will be to devise and design the Audi MMI and driver information systems of the future and the applications for mobile devices. Working closely with other interior design departments and with the technical disciplines “Operating Concept” and “Ergonomics”, the Audi Design GUI team designs sophisticatedly progressive interfaces for future vehicle generations.

Technical and personal – your qualifications:

- ▶ Degree in Design
- ▶ Ideally, several years of professional experience in the field of GUI design
- ▶ Design expertise in the field of 2D and 3D visualisation
- ▶ Advanced competence in working with commonly used tools for visualisation, modelling and animation
- ▶ Ability to present complex interaction structures in a self-explanatory and appealing manner
- ▶ Technical understanding
- ▶ Good knowledge of German and English essential
- ▶ Curiosity and fascination beyond the core field of activity, e.g. gaming, films, new media
- ▶ Team spirit and good communication skills

Where and when – your position:

This post is to be filled from 1 September 2012 at AUDI AG in Ingolstadt.

Take advantage of excellent future prospects, experience unforgettable moments:
find out more at www.audi.com/career and www.facebook.com/audikarriere

Apply now:

www.audi.de/meine-bewerbung

Reference code: I-D-4533

Should you have any questions about this vacancy,
please contact Ms Ulrike Krist on telephone number +49 (0)841 893 4877



AI

ARCHITECTURES

🇲🇽 WHAT'S ON THE MENU?

BY DAVE MARK

////////// Asking “What artificial intelligence architecture should I use for my game?” is like asking a waiter “What should I have for dinner?” The answer is always “It depends.”

As a former waiter-turned-AI-consultant, I know the way out of both situations is to ask return questions. “Well, how hungry are you? Are you in a hurry? What are you in the mood for? Steak? Chicken? Allergic to peanuts? Oh... you’re vegan? And you need gluten-free, eh?” Ask the right questions, and eventually the customer will discover what it is he really wants. Likewise, designers need to figure out their goals, technical and expertise limitations, time frame, and necessary degree of authorial control before I can point them toward an AI framework that works for their game.

There are plenty of books and articles out there on AI architectures, but most of them just tell you how the different architectures work, not the pros and cons to each approach—which leads many people to try building an amazing AI with a less-than-ideal tool for the job. In this article, we’ll walk through a high-level overview of the dominant AI approaches out there to help you find the one that’s right for you.

SAME STUFF, DIFFERENT SHAPES

» Selecting an AI architecture is a bit like selecting American Mexican food: It’s a lot of the same stuff (tomato, cheese, beans, lettuce, onions, meat) in different delivery formats (tacos, burritos, tostadas, and so on). But why do we think in terms of the outside form when it is the stuff that is on the inside that is pretty much the point of the order in the first place?

The answer lies in the fact that the outside—the shell or wrapper—is merely a content delivery mechanism that exists mostly to keep those internals together long enough for consumption. In this way, these “delivery mechanisms” compare to AI architectures. They serve to package and deliver



HOW TO CHOOSE
**THE RIGHT
ARTIFICIAL
INTELLIGENCE**
FOR YOUR
GAME!

the tasty behavioral content that we want our players to experience, so when we talk about our AI systems we often speak in terms of the mechanism rather than the content. We are writing a finite state machine, a behavior tree, or a planner, just like we are filling a taco, a burrito, or an enchilada. Once we have decided on that packaging, we can put any (or all!) of the tasty stuff into it that we want to.

THE NO-ARCHITECTURE TOSTADA

» If you're trying to build behavioral content for your game without a proper AI architecture, you're going to end up with something messy—like a tostada. The tostada is, quite literally, about as simple a delivery platform that you can use for Mexican food—



everything just sits on top of it right where you can see it. You can add and remove ingredients with ease, though if you add too much it'll fall off the sides. If you pick it up correctly, everything stays put, but it's not terribly stable; tip it in the slightest, and you

run the risk of sending things tumbling. What's more, as soon as you start biting into it, you run the risk of having the whole thing break in unpredictable ways, at which point your entire pile of content falls apart.

When you're simply adding rules here or there around your code, it'll change the direction of things in a fairly haphazard manner. Like the tostada, you can only get so much content before things become unstable, and every time you take a bite of your content, you never know when the entire platform is going to simply fall apart. For example, perhaps you have just triggered a rule in this event that seems to make perfect sense unless, of course, you were already responding to a rule that was triggered in some other event. If there is no structure to the AI, those rules may not have any "awareness" of each other; each was doing what it was told. The resulting conflicts can be amusing—or even disastrous depending on your frame of mind.

ADDING STRUCTURE WITH THE STATE MACHINE TACO

» Our tostada suffered from not having enough structure to hold the content stable before it started to fall off—or fall apart entirely. However, by just being a little more organized about how we arrange things, we can make sure our content is a lot more self-contained, which lets us hold a lot more and makes it easier to manipulate it. Take your AI tostada and bend the shell, and you'll have a delicious finite-state machine taco.

The finite-state machine (FSM) essentially adds a little bit of structure to a bunch of otherwise disjointed rules maps. The most basic part of a FSM is a state—that is, an AI agent is doing or being something at a given point in time. Theoretically, an agent can be in only one state at a time. (This is only partially correct because more advanced agents can run multiple FSMs in parallel...never mind that for now.)

Finite state machines organize AI agent behavior better because everything the agent needs to know about what it is doing is contained in the code for the state that it is in. The animations it needs to play to act out a certain state, for example, are listed in the body of that state. The other part of the state machine is the logic for what to do next. This may involve switching to another state or simply continuing to stay in the current one.

The transition logic in any given state may be as simple or as complex as you need. You could use a countdown timer that switches the agent to a new state after a certain amount of time, or a random chance for the agent to enter a new state. For example, State A might say that, every time we check, there is a 10% chance of transitioning to State B. We could even elect to make the new state that we will transition to a result of a random selection as well—say a 1/3 chance of State B and a 2/3 chance of State C.

More commonly, state machines employ elaborate trigger mechanisms that involve the game logic and situation. For instance our "guard" state may have the logic "If [the player enters the room] and [is holding the Taco of Power] and [I have the Salsa of Smiting], then attack the player," at which point my state changes from "guard" to "attack." Note the three individual criteria in the statement. We could certainly have a different statement that says, "If [the player enters the room] and [is holding the Taco of Power] and [I DO NOT have the Salsa of Smiting], then flee," which would send the agent out of the "guard" state.

So each state has the code for what to do while in that state and, more notably, when, if, and what to do next. While some of the criteria can access some of the same external checks, in the end each state has its own set of transition logic that is used solely for that state. Unfortunately, this comes with some drawbacks.

First, as the number of states increases, the number of potential transitions increases at an alarming rate: If any given state could potentially transition to any of the other states, the number of transitions increases fairly quickly. Specifically, the number of transitions would be the [number of states] × [(number of states) - 1]. In **Figure 1**, there are four states, each of which can transition to three others for a total of 12 transitions. If we were to add a fifth state, this would increase to 20 transitions. Six states would merit 30, and so on. When you consider that games could potentially have dozens of states transitioning back and forth, you begin to appreciate the complexity.

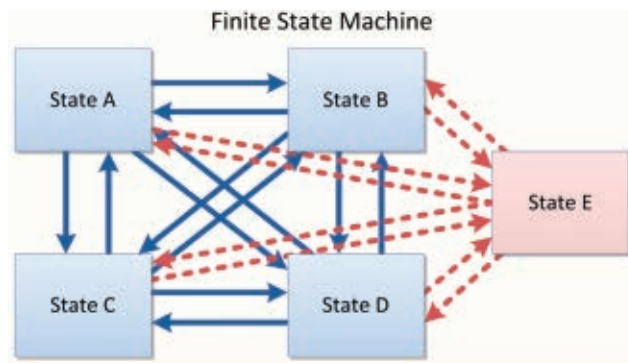


FIGURE 1: AS WE ADD STATES TO A FINITE STATE MACHINE, THE NUMBER OF TRANSITIONS INCREASES RAPIDLY.

You'll feel this complexity when you try to add a new state to the mix: In order to have that state accessible, you have to go and touch every single other state that could potentially transition to it. Looking back at **Figure 1**, if we were to add a State E, we would have to edit states A-D to add the transitions to E. Editing a state's logic invokes the same problem, since you have to remember what other states may be involved and revisit each one.

Additionally, any logic that would be involved in that transition must also be interwoven into the other state-specific logic that may already be there. With the sheer numbers of states in which to put the transition logic and the possible complexity of the integration into each one, we realize that our FSM taco suffers from some of the same fragility of the ad hoc tostada we mentioned earlier. Sure, because of its shape, we can pile a little more on and even handle it a little better. One bite, however, could shatter the shell and drop everything into our lap. And the bigger it gets, the more opportunity for disaster.

Another advantage over the ad hoc tostada is that the agent is only concerned with (and therefore only checks) the transition rules that are in the current state. If you are in a state you know that those rules—and only those rules—are going to be processed. From a processing standpoint, it's far more streamlined than the "here a rule, there a rule, everywhere a rule rule" approach.



THE BEHAVIOR TREE: A SOFT (TACO) APPROACH

» Our state machine taco shell can hold more content than the tostada, but it's still a bit brittle due to its hard shell. Fortunately, we've got a soft taco that can hold the same content without being prone to shattering under pressure: the behavior tree.



At this point, it is useful to point out the difference between an action and a decision. In the FSM above, our agents were in one state at a time—that is, they were “doing something” at any given moment (even if that something was “doing

nothing”). Inside each state was decision logic that told them whether they should change states, and which state to change to. That logic often has very little to do with the state that it is contained in and more to do with what is going on outside the state or even outside the agent itself. For example, if I hear a gunshot, it really doesn't matter what I'm doing at the time—I'm going to flinch, duck for cover, wet myself, or any number of other appropriate responses. Therefore, why would I need to have the decision logic for “react to gunshot” in each and every other state I could have been in at the time?

The behavior tree separates the states from the decision logic. Both still exist in the AI code, but they are not arranged so that the decision logic is in the actual state code. Instead, the decision logic is removed to a stand-alone architecture (see **Figure 2**). This allows it to run by itself (either continuously or as needed) where it selects what state the agent should be in. The state code itself is only responsible for doing things that are specific to that state such as animating, changing values in the world, and so on.

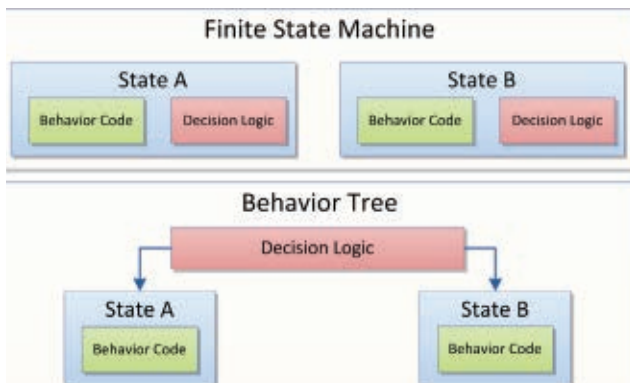


FIGURE 2: IN A BEHAVIOR TREE, THE DECISION LOGIC IS SEPARATE FROM THE ACTUAL STATE CODE.

The main advantage to this is that all the decision logic is in a single place. We can make it as complicated as we need without worrying about how to keep it all synchronized between different states. If we add a new behavior, we add the code to call it in one place rather than having to revisit all of the existing states. If we need to edit the transition logic for a particular behavior, we can edit it in one place rather than many.

Another advantage of behavior trees is that they are a far more formal method of building behaviors. Through a collection of tools, templates, and structures, very expressive behaviors can be written—you can even sequence behaviors together that are meant to go together (see **Figure 3**). This is one of the reasons that behavior trees have become one of the more “go-to” AI architectures in games, including major triple-A titles ranging from HALO 2 and 3 to SPÖRE.

A detailed explanation of what makes behavior trees work, how they are organized, and how the code is written is beyond the scope of this article.

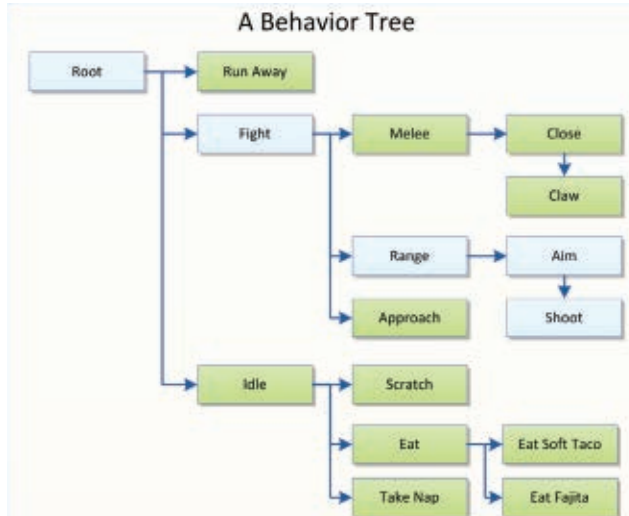


FIGURE 3: A SIMPLE BEHAVIOR TREE. AT THE MOMENT THE AGENT HAS DECIDED TO DO A RANGED ATTACK.

Suffice it to say that they are far less prone to breaking their shell and spilling their contents all over your lap. Because the risk of breaking is far less, and the structure is so much more organized, you can also pack in a lot more behavioral content. For an excellent primer on behavior trees, check out Bjoern Knafila's “Introduction to Behavior Trees” (listed in the Resources section at the end of this article).

A HYBRID TACO—THE HIERARCHICAL FINITE-STATE MACHINE

» A brief note before we leave the land of tacos behind: One of the advantages of the behavior tree—namely the treelike structure—is sometimes applied to the finite-state machine. In the hierarchical finite-state machine (HFSM), there are multiple levels of states (see **Figure 4**). Higher-level states will only be concerned with transitioning to other states on the same level. On the other hand, lower-level states inside the parent state can only transition to each other. This tiered separation of responsibility helps to provide a little structural organization to a flat FSM and helps to keep some of the complexity under control.

If we were to place the HFSM into our Mexican metaphor, it would be similar to one of those nifty hard tacos wrapped in a soft taco shell. There's still only so much you can pile into it before it gets unwieldy, but at least it doesn't tend to shatter and make as big of a mess.



FIGURE 4: IN A HIERARCHICAL FINITE-STATE MACHINE, SOME STATES CONTAIN OTHER RELATED STATES MAKING THE ORGANIZATION MORE MANAGEABLE.

A LA CARTE AI: THE PLANNER FAJITA

» My wife wants to choose exactly what is in her Mexican dish, right down to every single bite. That's why she orders fajitas. While the fajita looks and acts like a soft taco, you build it by taking the tortillas and a few piles of content,

and constructing your own on the spot. You can choose what you want to put in the first fajita and change it up for each subsequent one, depending on what you want in any given moment. The AI equivalent of this is the planner. While the end result of a planner is a state (just like the FSM and behavior tree above), how it gets to that state is significantly different.

Like a behavior tree, the reasoning architecture behind a planner is separate from the code that “does stuff.” A planner compares its situation (the state of the world at the moment) and compares it to a collection of individual atomic actions that it could do, then assembles one or more of these tasks into a sequence (the “plan”) so that its current goal is met.

Unlike other architectures that start at its current state and look forward, a planner actually works backward from its goal (see **Figure 5**). For example, if the goal is “kill player,” a planner might discover that one method of satisfying that goal is to “shoot player.” Of course, this requires having a gun. If the agent doesn’t have a gun, it would have to pick one up. If one is not nearby, it would have to move to one it knows exists. If it doesn’t know where one is, it may have to search for one. If another method of satisfying the “kill player” goal is to throw a Taco of Power at it, and the agent already has one in hand, it would likely elect to take the shorter plan and just hurl said taco. The result of searching backward is a plan that can be executed forward.

The planner diverges from the FSM and BT in that it isn’t specifically hand-authored; agents actually solve situations based on what is available to do and how those available actions can be chained together. One of the benefits of this sort of structure is that it can often come up with solutions to novel situations that the designer or programmer didn’t necessarily account for and handle directly in code.

From an implementation standpoint, a major plus of the planner is that a new action can be dropped into the game and the planner architecture will know how to use it. This speeds up development time markedly. All the author says is, “Here are the potential things you could do...go forth and do things.”

Of course, a drawback of this is that authorial control is diminished. In a FSM or BT, creative “outside the box” solutions were the exception from the predictable, hand-authored systems. In a planner, the scripted, predictable

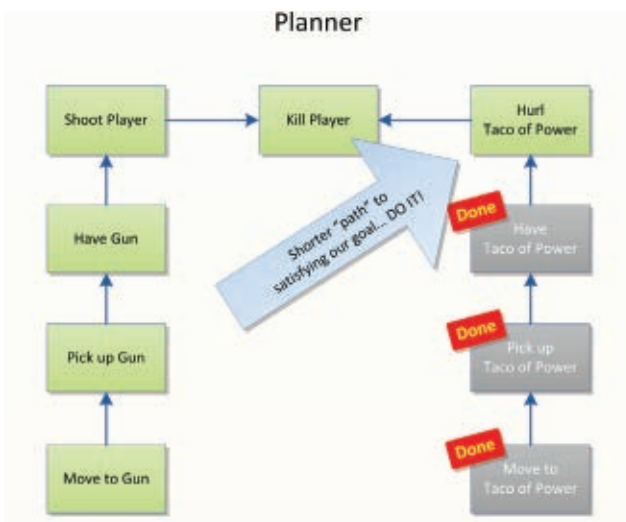
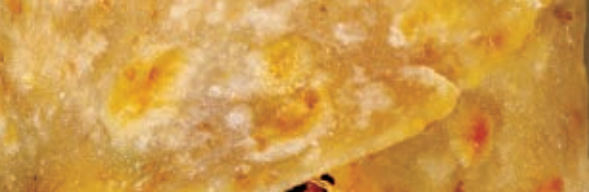


FIGURE 5: THE PLANNER HAS FOUND TWO DIFFERENT METHODS OF ACHIEVING “KILL PLAYER” AND SELECTED THE SHORTER ONE.

moments are the exception; you must specifically override or trick the planning system to say, “No...I really want you to do this exact thing at this moment.”

While planner-based architectures are less common than behavior trees, there are notable titles that used some form of planners. Most famously, Jeff Orkin used them in Monolith’s creepy shooter, F.E.A.R. His variant was referred to as Goal-Oriented Action Planning or “GOAP.” A more recent flavor of planner is the hierarchical task network (or HTN) planner such as was used to great effect in Guerilla’s KILLZONE 2. (Check the Resources section for more on GOAP and HTN.)





PUTTING IT ALL IN A BOWL—A UTILITY-BASED SALAD

» Another architecture that is less structured than the FSM or behavior tree is called the “utility-based” method. Much like the planner, a utility-based system doesn’t have a predetermined arrangement of what to do when.



Instead, potential actions are considered by weighing a variety of factors (what is good and bad about this?) and selecting the most appropriate thing to do. Like the planner, a utility-based approach lets the AI choose what’s best at the time.

Instead of assembling a plan like the fajita-style planner, the utility-based system simply selects the next bite. This is why it is more comparable to a taco salad in a huge bowl. All the ingredients are in the mix and available at all times. You simply select what it is that would like to poke at and eat. Do you want that piece of chicken in there? A tomato, perhaps? An olive? A big wad of lettuce? You can select it based on what you have a taste for or what is most accessible at the moment.

The AI system in THE SIMS is an excellent example of a utility-based approach—in fact, the considerations are largely shown in the interface itself. The progression of AI architectures throughout THE SIMS franchise is well documented, and I recommend reading up on it. Essentially, each potential action in the game is scored based on a combination of an agent’s current needs and the ability of that action or item to satisfy that need. The agent then constructs a weighted sum of the considerations to determine which action is “the best” at that moment. The action with the highest score wins (see Figure 6).

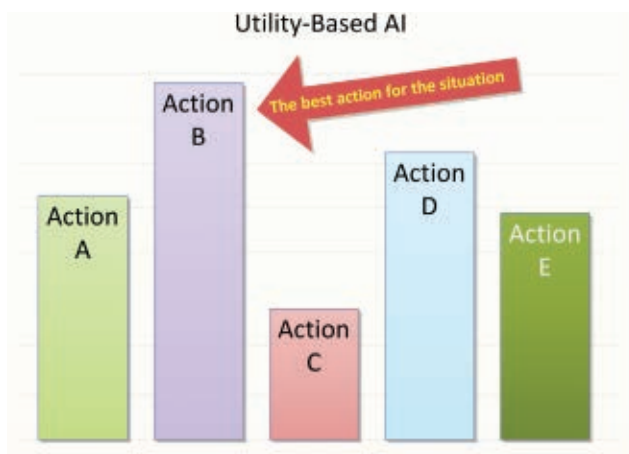


FIGURE 6: A UTILITY-BASED SYSTEM RATES ALL THE POTENTIAL ACTIONS ON A VARIETY OF CRITERIA AND SELECTS THE BEST.

While utility-based systems can be used in many types of games, they are more appropriate in situations where there are a large number of potentially competing actions the AI can take and no obvious “right answer.” In those times, the mathematical approach that utility-based systems employ is necessary to ferret out the ideal action. Aside from THE SIMS, utility-based systems are appropriate in role-playing games, real-time strategy games, and simulations.

Like behavior trees and planners, the utility-based AI code is a reasoner. Once an action is decided upon, the agent still must transition to a state. The utility system is simply selecting what state to go to next, and the reasoning code is all in a single place. This makes building, editing, tuning, and tweaking the system much more compartmentalized. Also, like a planner, adding actions

to the system is fairly straightforward; add the action with the appropriate weights, and the AI will automatically take it into account and use it in relevant situations. This is one of the reasons that games such as THE SIMS were as expandable as they were—the agents simply included any new object into their decision system without any changes to the underlying code.

On the other hand, one drawback of a utility system is that there isn’t always a good way to intuit what will happen in a given situation. With a behavior tree, it’s easy to find the branches and nodes that would be active in a particular situation, but since a utility system is inherently more fuzzy than binary, determining how the actions stack up is often more opaque. That’s not to say a utility-based AI is not controllable or configurable—quite to the contrary. The difference is that rather than telling the system exactly what to do in a situation, the system is providing suggestions as to what might be a good idea. In that respect, a utility system shares some of the adaptable aspects of planners—the AI simply looks at its available options and then decides what is most appropriate.

For more reading on utility-based systems, please check out my book, Behavioral Mathematics for Game AI, as well as my lectures with Kevin Dill on the GDC Vault, available for free (see Resources), titled “Improving AI Decision Modeling through Utility Theory” and “Embracing the Dark Art of Mathematical Modeling.”

WRAP IT UP WITH A NEURAL NETWORK BURRITO

» The last entry in my metaphorical cornucopia is the neural network burrito. In the other examples, all the content was open and easily inspected. In the case of the fajita, you (the AI agent) were able to assemble what you wanted in each iteration. In the taco salad, the hard and soft tacos, and even the



tostada you could add cheese or tomatoes as you liked, and even if you didn’t change the content of your dish, you could still see what you were about to eat before you took a bite.

The burrito is different in this respect: The details are hidden. It is a riddle, wrapped in a mystery, inside a soft flour shell. While the burrito (and for that matter, the neural network) is extremely flexible, you have absolutely no idea what is inside or what you are going to get in the next bite. Don’t like sour cream? Olives? Tough. If it’s in there, you won’t know until you take that bite. At that point, it is too late. There is no editing without completely unwrapping the package and starting from scratch.

This is the caveat empor of the neural network-based AI solution. As a type of “learning” AI, neural nets need to be trained with test or live performance data. At some point you have to wrap up the training and say, “This is what I have.” If a designer wanders in, looks over your shoulder and says, “It looks pretty cool, but in Situation A I would like it to do Action B a little a little more often,” there’s really nothing you can do to change it. You’ve already closed your burrito up, and all you can do is try to retrain the neural network and hope for the best.

So while the neural network offers some advantages in being able to pile a lot of things into a huge concoction of possibilities, you don’t have much designer control at the end of the process. Unfortunately, this tends to disqualify NNs and other machine-learning solutions from consideration in the game AI environment where that level of control is not only valuable but often a requirement. That said, there have been a few successful implementations of NNs in games—for example, Michael Robbins used NNs to improve the tactical AI of SUPREME COMMANDER 2 from Gas Powered Games (see Resources).

In the case of SUPREME COMMANDER 2 and other similar implementations, the AI was trained in the studio until it was acting in a reasonable way and then that data was inserted into the game and shipped. However,

ARCHITECTURE	PROS	CONS
AD-HOC RULES	> minimal set-up	> gets unwieldy past the most basic behaviors
FINITE STATE MACHINE (FSM)	> easy to understand, build	> transition between states get hard to manage with more behaviors
HIERARCHICAL FSM	> hierarchy helps cluster behaviors > easy to understand, build	> transitions still can get difficult to manage
BEHAVIOR TREE (BT)	> separates decision logic from state code > easy to understand, build, edit	> hard-coded priorities of behaviors
PLANNER	> ai "discovers" solutions on the fly > handles unique situations better > easily accommodates new action	> some loss of designer control > "re-planning" can be processor-intensive
UTILITY-BASED SYSTEM	> ai constantly weighs all actions > handles unique situations gracefully > allows for variation in behavior	> some loss of designer control > harder to design, edit, and tune
NEURAL NETWORK	> able to "learn" how to play > can be set up relatively quickly	> complete loss of designer control > nearly impossible to edit or tune

FIGURE 7: THE TYPE OF ARCHITECTURE YOU SELECT NEEDS TO BE BASED ON YOUR NEEDS.

one additional use for neural nets is to allow the game to change, "learn," and adapt to the player after it has been shipped. This is a somewhat controversial practice that sounds a lot cooler than it often turns out to be. Again, a fair treatment of the hows and whys of this is beyond the scope of this article and is well documented elsewhere. Just remember that even uttering the phrase "an AI that learns from the player!" is a siren's song. And there's some pretty big ol' rocks waiting for you!

BROWSING THE BUFFET

» Now we've covered a variety of architectures [and their corresponding Mexican delights]. This has by no means been an exhaustive treatment of AI architectures, but should identify their major differences. Now let's see if we can find the right AI architecture for your game. Let's look at the table in **Figure 7** to recap.

You can certainly just throw the occasional rule into your code that controls behavior—but that's not really an "architecture." By organizing your AI into logical chunks, you can create a finite-state machine, which is relatively easy to construct and easy for nonprogrammers to understand. FSMs are good for simple agents with a limited number of behaviors, but they get unwieldy as the number of states increases. By organizing the states into the logical tiers of a hierarchical finite state machine (HFSM), you can mitigate some of this complexity.

resources

"Introduction to Behavior Trees:" Bjoern Knafle: www.altdevblogaday.com/2011/02/24/introduction-to-behavior-trees

"Goal-Oriented Action Planning:" Jeff Orkin: <http://web.media.mit.edu/~ffjorkin/goap.html>

"HTN planning in Killzone 2:" <http://aigamedev.com/open/coverage/htn-planning-discussion>

"Improving AI Decision Modeling through Utility Theory:" <http://gdcvault.com/play/1012410/Improving-AI-Decision-Modeling-Through> and "Embracing the Dark Art of Mathematical Modeling:" <http://gdcvault.com/play/1015683/Embracing-the-Dark-Art-of> and "Off the Beaten Path: Non-Traditional Uses of AI:" <http://gdcvault.com/play/1015667/Off-the-Beaten-Path-Non>

If you need something flexible, you can remove the reasoning code from the states themselves and instead organize behaviors into logically similar branches—a behavior tree. BTs are also fairly easy for designers and other nonprogrammers to understand, but their main advantage is that they scale very well without needing a lot of extra programming. Once the BT structure is in place, you can easily add new branches and nodes. That said, even the most robust BT implementation is still a form of hand-authored scripting—"when X, do Y."

Like the behavior tree, a planner allows for very extensible creation. However, where the BT is more hand-authored by designers, a planner simply "solves" situations using whatever it feels is best for the situation. This can be powerful, but also leads to a very scary lack of control for designers.

Similarly, utility-based systems depart from the specific script approach and allow the characters to decide freely what to do, which might unsettle some designers. They are incredibly expandable to large numbers of complex factors and possible decisions, but they are slightly more difficult to intuit at times unless you're capable of building tools to aid that process.

The ultimate hands-off black box is the neural network. Even the programmers don't know what's going on inside their little neurons at times. The good news is that they can often be trained to "do what human players would do." That aspect itself holds a lot of appeal in some genres. They are also a little easier to build since you only have to construct the training mechanism and then...well...train it.

There is no "one size fits all" solution to AI architectures. (You also don't have to limit yourself to a single architecture in a game.) Depending on your needs, your team, and your prior experience, any of the above may be the right way for you to organize your AI. As with any technical decision, the secret is to research what you can, even try a few things out, and decide what you like. If I am your waiter AI consultant, I can help you out, but the decision is ultimately going to be what you have a hankerin' for.

Now let's eat! 🍽️

DAVE MARK is the president and lead designer of Intrinsic Algorithm, an independent game development studio and AI consulting company in Omaha, Nebraska. He is the author of the book "Behavioral Mathematics for Game AI" and a GDC AI Summit advisor.

The author would like to dedicate this article to all the GDC, Gamasutra, and Game Developer staff who are still lamenting the departure of Maya, the beloved Mexican restaurant that used to be located in their building. I mourn with you, my friends.



WHY INDEPENDENT DEVELOPERS TURN TO UDK

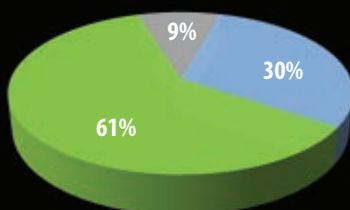
There's never been a better time to check out the Unreal Development Kit (UDK), the free edition of Unreal Engine 3 (UE3), available at www.udk.com.

It's always been free to use until you're ready to deploy a commercial product, and even after the one-time \$99 studio fee, Epic doesn't take a royalty until you pocket \$50,000. We also regularly update UDK and never charge for upgrades.

Here is one example of how the commercial UDK back-end royalty structure works. Say you sell 15,000 copies of a \$4.99 (USD) app, and your digital distribution platform takes 30 percent of total retail sales to \$74,580, leaving you with \$52,395. Epic takes 25 percent of the net amount over \$50,000, or \$598.75. That means you keep \$51,796.25 or 69 percent of the total retail sales. Epic's total cut at this point is \$697.75 or 1 percent of gross revenue.

In a second example (see chart), sales double and 30,000 copies at \$4.99 generates \$149,700. You have \$104,790 after digital distribution fees. At this point, 61 percent of retail sales or \$90,933.50 stays with you, the developer, and the total UDK investment comes to \$13,796.50 or 9 percent of net revenue.

% Breakdown on Total Sales of \$149,700 (USD)



■ Gross Revenue ■ UDK Investment ■ Digital Distribution Fee

UDK provides access to the full UE3 feature set, including the Unreal Editor and its robust suite of tools, such as the Unreal Kismet visual scripting system, Unreal Matinee cinematic toolset, Unreal Cascade particle effects, Unreal Landscape terrain editor and Unreal Lightmass global illumination.

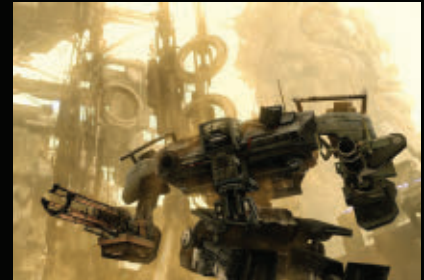
Also included are UE3's navigation mesh AI system, the Unreal skeletal animation system and other tools that would normally cost money to license or require significant internal resources to build.

UDK also ships with hundreds of thousands of dollars' worth of industry-leading middleware technologies, including the full Autodesk Scaleform GFX user interface software package, NVIDIA PhysX and APEX support, SpeedTree foliage editing, FaceFX facial animation, Bink Video, RealD stereo 3D features and more. All of this is available in the base licensing fee. Epic absorbs technology integration costs to ensure developers only gain from our work with great partners.

Epic doesn't take a royalty until you pocket \$50,000. We also regularly update UDK and never charge for upgrades.

UDK scales from lightweight mobile experiences to high-end PC gaming. After Epic demonstrated its next-gen "Samaritan" demo last year, UDK began shipping with DirectX 11 support, so anyone can use UE3 for high-end rendering, tessellation and beautiful post-processing effects.

Every day, developers release UDK projects without needing full source UE3 access. Some studios, such as *Hawken* developer Adhesive Games and *Dungeon Defenders* creator Trendy Entertainment, start out using UDK and then transition to a more traditional UE3 deal.



"By using UDK we were able to not only quickly create a prototype for *Hawken*, but also implement the majority of our gameplay and other game features," said Jon Kreuzer, technical director at Adhesive Games. "When the time was right we were able to easily transition to the full Unreal Engine license. Epic has been very helpful to us in our quest to achieve our creative vision for *Hawken*."

If your needs require full source code access, contact us at licensing@epicgames.com. We'll help you get started and find business terms that are right for your team.

UPCOMING EPIC ATTENDED EVENTS

GDC Europe
Cologne, Germany
August 13-15, 2012

gamescom
Cologne, Germany
August 15-19, 2012

MIGS
Montreal, Canada
November 1-2, 2012



Please email licensing@epicgames.com for appointments



The best ideas evolve.

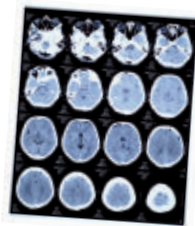
Great ideas don't just happen.
They evolve. Your own development
teams think and work fast.

Don't miss a breakthrough.
Version *everything* with Perforce.

Software and firmware. Digital assets
and games. Websites and documents.
More than 5,500 organizations and
400,000 users trust Perforce for
enterprise version management.

**Try it now. Download the free
20-user, non-expiring Perforce Server
from perforce.com/try20**

Or request an evaluation license
for any number of users.



beyond the heat map

understand your
players' behavior
with spatial
game analytics

BY ANDERS DRACHEN

Designers love their heat maps, and with good reason, since they present clear and intuitive feedback about players' spatial behavior. But heat maps are simply the tip of a very deep iceberg that is spatial game analytics—there's a lot more to offer a designer who wants to understand players better.

what is spatial game analytics?

.....

In recent years the term “analytics” has entered into the broad vocabulary of game development, bringing with it a shiny, sparkly new source of data on players and systems: telemetry. Telemetry refers to behavioral logs of players or systems that can offer incredible granularity and an unobtrusive way of tracking what people are actually doing when playing games.



Game analytics, the analysis of games and gameplay, has been around for a while. Strictly speaking, everything related to user testing and research is analytics, but we’re interested in the analysis of telemetry data, similar to what web analytics does for web sites. In most respects, game analytics is in its infancy; most current work doesn’t dig much deeper than key performance indicators (KPI) like average revenue per user (ARPU) calculations. But looking at the influence of analytics in productivity software development, geographical information systems, and web analytics, game analytics has far greater potential than calculating KPIs.

Spatial game analytics integrates the spatial dimension of player behavior to give you information that comes from the space that players experience games in (namely, the space inside the game itself), which gives you clear links between the game’s design and corresponding player behavior. This is relevant everywhere; even the simplest word game contains movement mechanics, and thus the potential for spatial analytics. You might know that your players took longer than expected to complete a game level, but with actual spatial information from your players, you can pinpoint exactly where your players get held up. In short, it is all about where things happen (or stubbornly refuse to happen!).

what, where, and when

.....

First, let’s explain the basic properties of spatial data.

For any given game, most of the variables you can log about a play session can be attached to spatial and temporal information one way or another. Every time a player throws a punch, fires a gun, runs for cover, or starts a quest, she’ll be doing so from a particular location in 2D or 3D space.

Whenever a player does something (or something happens to that player), we can log three types of

information: what is happening (and to whom it’s happening to), where it’s happening, and when it’s happening. We can refer to these three components of player-derived telemetry data as Event, Space, and Time information.

Of course, how we record spatial information depends in part on the nature of the game; in some games it may not make sense to use geographic coordinates (X, Y, Z), but rather use vector-based logging (with games featuring point-and-drag operations, for example), or even just registering the zone or area a player is in. For some mobile games, logging the geographic coordinates of player positions in the real world can be useful. In any

case, these are all examples of spatial information that are attached to specific player behaviors.

Spatial information generally comes in three types: points (coordinates), lines, and areas (polygons). Point-based data sets are typically defined using coordinate sets (X,Y, and possibly Z); the location of a player when shooting and killing an opponent in TEAM FORTRESS 2, for example. Lines are spanned by multiple points, but like points have no areal extent, unless they encompass an area and thus become polygons. All these types of spatial information come with associated features—a set of coordinates is associated with a feature class providing the player name. Spatial

data is stored in spatial databases, which are essentially similar to any other game asset database that includes spatial information.

Lines can integrate information about direction. Take, for example, the line describing the path a player takes through a level in GEARS OF WAR 3, composed of many small lines connecting the position of the player, which are logged once per second. Area data has a spatial distribution, such as a building or area with a particular type of vegetation. Note how the spatial measures never stand alone—we measure the spatial component in relation to some meaningful variable: the path of a particular player,



the position of players using particular weapons, the areas where specific environmental situations are in effect, and so on. In other words, Space is always measured on conjunction with Event (and usually also Time); we log positional information in conjunction with sets of actions performed by the player, and a time stamp.

Note that telemetry data logging does not need to be triggered by a player doing something. There is equal value in tracking and logging Event, Space, and Time information from the computer-controlled agents of the environments and virtual objects. For example, logging the spatial behavior of AI-controlled enemies lets you evaluate whether their pathing routines are performing as expected or if any bugs are occurring (dynamic object tracking). Currently, the majority of spatial analytics performed in games is done using player-derived telemetry data, but it is important to note that play experience is shaped by the interaction between player and game—the system side should also be analyzed. We can refer to these two different sources of data as player and system telemetry.

spatial analysis tools

Once you have your telemetry data, you can use many different techniques to analyze it, depending on what you want to learn. If you don't want to build your own in-house analysis tool, you can adopt an existing toolset that fits your needs, even down to small packages focusing on specific analyses to systems that handle any kind of spatial data or task, known as geographic information systems (GIS).

A GIS is conceptually similar to the data management systems already used in games, which control objects and entities with specific attributes and behaviors inside the game environment. The difference is that a GIS is specifically designed for the management and analysis of spatial and spatio-temporal data.

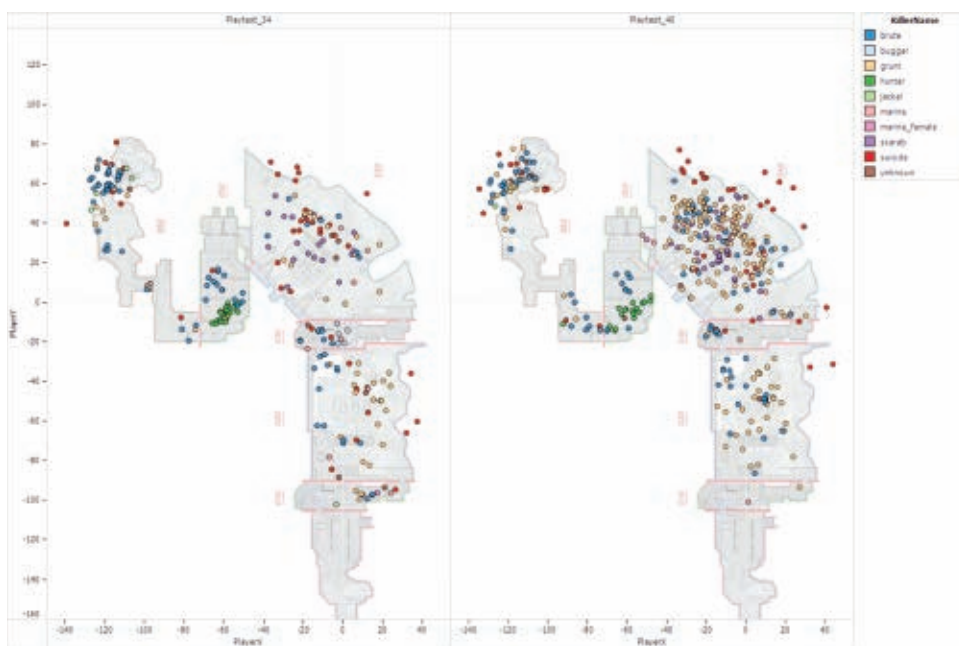
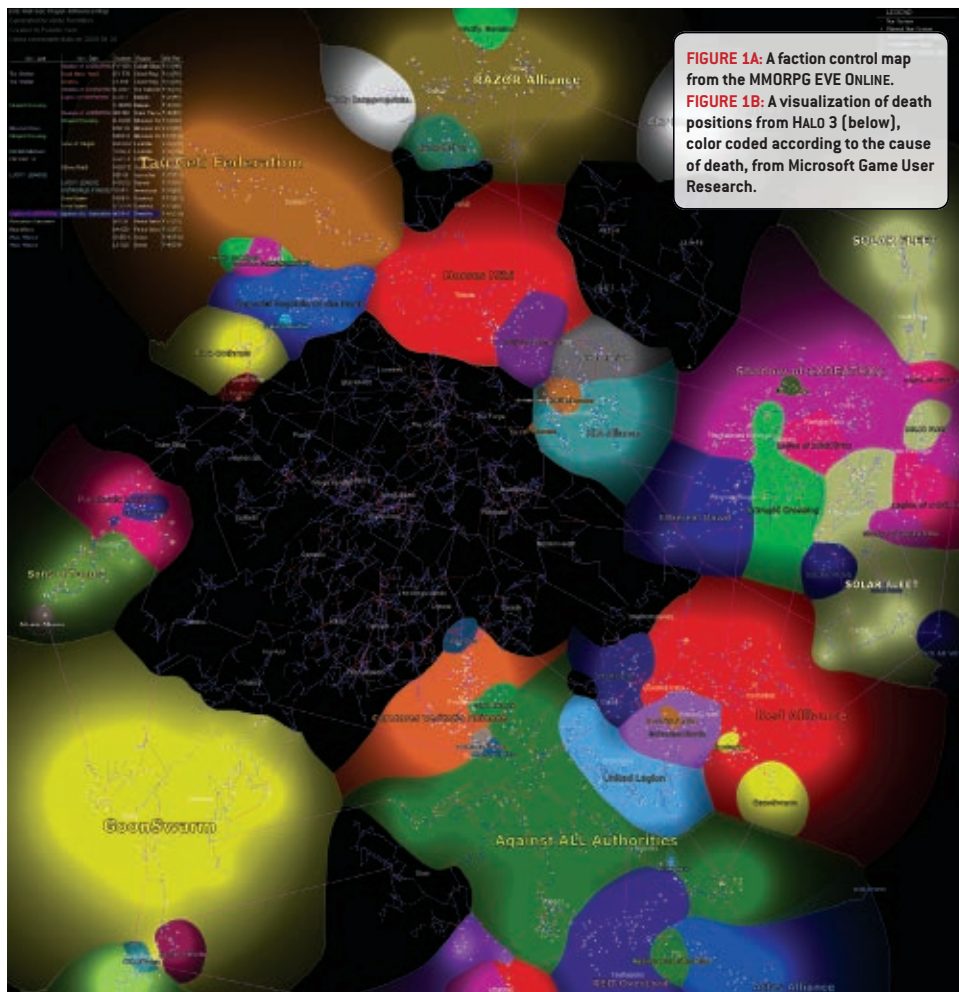
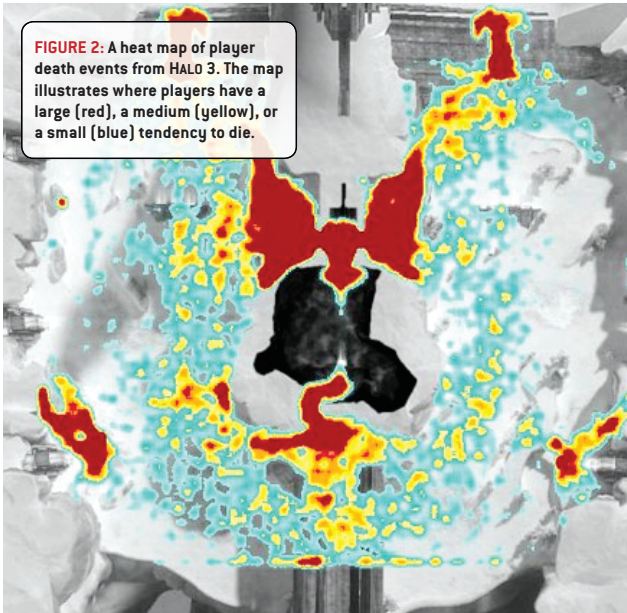


FIGURE 2: A heat map of player death events from HALO 3. The map illustrates where players have a large (red), a medium (yellow), or a small (blue) tendency to die.



In a GIS, map features are linked with attribute information. For example, a level map is linked with locations of quest providers, different types of environments, player trajectories, or other spatially distributed features. When you map spatial game metrics onto maps that themselves can contain detailed feature information, you can be much more flexible with your analyses; for example, you can calculate the number of kills occurring “inside” a specific type of environment vs. “outside” a type of environment—as well as specific numbers for each environment type.

Recently a minor swarm of start-ups have begun offering technologies for the tracking, logging, and storage of game telemetry data. They are mainly

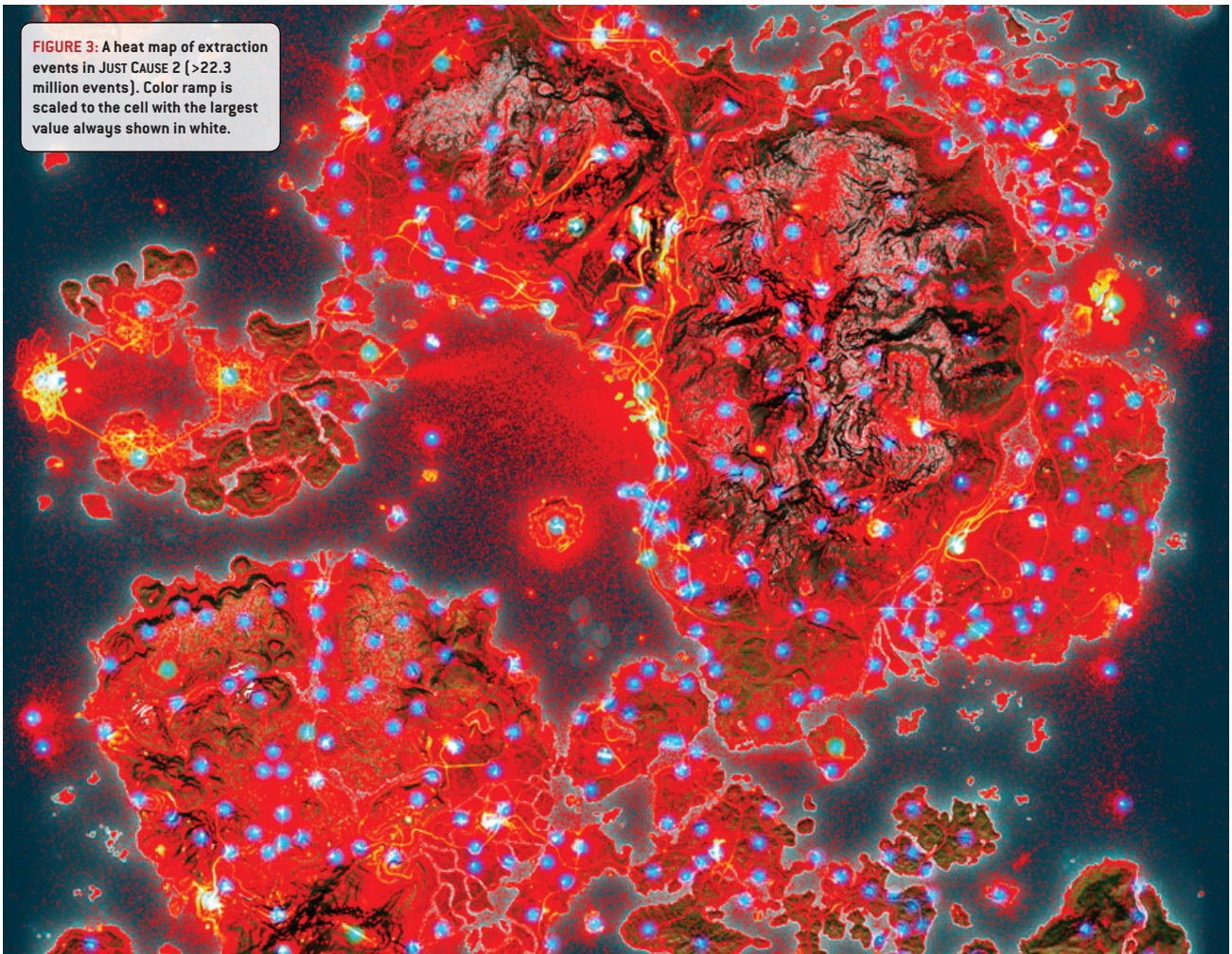
focused on nonspatial metrics, so we still need to look outside game-specific solutions, such as the open-source tool Quantum GIS (www.qgis.org).

visualizing game metrics

.....

Game metrics can be visualized without there being any operations on the data. These visualizations are among the most common, and are relatively easy to generate and interpret. Almost all metrics visualizations to date have been generated based on point-based data, with line and area features more or less unused, but in principle these types of spatial features are equally applicable. You can use these visualizations internally to evaluate your game and level

FIGURE 3: A heat map of extraction events in JUST CAUSE 2 (>22.3 million events). Color ramp is scaled to the cell with the largest value always shown in white.



design: For example, you can see if any mobs wander outside the area they are supposed to stay within, or where players pick up items, and so forth (see **Figure 1A** and **Figure 1B** for examples).

more than just a heat map

Heat maps began as 2D displays of the values in a data matrix, but have since been adapted to a variety of contexts, from heat mapping of eye gaze on web sites to mapping of environmental factors, among other uses. A common way to generate a heat map is to divide the game area under scrutiny into a grid of cells (bins), sum up the number of events that occur within the area covered by each cell, and add a color ramp (green to red, for example) to make the data easy to interpret. Other approaches employ density kernel functions or interpolation in order to attempt simple prediction in areas with no registered events.

Heat maps are commonly used in games to aggregate and visualize death events (see **Figure 2**). They show designers exactly where players and bots die. Add information about the different causes of death to a heat map, and you've made your heat map more useful, because you can evaluate the impact of different enemies or weapons in different regions of a space.

❗❗ Currently, the majority of spatial analytics performed in games is done using player-derived telemetry data, but it is important to note that play experience is shaped by the interaction between player and game—the system side should also be analyzed. ❗❗

Of course, you can use heat maps for more than just mapping death events (or any other point-based data, for that matter). You could just as easily use player trajectory data (line data) to show how often players choose a particular movement path. Similarly, area data (polygon data) can be aggregated. Any spatial variable

can form the basis for heat maps; **Figure 3** uses extraction events from JUST CAUSE 2.

Most game heat maps I've seen have been in 2D using point data [X,Y]; but there are applications for generating 3D heat maps [X,Y,Z], which let you more accurately interpret your level design, especially when you're dealing with data from multiple Z-axis levels (for example, data points coming from players inside a building with two floors). In that case data points are layered on top of each other.

Combining heat map visualizations with the temporal dimension of telemetry data adds a dynamic quality and allows for a better understanding of game flow. Essentially, heat maps combined with temporal information allow designers to follow how the mapped events occur over time; for example, you can track what players do in the first 30 seconds on a map, instead of simply logging an aggregate sum of their activities.

analyzing multiple spatial variables

Now that we've addressed the basic concepts behind heat maps and spatial analytics, let's take a look at how you can combine and visualize multiple variables to see how those variables interact. For example, you can combine heat maps with trajectory data from

player movement or projectile paths to give you a highly detailed analysis of the dynamics of a playfield. You could also take heat maps from two different builds of a specific level to easily determine what kind of impact the design changes had on the gameplay. Or you could take two heat maps generated

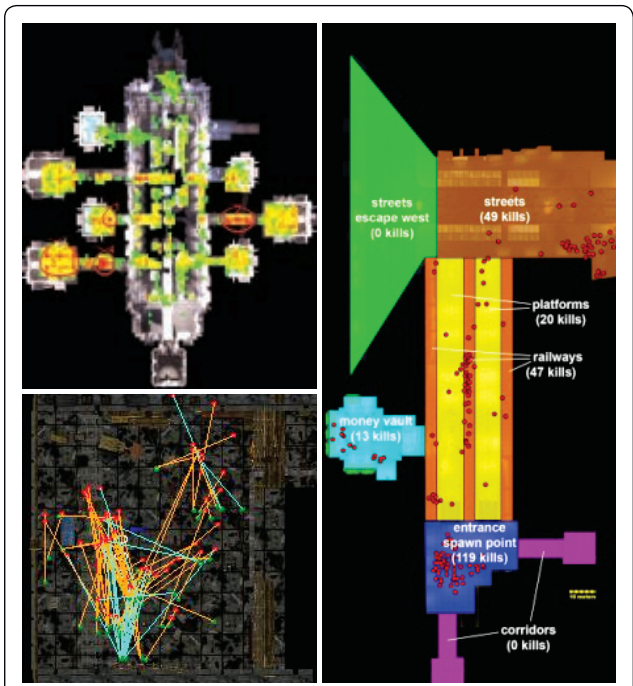


FIGURE 4: Various examples of visualization of multiple features: (top left) Analysis of the density of causes of death in the Valaskjalf map unit from TOMB RAIDER: UNDERWORLD, used for evaluating whether areas are too challenging (the higher the number of threats, the more challenging the gameplay); (right) Overlay analysis of the Subway map from FRAGILE ALLIANCE, combining a point-based layer (locations of player death) and polygon layer (describing key areas of the Subway level) from a series of playtests, used for evaluating progress of the two player teams; (bottom left) A combined visualization of the locations of players when they kill an opponent, the location of the opponent, range of the shot and the weapon used (indicated by the line color), from a playtest of KANE & LYNCH 2.

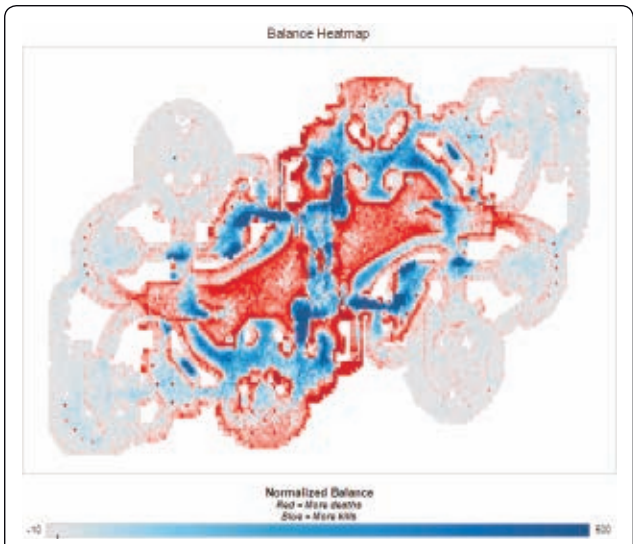


FIGURE 5: A balance heatmap from the Molten map of the game TRANSFORMERS: WAR FOR CYBERTRON, generated using an overlay function, where a death heat map is subtracted from a kill heat map. Areas with negative values (red) indicate dangerous areas; areas with positive values (blue) indicate areas that are safer. An interesting conclusion is that wall areas appear to be dangerous, possibly due to restricted movement.



FIGURE 6: (Top) 980 player trajectories from the Arathi Basin battleground of World of Warcraft. Red circles indicate strategic points. (Middle) Map of the Arathi Basin battleground showing main pathways. (Bottom) Placement of observers in Arathi Basin.

middleware for game analytics



There exist a variety of middleware solutions for logging and transforming game telemetry, mostly formed by relatively new start-ups such as Game Analytics (Disclaimer: Game Analytics is my employer), Playtomic, and Turiya Media. Some business intelligence providers like Plateau are also moving into this area, as are engine developers. The specifics of the solutions offered by each company vary, but they generally provide a system for adding hooks into the game code, which ensures that relevant telemetry data are transmitted to a server-side database, which communicates with a web-based front end through which the user interacts with the data. Telemetry data is then transformed into game metrics; generating an average completion time metric by combining session time and player ID data, for example. When it comes to spatial analytics, the tools offered by these companies remain in their infancy compared to the commercially available packages available in other sectors (geographic information systems). In-house systems for game analytics appear to exist in most major publishers and developers, such as the TRUE system at Microsoft Game User Research and Skynet at BioWare. In the public research a few tools have also been developed: Check out Ben Medler's Data Cracker, Andre Gagne's Pathway and Pedersen/Canossa's G-player.

from different variables, such as kills and deaths, to see whether players kill and die in the same spots, or if there are areas in which players more commonly kill and die—from, say, a sniping alcove that provides a player with a convenient kill zone. (See **Figure 4.**)

In order to map multiple variables, you'll need to perform a spatial operation called overlapping, where you take two or more maps or layers registered to a common coordinate system and superimpose them on top of each other in order to create a composite map. This way, you can visualize and analyze the relationship between variables occupying the same space; for example, in a given map of a game level, you can combine player trajectories logged for that level and death events (see **Figure 5**). The overlay function can be based on simple operations such as summing, subtraction, multiplication, averages, or co-occurrences.

As mentioned above, spatial data can be points, lines, or polygons (areas). In turn, these can be displayed in two basic formats: raster or vector models. Raster models are grids, and each grid cell contains information about one or more variables (usually one); vector models are composed of points and lines. Heat maps are based on raster models. Raster and vector models differ significantly in the way overlay operations are performed, and generally overlay operations are performed most effectively in raster-based models. A GIS is usually able to convert raster models to vector models and vice versa, so you can convert vector models to raster models, perform an overlay operation, and then convert the result back into a vector model.

You can use overlay operations to perform a variety of analyses. For example, flow maps are

visualizations of direction that you can use to show the main vectors of player navigation through a playfield (whether 2D or 3D). Take note when you're projecting 3D data to a 2D plane, however: Most 3D games have variations in the Z-plane, leading to different opportunities for players (such as elevated positions). Levels may even have multiple planes on top of each other (floors in a building, for example). Projecting multiple layers of data on top of each other provides bias, and should ideally be kept separate.

Similarly, some visualized patterns may not be clear unless you take the 3D nature of the playfield into account. You can solve this with a 3D visualization of the data, which is still rare in games, but could be a powerful tool for designers for evaluating player behavior in the actual space in which players experience your games—namely, the game engine itself—once you add the temporal player data. Georg Zoeller from BioWare presented some great examples at GDC last year (you can download the slides from <http://gdmag.com/resources/code.php>) In summary, multivariate synthesis and analysis provide a good starting point for understanding how people play a map—and to a certain degree, why they play it the way they do, giving you one more source for design feedback.

off the deep end

.....
All gameplay occurs over time, and including time in game analytics allows us to work with much more complex events. For the purposes of this article, we'll stick to introducing trajectory analysis.

A trajectory is a description of an object's movement in space over a specific period of time, such as a player's navigation through a game level. Along the way, various events occur—fights, item pick-ups, item uses, and so on. Analyzing trajectories is currently used to locate illegal bot programs in online games, examine group behavior, study player tactics, and more (see **Figure 6**).

Trajectories are essentially approximations. There are various ways to record spatial movement; as a series of point positions, for example, ordered according to time played, with lines drawn to connect them. Depending on the frequency of waypoint logging, it can be a good idea to measure the relative speed of the player together with the position, in order to estimate which sections a player is running through and which sections said player is walking through. You could also describe trajectories in terms of the change in relative movement, movement stored as a sequence of commands such as "go straight," "turn right," and "stop." This form of trajectory pattern logging is currently used in research to distinguish between bots and players. Trajectories are among the more bandwidth-heavy player behavior variables, so it's worth testing to see what kind of data you can filter out to avoid any issues related to transferring and storing that telemetry data.

One of the important forms of trajectory analysis in games is clustering: classifying data based on the degree of similarity and dissimilarity. Clustering can be based on just trajectories, or trajectories and event data (where players use specific abilities, pick up items, and interact with other players). Combining trajectories with event data is useful because paths alone do not necessarily tell us why players navigated in a particular way, or whether a group of players traveled together—something important for any game involving team strategy, such as first-person shooters and MMORPGs. Usually cooperative behavior is limited to a specific period of time, which is why spatiotemporal data analysis is so useful for these game elements.

By analyzing player trajectories, we can isolate certain patterns that are indicative of specific trajectory behaviors—tracks, flocks, and leadership patterns, for example. A track describes a player having a constant movement in a given spatial range; a set of players moving on a straight line within

further reading

1. Thompson, C. "Halo 3: How Microsoft Labs Invented a New Science of Play." URL: http://www.wired.com/gaming/virtualworlds/magazine/15-09/ff_halo
2. Drachen, A.; Canossa, A. "Evaluating Motion: Spatial User Behaviour in Virtual Environments." URL: http://andersdrachen.files.wordpress.com/2011/01/05_drachen_ijart.pdf
3. Hoobler, N., Humphreys, G., and Agrawala, M. "Visualizing Competitive Behaviors in Multi-User Virtual Environments." URL: <http://www.cs.virginia.edu/ffgfx/pubs/lithium/>
4. Demers, N. Fundamentals of Geographical Information Systems. URL: <http://www.amazon.com/Fundamentals-Geographical-Information-Systems-Michael/dp/0470129069>
5. Houghton, S. "Balance and Flow Maps." URL: <http://altdevblogaday.com/2011/06/01/balance-and-flow-maps-2/>
6. Kennerly, K. "Better Game Design Through Data Mining." URL: http://www.gamasutra.com/view/feature/2816/better_game_design_through_data.php

sources

- Figure 1: EVE ONLINE diagram from http://wiki.eveonline.com/wiki/EN/images/d/d0/Territorial_maps.png, HALO 3 diagram from Ramon Romero's 2008 GDC presentation, "Successful Instrumentation: Tracking Attitudes and Behavior"
Figure 2: www.bungie.net/online/heat_maps.aspx
Figure 3: <http://jimblackhurst.com/wp>
Figure 4: Visualizations from analytics work at Square Enix, Crystal Dynamics, and IO Interactive.
Figure 5: <http://altdevblogaday.com/2011/06/01/balance-and-flow-maps-2>
Figure 6: <http://research.microsoft.com/pubs/140998/WowGroupMovement.pdf>

a certain area of the map form a track. A flock describes a set of players traveling together in a way that all members are within a certain diameter or spatial range at each point of time. In a leadership pattern, players might join the movement of a leading player at some point in time. This kind of analysis is useful to evaluate player behavior in team-based games like TEAM FORTRESS 2 or player tactics in MMORPG instances, and to debug bot behavior.

Applying data mining techniques for trajectory data has been already employed for various use cases in game analysis; **[Figure 6]** shows trajectory data from the WORLD OF WARCRAFT Battlegrounds, where the distribution of the players in time and space can reveal design problems in the environment or successful strategies.

Spatial game analytics—and game analytics in general—have the potential to transform

game development in much the same way it has transformed marketing, web applications, and environmental modeling, but games are unique beasts and it remains to be seen how big an influence analytics will have.

Currently, the reliance on analytics tools from sectors outside of game development means that the barrier of entry for nonexperts can be relatively high for some types of spatial work, but the current innovation in game analytics is probably going to change this in the coming years as more and more tools become available. Also, gaining insights from spatial analytics does not require complex modeling—try plotting some data on a game map and see where it takes you. 📍

ANDERS DRACHEN is an assistant professor at Aalborg University and lead game analyst for Game Analytics. Follow him on Twitter @andersdrachen.

dear esther

how thechineseroom turned an experimental first-person shooter mod into an award-winning indie darling

BY DAN PINCHBECK

DEAR ESTHER started out in 2007 as a university research project exploring different ideas about story and gameplay in a first-person shooter. The central question was pretty simple: Could you strip everything resembling traditional FPS gameplay out of the traditional first-person universe, leaving just a story and a world, and still build an engaging, rich experience?

The mod was something of a cult hit, and in 2009, in collaboration with Rob Briscoe from Littlelostpoly, we started developing a commercial remake that was released on February 14, 2012. It sold over 50,000 units in the first week, paying back the game's investor, Indie Fund, in just under six hours. To date, DEAR ESTHER has sold more than 125,000 copies—not bad for a project that started life in a bleak cupboard in an equally bleak university building!

DEAR ESTHER was simply an experiment that we never expected to do more with than generate some data for an academic study. We formed thechineseroom around the game, so we've been on a pretty steep learning curve in terms of both development and running a studio, but I think the fact that we came at DEAR ESTHER completely blinkered by idealism turned out to be crucial to the things we got right with the title.



what went r i g h t

1. STARTING FROM A MOD CAN BE A GOOD THING

We launched the original DEAR ESTHER onto ModDB in 2007. Jessica Curry made an amazing soundtrack, and Nigel Carrington's voice-over was brilliant, but I kept my expectations low; DEAR ESTHER was short, slow, and about as far away from most FPS mods as you could get. Around the same time, Tale of Tales had released THE PATH, and although I hadn't played it, I was aware of some of the friction it had caused, and was expecting at best a few hundred hits and maybe enough public response to justify the grant by yielding some data we could use.

I was completely wrong.

The mod racked up thousands of downloads in a matter of weeks, and got picked up by a couple of critics who just loved it. We owe both the critics and the community a lot, because without that support, there would never have been a remake. We knew when we went into the remake that there was already a fan base, that the concept worked, and we had some really dedicated people out there spreading the word. The mod ended up functioning like an open beta, which was really powerful for us.

Another benefit of having a mod out there for a couple of years was the community had

come up with some amazing interpretations of the story that I hadn't thought of when writing it. When we recorded new voice-overs for the remake, we added a few things to encourage those interpretations, in part to add to the story, but also to give a nod of thanks to the mod community. Those new cues we added have in turn yielded a completely different interpretation of the story. I love that the game has continued to evolve.

2. SOUND IS POWERFUL

DEAR ESTHER wouldn't be the game it is without its amazing soundtrack and voice acting. Composer Jessica Curry and I had worked together before on a few multimedia art projects going back to 2003, so I knew her work really well and how the tone of her music would contribute to the game. We imported previous music by Jessica into the engine early on, and listened to it a lot while we were writing the story and building the game, which I think was instrumental in setting the emotional tone. Music is an investment that can't fail to pay off—give your composer a lot of space and freedom and it will make all the difference.

The actual story was written very late in the development cycle; I'd estimate the world was 75% complete before I started writing. When Jessica and I were casting, we heard Nigel Carrington's voice, looked at each other, and

said, "That's him." It was an instant decision.

When we went into the studio, we did three takes of each cue. For the first, I would let Nigel find the emotional tone and intensity that he felt was natural to the words, then in the second and third takes I'd have Nigel push the intensity up and down a notch from there. We ended up using the first take on most of the cues. Get a good voice actor, and you can trust his instincts.

3. ART DIRECTION IS CRITICAL

The world's relative lack of detail in the original mod made the players pay more attention to the voice-overs. With the remake, we aimed for something deeper and subtler, using the environment as a means of steering not just the players' movements, but their emotional journey. Rob calls this "subliminal signposting," manipulating mood and interpretation through the world.

The remake of DEAR ESTHER brought the island to the center, making it a living space that followed the principles of ambiguity and abstraction in the narrative. We did this by using randomized objects that operated against the narrative as much as with it, and by focusing heavily on lighting and the actual shapes of the geometry to influence the players' mood. In the final level, for example, the shape of the distant mountain is a torso and head of a sleeping figure. It's subtle, and it's difficult to determine whether it has an explicit, identifiable impact on all players, but the game uses many



instances like this to suggest the island has a more complex relationship to the narrator than simply being a realistic space.

A key location in the game that makes this more explicit is the caves level. The caves were originally designed to provide a break in tone and space from the open levels in the beginning, though they also served as a device that let us move the time of day radically from afternoon to night. Once the player enters the caves, the game breaks away from reality in a very obvious way, synchronizing with the voice-over's shift in tone to something more fragmented and confused. Rob created areas in the caves that were highly suggestive and deeply unreal, such as a heart-shaped cave lit orange and pink, in order to evoke an emotional response from the player.

Later, a chimney stretches up toward the sky to create a sense of being deep underground, tying in with the old cliché of the light at the end of the tunnel—or the view toward light sometimes mentioned in near-death experiences. The chimney was clearly not just a natural feature; it represented something more psychological, possibly even spiritual. At the foot of the chimney is a clear pool, scattered with coins—a wishing well—which is followed by a crawl through a tight, claustrophobic tunnel covered with strange symbols and scrawled references to the Bible. Then the player drops into a dreamlike representation

of an underwater motorway. The overall effect is to constantly shift the player away from just seeing the environment as a backdrop, however beautiful, into something that tells its own version of the story. Rob's excellent art direction was able to maintain and communicate DEAR ESTHER'S story even without the voice-over.

4. INVESTING IN SPACE AND TIME AS DESIGN TOOLS

The thing that makes DEAR ESTHER work more than anything else is that the player has very little to actually do, which creates a space for the player to think and feel. If you set up powerful images and ideas, then the player will naturally want to consider them, so we made sure to leave opportunities for that to happen. When I said at GDC 2012 (See **References**) that overstimulation can kill atmosphere, what I meant was that if you expect your players to respond to the system all the time, you don't leave space for them to respond to the stuff in their own heads.

This isn't a new thing—SHADOW OF THE COLOSSUS relies heavily on those long periods of reflection—but I think DEAR ESTHER was different in that we were loading the player up with these symbols and ideas and making those reflective periods directly tied to trying to square away all the ambiguous and conflicting things they were

seeing and hearing. One of my favorite quotes is from the literary theorist Frank Kermode, who said, "Why does it take a more strenuous effort to believe that a narrative lacks coherence than to believe that somehow, if one could only find it, it doesn't?" That idea lies at the center of DEAR ESTHER, but it requires space and time to work.

Also, different emotions run at different speeds. Joy and anger are very fast emotions, they arrive quickly and can be felt in a short time-frame. Sadness or loss are much slower and easily disrupted. If you want a player to feel lonely or sad, you need to time and space for that to happen. In a game like S.T.A.L.K.E.R., the environment is desolate and empty, which gives players time to trudge around, submerged in their response to that emptiness.

As players, we're so used to stuff happening in games all the time that when nothing is happening we start feeling on edge. This is a cornerstone of horror games. So underpinning DEAR ESTHER is the tension of waiting for something to happen, and this desolation and isolation, and these weird images that just ache to be formed into proper understanding, and then lots and lots of time and space to be introspective about that mix.

5. TRUST YOUR PLAYERS

It always makes me smile when we get accused of being hipsters, or intellectual snobs, which



was always going to be a response from some gamers. Aside from the fact I'm a massive kill-junky who'd take PROTOTYPE over HEAVY RAIN any day of the week, I think what's going on with DEAR ESTHER is the opposite.

We trusted gamers to be adaptable, open to a slightly different experience, able to think and feel for themselves. They don't need their hands held, or things always spelled out. They are sophisticated, media-literate, smart people. It's okay for things not to make sense. Players will cope with that, and it's often a more rewarding experience. Like most things we do, this is inspired by games. I love the fact that after dealing with the biomass in D6 in METRO 2033, the nearest you get to an explanation is Miller saying, "If only you'd seen what lives down there!"

I think it's a strength of a narrative to not have to join all the dots. The best books throw away ideas left right and center that you never find out more about. It creates a sense of scale and mystery. That's not going to work for all games—lots of games rely on delivering explicit history and mythos, like MASS EFFECT or SKYRIM—but there's more space for the unknown in games. I think it's a really powerful device, but you have to trust the community.

The other analogy I often use is that you don't look at a Jackson Pollock painting and obsess over which order the paint was applied

in. It's an overall effect—you feel it, and that's enough. We trusted that players would get that. We believed that they would take the experience as it was and immerse themselves in it, and that belief paid off. That's the biggest lesson of all for me from making DEAR ESTHER.

what went wrong

1. UNIVERSITY COLLABORATION IS COMPLICATED

This needs a caveat: We began life as a University of Portsmouth research project, funded by the Arts & Humanities Research Council (U.K.), and they've both been good to us, but we had a few problems as well. Universities have pretty strict policies about everything from employment to IP, and we faced an uphill battle to gain their respect and trust. The last two are inevitable.

One of the things I love about the game industry is that it is product-driven. Everyone has an idea about a game, or an opinion about what makes a good game, and the problem is that academia exists to be critical about these things, which means it generates a lot of opinions and no product. This doesn't sit easily with the game industry, where you earn your stripes by making a game and having it assessed by the community and the market. Once we'd actually made something and put it out there, not as a

protected academic experiment, but a game to be played and assessed as any other, suddenly people started talking to us. I'd have built it sooner if I'd been less idealistic about academia's relationship with industry, and I'm much more careful now about respecting just how damn hard it is to make games.

On a practical level, we discovered that commercialization is a very hard thing for universities to pull off. There are standard clauses in distribution agreements, like who takes liability and to what value, that are almost impossible to square with a university's insurance protection. We were lucky to survive running into this one—the university was happy to release the IP, and we were lucky to find Indie Fund. But for about six weeks I thought the project was dead and I was out of a job. It was hellish, stressful, gutting.

If I'd known going in what we'd face, I'd have set things up a long way in advance. That's not to say that working with universities is impossible—they can be amazing partners—but you need to wrap your head around a very different way of working and set of priorities. They work on a timescale that can be extraordinarily slow, so you need lots of lead-in time. The decision-making process can be very distributed, which makes it hard to get financial agreements, NDAs, and other contracts signed compared to your average small business. Also, a university has a direct responsibility to its students, so they're constantly



searching for teaching and research opportunities. If you can figure out what kind of opportunities you can offer a university in advance, you can speak in the right language—learning, generating publicly available information, and so on. You are also dealing essentially with public money, which changes what you can do politically.

Finally, if your funding is running from a grant, like we were, then studio processes like staffing have to be done very openly; you have to make a case for why you'd want a specific individual (from knowing them, or reputation, or recommendation) without going through an open recruitment process. On top of that, you have a responsibility to get changes in the budget cleared, which can slow you down quite a bit.

Overall, they are slow. Really slow. As a small independent studio, one of your biggest advantages is how responsive and fast you are. Trying to operate as an indie in the context of a slow moving monolith like academia can be very frustrating. I'm not sure I'd do it again. There's got to be a better model, and we're going to keep looking for it.

2. DECIDING TO PORT TO A NEW ENGINE VERSION CAN BE A WORLD OF HURT

This is how the conversation started: "Hey, Rob, you know we ought to ask Valve if we can migrate from THE ORANGE BOX to the PORTAL 2 version of the Source Engine. That'd be straightforward, right?"

DEAR ESTHER was due out in early summer 2011, but it didn't come out until Spring 2012—because of the port.

If it's your own engine, fine. If you know an update is coming, great. If you research what changes are going to happen when you move to a new version, brilliant. Spur of the moment decisions without really casing out what is going on under the bonnet of a new version? I'd advise against it. We went in unprepared and a little naive and paid the price. That's not to say it wasn't worth it—we couldn't have done certain things with DEAR ESTHER without the migrate. It made the Mac port easier and has benefits for potential console ports further down the line, and gave us improved shaders, particles, and visuals (as well as being a more optimized, stable engine and toolset) but I wish we been a little more prepared for how complex it was going to be. All credit is due to Rob and our programmer, Jack Morgan, for making it work. But it was tricky.

3. VOICE-OVERS SOUND DIFFERENT IN THE STUDIO THAN IN THE GAME

This didn't impact on the final build, and is more about the mod version, but it's an interesting one: When we first made DEAR ESTHER, we brought in a local actor to record a script I'd written over a three-day period as a kind of stream-of-consciousness thing, and this was

really useful. What I hadn't figured was that what sounded great in the studio, as a radio play, was completely emotionally overcooked when we put it in the game. I've still got the original recordings, and as pure audio they work really well. In the game, they just felt slightly melodramatic.

I still don't know what it is exactly about this "amping-up" effect on voice-overs with games, but you see it all the time even in high-budget games, where a voice-over you think would probably work fine on its own ends up overblown. Emotional tones seem to get inflated in-engine, and when you are working with voice-overs it's really important to keep the drama nailed down tightly. Nigel is incredibly subdued in the final takes of DEAR ESTHER—I think there's only one or two instances where he even raises his voice—and despite that, it's a really emotional performance with quite a wide range to it.

I've just finished MASS EFFECT 3 and I noticed that BioWare did a similar thing with its voice-overs—quieter moments and a toned-down delivery often achieve the best emotional results. We had this problem again with the KORSAKOVIA mod we made in 2009—what sounded great in the studio didn't work in the game, and since recording sessions can be time-consuming to organize, you want to get it right first time. Now we always take a copy of the game into the studio with us and get a few rough cuts into the engine before we commit to a take.



4. THE GAME INDUSTRY WORKS IN U.S. DOLLARS

This is really short and only applies to non-American developers: If you are making a game for digital distribution on PC, your major source of income is going to be Steam. That means getting paid in dollars, which means you have to deal with exchange rates. Now imagine that your investors are U.S.-based, and you have to pay them back. That's going to be in dollars too. Not only are you then dealing with exchange rates in two directions, but the bank will apply different rates for conversions backward and forward, so if you don't have a bank account set up to deal with USD, you could lose a lot of money. We didn't have a dollar account set up by the time we had to pay back Indie Fund and it hurt us financially.

It's a bit gutting to see your hard-earned cash vanish into a banker's pocket (particularly at the moment). Get a dollar account attached

reply to them on forums. Other distributors will want to talk to you and sign you up, so you have to go through contracts, manage all of that. Game sites you've never heard of want free copies to give away to their readers. Freelance journalists want review copies and you need to assess how serious they are. Your accountant wants to know how U.S. VAT works and you have no idea. You're trying to maintain a PR campaign without repeating yourself constantly, you're trying to manage incoming bugs, get community-driven localization dealt with, and look ahead to porting opportunities.

I was pretty naive about just how much time it would all take. We didn't exactly have much of an option in terms of managing our workload around the time DEAR ESTHER released, and we didn't have the cash in the bank to hire someone to help manage all that stuff when it hit, but if I had known how important all that post-release work was, I would have blocked out

reference

DEAR ESTHER TALK AT GDC 2012
<http://gdcvault.com/play/1015529/Dear-Esther-Making-an-Indie>

(around the whole "is it a game?" question). This isn't something we were ever particularly interested in, and DEAR ESTHER absolutely is not any kind of critique of games, but there's no doubt it helped exposure. I wouldn't ever court controversy, but I can't deny it helped.

Second, I think that pushing the production values was massively important. Because of the gameplay, which tends to be very mechanic-oriented in play and retro in feel, DEAR ESTHER doesn't sit easily within the indie sector. Beautiful visuals made the game attractive to a wider audience, and that level of quality was critical in captivating players while the slower burn of the emotional experience had a chance to hook them. I genuinely believe that without the emphasis on presentational quality, the game wouldn't have succeeded in the same way.

Third, there's an attention to detail in terms of the player experience that isn't just about a different, new, neat idea, but about the minute-by-minute play. Rob's focus on always having some form of reward at any given point in the game, the way that the soundtrack repeats motifs in different ways to constantly keep this kind of tide of emotions going, and perhaps most importantly, creating an emotional architecture for the player and then standing back, not overloading them with things to see, hear, do, just giving them time to think and feel. A game may rest on a great idea, but making that great idea ever-present and always-optimized is the difference between a good game and a bad one.

For all its experimental beginnings, DEAR ESTHER grew out of a love for FPS games, and I've always said I see it as a logical extension of the design history of that genre. Underneath all of those things I still maintain there's a pretty simple reason why DEAR ESTHER worked and succeeded. It's a love letter to FPS games, shot through with our passion for the genre and what it can do. I think players see that, and I think it's another case study of how that combination of passion and hard work pays off—and lets us build the most exciting medium on the planet. 🎮



to your business account, and if your bank doesn't have the facility to do this, switch. We only found out about this from Toxic Games when they hit it a couple of weeks before us (they were also distributing on Steam and supported by Indie Fund), but it was too late for us. Given that you make most of your money in the first couple of weeks, I wish all of us could have found out sooner.

5. POST-RELEASE ADMINISTRATION TAKES TIME

In other words: Don't dive headlong into two new games on top of releasing your first. (You could file this in the "Nice Problem to Have" category.) If your game is a surprise hit, people will want to talk to you. Fans are critically important to a small company; they are your ambassadors, and you have to respond to them, write back to them, and

more time in our development schedule for it.

Your highest point of exposure and currency as a studio is the time immediately after you release a game. We already had two games in the works, but next time around, I'll be ready to suspend everything for a week or two and clear the backlog faster.

what we learned from dear esther

First, like Capy Games's Nathan Vella said in his GDC 2012 talk, sometimes taking risks is less risky than not taking them. DEAR ESTHER stands out from the crowd—it looks and sounds like a triple-A title, but plays like nothing else from that sector. It's a difficult game to categorize, and this kicked off a lot of online discussions

DAN PINCHBECK is creative director of the chineeserom, which basically means he writes, designs and produces their games. He's got a PhD in first-person shooters, has just written a book about DOOM (due out Spring 2013) and is leading the studio on two new games. AMNESIA: A MACHINE FOR PIGS is the sequel to Frictional Games' 2010 cult horror (due Q1 2013) and EVERYBODY'S GONE TO THE RAPTURE, an openworld purestory game (release date TBD). He is also a reader in Computer Games at the University of Portsmouth, UK.



Services

Portishead

Royal Portsbury Dock

2mil

A369

!! We trusted gamers to be adaptable, open to a slightly different experience, able to think and feel for themselves. They don't need their hands held, or things always spelled out.

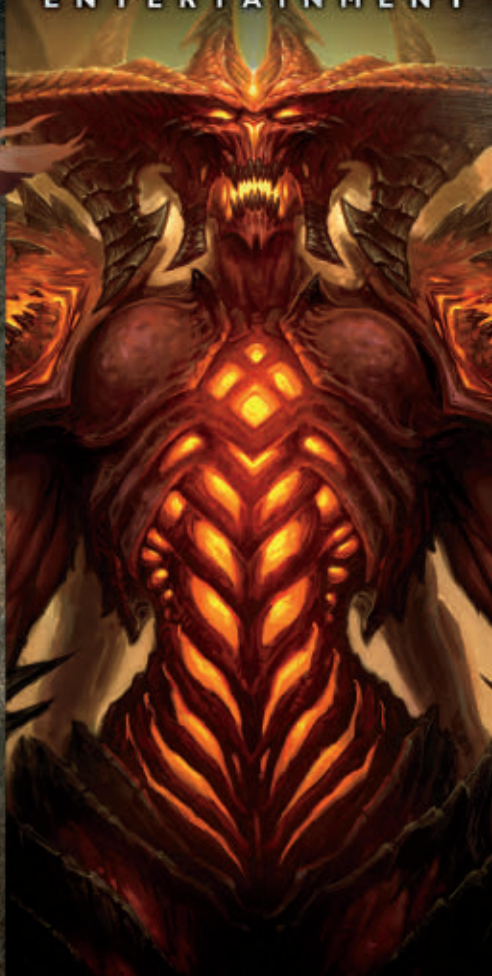
They are sophisticated, media-literate, smart people. It's okay for things not to make sense. Players will cope with that, and it's often a more rewarding experience. Like most things we do, this is inspired by games. I love the fact that after dealing with the biomass in D6 in METRO 2033, the nearest you get to an explanation is Miller saying, 'If only you'd seen what lives down there!' !!

BLIZZARD®

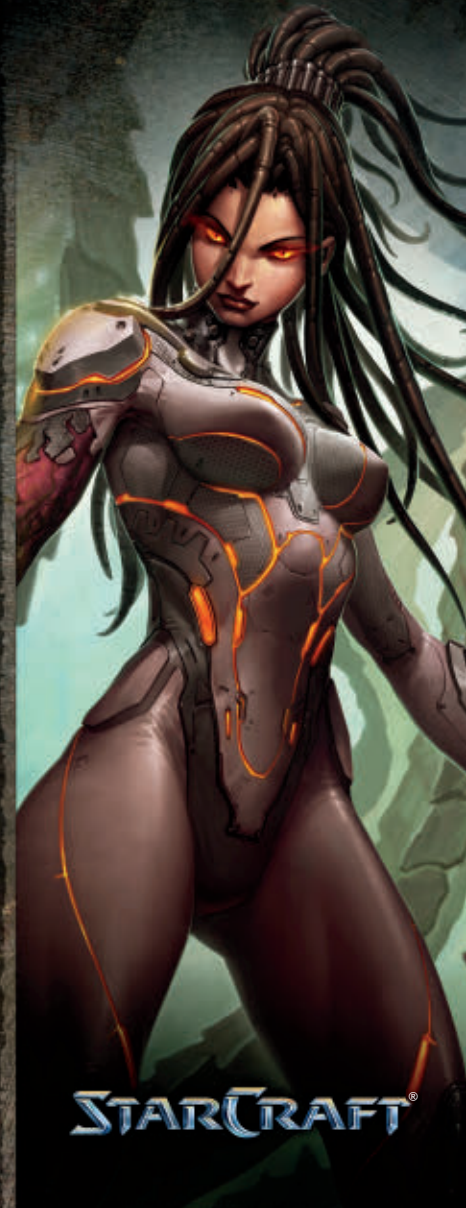
ENTERTAINMENT



WORLD
WARCRAFT®



DIABLO®



STARCRAFT®

BLIZZARD IS HIRING

We are actively recruiting for the following key positions across our game and online technology teams:

SENIOR SERVER ENGINEER | SENIOR RELIABILITY ENGINEER | SENIOR TOOLS ENGINEER
SENIOR CONSOLE ENGINEER | LEAD 3D ENVIRONMENT ARTIST
SENIOR 3D ENVIRONMENT ARTIST | SENIOR 3D CHARACTER ARTIST | FX ARTIST
LEAD BATTLE.NET DESIGNER | LEAD CAMPAIGN DESIGNER | SENIOR LEVEL DESIGNER
PRODUCTION DIRECTOR | BUSINESS OPERATIONS DIRECTOR

jobs.blizzard.com

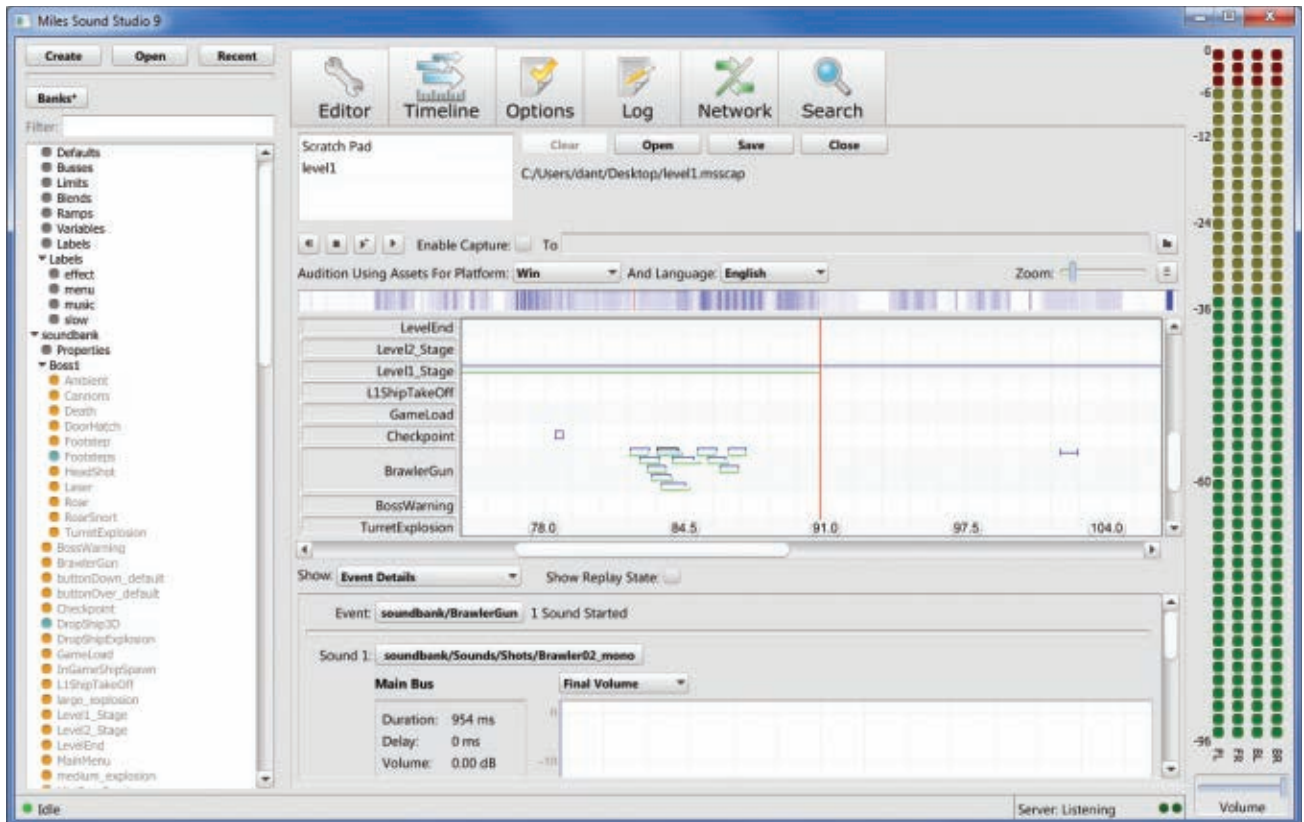
Follow us on Twitter: @blizzardcareers



RAD GAME TOOLS

Miles Sound System 9

BY ALEXANDER BRANDON



The Miles Sound System is the longest-running commercially available sound middleware for games. Miles was developed by John Miles in the early 1990s, and acquired by RAD Game Tools in 1995. Version 9 brings a new feature called Timeline, which brings Miles into the modern era.

Miles's compression includes support for Bink audio encoding (of course), a clean-room Ogg implementation, and MP3 rights. Channel control and surround support, and the means to create

as much high-level functionality as you want (such as custom effects, the ability to control channel count for things like rapid-fire weapons, metrics reporting, and profile for memory) are also included. When compared with its competitors, Miles holds its own—it's fast, cross-platform, and easy to integrate into your game's codebase, but it was mostly limited to managing your sound bank rather than giving you direct control in the game runtime.

I did some high-level functionality creation alongside some Obsidian programmers around 2007, using Miles and a proprietary engine. But over the years, Miles has started to show

its age. For some time, Miles was little more than a robust and reliable low-level system, providing the core functionality most game engineering teams needed to play back just about any audio they wanted. It was (and still is) built like a tank, with hardly any crashes or bugs, and most senior engineers know it won't give them any surprises. Since then, though, higher-level graphical user interfaces, real-time controls, and advanced bank management across all platforms were being exploited by Miles's main competitors—AudioKinetic's Wwise and Firelight Studios's FMOD. Wwise and FMOD became the Sony

RAD GAME TOOLS
Miles Sound System 9
 550 Kirkland Way - Suite 406
 Kirkland, WA 98033
 425.893.4300

PRICE

> \$6,000 per game per platform

SYSTEM REQUIREMENTS

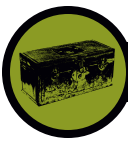
> Windows Vista and higher

PROS

1. Timeline editor
2. Robust code base
3. Wide platform support

CONS

1. No Android support yet
2. Lacks control surface connectivity
3. Lacks effect plug-in capability



PlayStation 2 and Microsoft Xbox to Miles's Sega Dreamcast. I believe that's about to change.

At this year's GDC, RAD showcased the newest version of Miles, including Miles Sound Studio 9. Its Timeline feature might change the way we handle game audio.

Up until the last few years, designing and manipulating game audio within a game build was a time-consuming challenge for audio designers—often far more frustrating than, say, audio postproduction work on a film. With Miles's Timeline tool, you can play a build of the game and see the audio events reproduced visually in the order they play during your gameplay session. For the first time, you can click on any event and change either the individual attributes of that event, or employ a sweeping change that affects all events of that type.

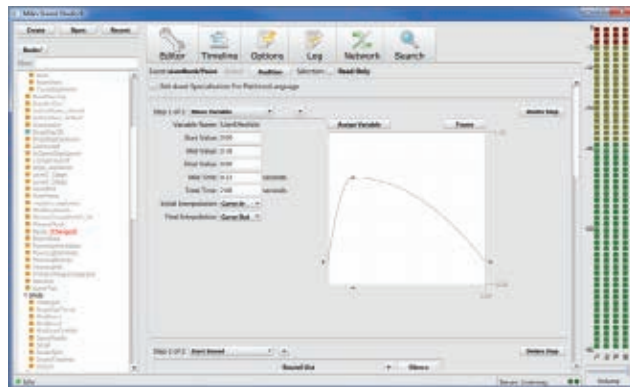
I gave this a whirl using the included Breakout-clone sample game. You have a Start button that captures events when Miles Studio is open, even if no sounds are assigned to them. You can then click on the events and assign sounds and properties as you see fit. In other audio engines you can't just rewind a play session and examine the audio except in profiling tools.

Timeline is particularly useful when the engineer is integrating the audio into the game build itself. During this process, the engineer needs to edit files in a bank

manager and continually restart builds, which can take as little as two minutes if he's swapping out a file, or as long as five minutes with a lot of parameter changes not handled in real time. With Timeline, you get instantaneous feedback; two minutes per change could take up hours of that engineer's time, but Timeline brings that down to mere minutes.

This is where you say "That sounds great, but where do events get assigned in the first place?" Miles's lead developer Dan Thompson also has a few tricks up his sleeve to make Miles easy to work with—specifically, the team has built in easy ways to integrate Miles with existing widespread middleware such as Unity and Unreal Development Kit.

For example, once you've integrated Miles into your UDK project, you'd go into Kismet and use Event objects from Miles rather than Sound Cues, connect them to whatever game event you see fit, and see the results show up in the Miles Timeline. For Unity, Miles's well-documented set of hooks into Unity's scripting system allows a sound designer to enter the hooks at the appropriate points in the script. It's worth noting that Unity integration is a bit more complicated than UDK integration, because the audio designer needs to be familiar with whatever scripting language the team is using (JavaScript, Boo, or C#) in order to insert the appropriate



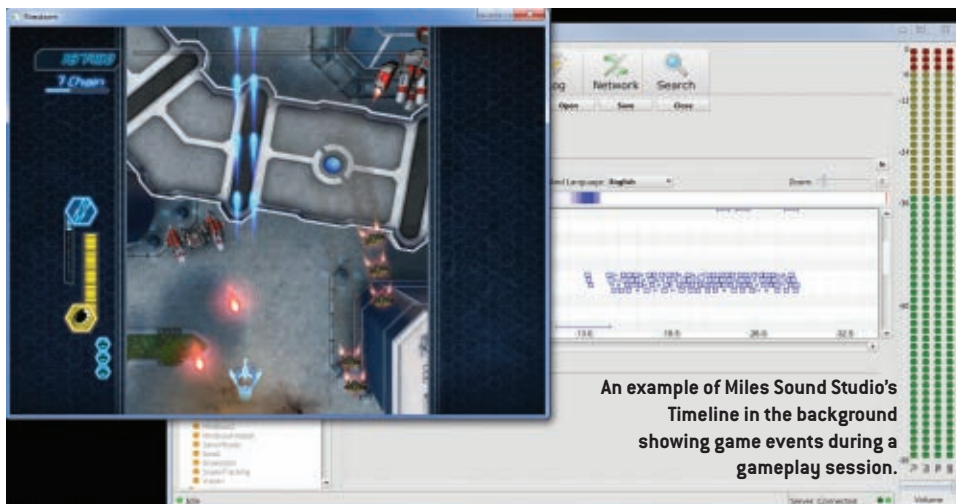
Miles hooks, whereas with UDK the designer can simply use Kismet once the low-level code is in place.

The rest of Miles's designer tool is fairly straightforward and follows the same logic as Wwise and FMOD. The sound files, related characteristics, and the events that trigger the sounds are represented as a nested hierarchy similar to a file and folder directory tree. Latency is pleasantly low when playing back files on both PC and 360 (my testing platforms). The designer tool has the usual box of tricks, including randomization of pitch and volume across a preset range, fade and crossfade control, and simple envelope control and basic effects, such as reverb for environments. In addition, there is support across the board for almost every game platform. The most notable missing platform at present is Android. Also new to Miles 9 is "asset hotloading," which keeps track of

all your edits to assets through the Network tab and lets you send out those edits to the rest of your team.

Miles still lags behind in a few key areas, though—chief of which is its lack of support for pro audio tools and effects plugins. Audio engineers are slowly adopting pro audio tools, such as the Mackie Pro Control, which have faders and knobs that you can assign to GUI representations onscreen. A Pro Control (or an SSL Nucleus, another popular pro audio manufactured control surface) can be to an audio designer what a Wacom tablet is to a game artist: It gives the audio designer a much faster way to manipulate audio properties. Also, Miles doesn't yet have any plug-in architecture for pro effects such as iZotope and Waves. Pro effects such as these are used in hit albums and film soundtracks as well, and both Wwise and FMOD have used them for several years now.

Overall, I believe Miles is ready to jump back into the fray of competitive mainstream audio engines, as Timeline combined with its optimized low-level code will be an excellent choice for engineers and audio designers alike—but some audio professionals might find that Miles is simply missing a few key features that its competitors have offered for years. 🎧



An example of Miles Sound Studio's Timeline in the background showing game events during a gameplay session.

ALEXANDER BRANDON is a composer, sound designer, voice actor, producer, and director for games with 18 years of experience in the video game industry. He is currently president of Funky Rustic, an audio outsourcing group in Georgetown, Texas.

JUST BECAUSE YOUR GAME IS AVAILABLE IN A COUNTRY, IT DOESN'T MEAN YOU ARE IN BUSINESS THERE

Xsolla offers localized payment solutions to your users and effective currency and tax management to you, ensuring you get the most from each and every transaction.



Our software platforms, Apps, and API are easy to integrate and online almost immediately.

Our 24/7 support is there to ensure that any issues that may arise are taken care of for you.

Xsolla provides you with over 350 Payment Solutions in over 85 countries. Because no two customers pay the same way.

*Visit us : www.xsolla.com
Email : bizdev@xsolla.com*

MAKE MORE ENEMIES

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

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

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



VFS STUDENT WORK BY NIKOLAS LAZAR

VFS

Find out more.
vfs.com/enemies

THE ONLY ONE-YEAR PROGRAM
IN PRINCETON REVIEW'S 2012
TOP GAME DESIGN PROGRAMS

GDC 2012 ONLINE

GAME DEVELOPERS CONFERENCE® ONLINE

AUSTIN, TEXAS | OCTOBER 9-11, 2012 | EXPO DATES: OCTOBER 9-10



Join us at GDC Online, October 9-11, for three days of world-class online and connected games content led by top industry experts.

SESSIONS

TUESDAY-THURSDAY

- 💰 Business & Marketing
- 👤 Customer Experience
- 🎮 Design
- 🏭 Production
- 💻 Programming
- 💰 Monetization (sponsored)

SUMMITS

TUESDAY-THURSDAY

- 📖 Game Narrative Summit
- 🏠 Game Dev Start-Up Summit
- 🎮 Gamification Day
- 📱 Smartphone & Tablet Games Summit

EXPO FLOOR

TUESDAY-WEDNESDAY

Explore the latest connected game technologies and innovations, and connect with product experts.

FEATURING

TUESDAY-WEDNESDAY

game developers choice
ONLINE AWARDS

GDC
Play

Register before August 24 and save close to 40%!

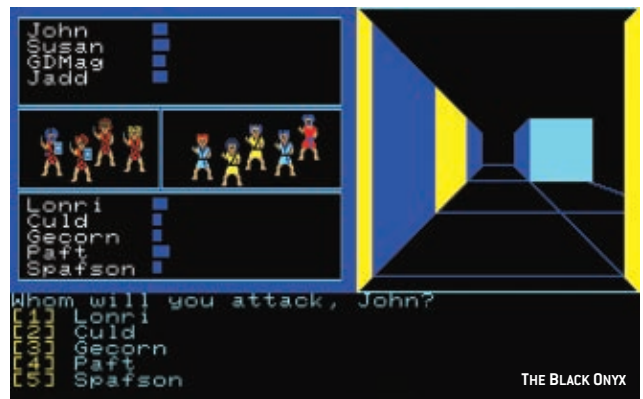
VISIT GDCONLINE.COM FOR MORE INFORMATION





A BASIC HISTORY OF BASIC

////////// AT 48 YEARS OLD, THE BASIC PROGRAMMING LANGUAGE IS OLDER THAN MOST WORKING GAME DEVELOPERS. WE ASKED ITS CREATORS AND EARLY ADVOCATES ABOUT ITS INCEPTION AND HISTORY, AND HOW IT INFLUENCED THE MODERN GAME INDUSTRY.



The BASIC programming language, or Beginner's All-purpose Symbolic Instruction Code, turned 48 this year. That's an extraordinarily long life for tech—even longer than the commercial video game industry has been around. Since its inception, BASIC has proliferated from a single source to include now over 280 variants and growing. Just about every single model of computer (including a few game consoles) has had at least one variant. Many veteran game developers got started working with BASIC out of their bedrooms on early personal computers, and BASIC remains popular with hobbyist coders.

The fathers of the original Dartmouth BASIC are Dr. Thomas Kurtz and the late Dr. John Kemeny, who died in 1992. Kurtz, who turned 84 this year, spoke with absolute clarity as he recounted BASIC's launch: "That first BASIC saw the light of day on May 1, 1964, at 4 a.m. We were a team, but [Kemeny] was always the team leader. He did the politicking and the money-raising that was always necessary while I attended to the details. But he was always careful to give me full credit for my share of the project, whatever the project. I had already used the word BASIC for a statistics text I wrote several years earlier [Basic Statistics, 1963]. John Kemeny liked acronyms, and so we settled on BASIC, for which he devised the expansion 'Basic All-purpose Symbolic Instruction Code.' By the way, it has always been 'Symbolic' and never 'Simplified.'"

BASIC BEGINNINGS

» Kurtz was first introduced to computers in 1951, when he spent the summer at the Institute for Numerical Analysis at UCLA, which was in the process of building the Standards Western Automatic Computer (SWAC), a clunky contraption that used both vacuum tubes and Williams tubes for

Top left: An early publicity photo for True BASIC with Dr. Thomas Kurtz, left, alongside Dr. John Kemeny. Photo courtesy of Dartmouth College Library. Bottom: Chip Elliott, John Kemeny, Tom Kurtz, Chris Walker, and Dave Pearson soon after commercial operations of True BASIC began. Photo courtesy of John Lutz.



```

COMMAND: EAST
COMMAND: NORTH
MOUNTAINS ARE IMPASSABLE!
COMMAND:
H.P.=84
FOOD=77
EXP.=1
GOLD=88

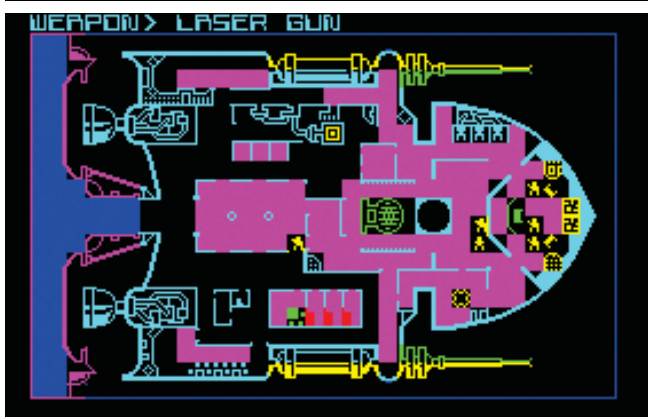
MOVING: YOU CAN MOVE ONE ROOM (OR ONE TUNNEL).
ARROWS: YOU HAVE 5 ARROWS. YOU LOSE WHEN YOU RUN OUT
EACH ARROW CAN GO FROM 1 TO 5 ROOMS. YOU AIM BY TELLING
THE COMPUTER THE ROOMS YOU WANT THE ARROW TO GO TO.
IF THE ARROW CAN'T GO THAT WAY (IF NO TUNNEL) IT MOVES
AT RANDOM TO THE NEXT ROOM.
IF THE ARROW HITS THE WUMPUS, YOU WIN.
IF THE ARROW HITS YOU, YOU LOSE.
HIT RETURN TO CONTINUE?
WARNINGS:
WHEN YOU ARE ONE ROOM AWAY FROM A WUMPUS OR HAZARD,
THE COMPUTER SAYS:
WUMPUS: 'I SNEEL A WUMPUS'
BAT: 'BATS NEARBY'
PIT: 'I FEEL A DRAFT'

HUNT THE WUMPUS

I FEEL A DRAFT
BATS NEARBY!
YOU ARE IN ROOM 15
TUNNELS LEAD TO 6 14 16

SHOOT OR MOVE (S-M)?

```



Top left: The first ULTIMA on the Apple II was programmed in BASIC—due to their slower nature, RPGs and adventure games were the perfect fit for BASIC. Top right: AKALABETH was the predecessor to ULTIMA, containing many elements which would later feature in the series. AKALABETH was written entirely in BASIC, and comes bundled with QB64 as example code (pictured). Bottom left: HUNT THE WUMPUS was a text-based game from the early 1970s. So popular, the creature made cameo appearances in other games, such as M.U.L.E. Bottom right: REBELSTAR RAIDERS on the ZX Spectrum, circa 1984, was an early BASIC title by veteran strategy developer Julian Gollop. It featured single-screen maps and tactical gameplay.

internal memory. He created a program that used SWAC to deal with tail probabilities for statistical distributions.

Five years later he met Kemeny. "Although John Kemeny had been at Princeton while I was a grad student there, I didn't meet him until he was recruiting for Dartmouth in 1956," Kurtz said. "I was recruited to teach statistics, among other things, and it turned out that IBM, MIT, and UCLA were establishing regional computer centers in the East and the West. To earn extra money I was made a 'research associate' to this effort and, even before I showed up at Dartmouth for my first class, I attended a course at MIT on Share Assembly Language (SAP) for the IBM-704."

But SAP was complicated, and Kemeny wanted to make programming accessible for students without a science background, because those students often became future decision-makers

in business and government. "In some of my talks in those early days, I called programming in BASIC a 'learning machine,'" Kurtz said. "I've always felt that being able to program meant that one had to develop the ability to abstract, to construct 'mental models' of something or other. I feel that this ability leads to being able to make contributions in fields other than the sciences."

So they needed a programming language that was more accessible than Assembly. "To make SAP more palatable to nonprogrammers at Dartmouth, John Kemeny devised DARSIMCO, Dartmouth Simplified Code, by presenting SAP in terms of three-instruction sequences," Kurtz explained.

For example, to perform $A + B = C$, you would write:

```

LDA A
FAD B
STO C

```

DARSIMCO and SAP were both abandoned in 1957, when FORTRAN arrived at Dartmouth. Although higher-level languages were generally believed to be inefficient compared to assembly language, Kurtz discovered that coding in higher-level languages saved both computer usage time—which was a valuable and limited commodity—as well as his personal time. So he continued to work with higher-level languages that would inform BASIC's development later on.

In 1959, an undergraduate student without a computer background, influenced by FORTRAN, prepared a higher-level language and compiler he called DART for the LGP-30 computer (a rotating-drum machine), which caught Kurtz's attention. Around 1961, Kurtz and several students worked on an ALGOL compiler for the LGP-30, which was later modified to be self-contained and named

SCALP. This new language was subsequently taught to hundreds of students until BASIC replaced it.

Meanwhile, Kemeny worked on one more important language in 1962 before BASIC, again for the LGP-30. As Kurtz explained:

"John Kemeny devised a simplified language for that environment he called DOPE, 'Dartmouth Oversimplified Programming Environment,' which anticipated BASIC in having one instruction per line and having instructions such as:

$5 + A B C$ to indicate 'Add $A + B$ and put the result in C '

This naturally led to the BASIC form $5 LET C = A + B .$ "

DARSIMCO, DART, ALGOL 30, SCALP, and DOPE didn't last long and weren't massively successful, but they all influenced Kemeny and Kurtz

with regard to BASIC's fundamental principles, which he explained as:

- A) *We made no distinction between floating-point arithmetic; we used only double precision floating-point internally and adopted several strategies to make loop termination come out right.*
- B) *No format-control on INPUT and default format for printed output.*
- C) *One line, one statement.*
- D) *Since we needed line numbers to facilitate editing (this was before WYSIWYG screen editors), we used these line numbers as GOTO targets.*
- E) *The word LET came from our mathematics background, because we are always saying things like: "Let G be an Abelian Group." Furthermore, LET was required for assignment statements so that all statements started with a keyword. This made understanding things like $X = Y + Z$ easier for nonscientists. For logical expressions, we used $IF X = Y + Z THEN...$ while for assignments we used $LET X = Y + Z$.*
- F) *No nonobvious punctuation, like the ALGOL semicolon to terminate statements, or the mysterious FORTRAN $IF (A) 100, 200, 300$. There is no comma allowed after the (A). Along the same line; we used ordinary-sounding English words for the commands to the time-sharing system. HELLO rather than LOGON, GOODBYE rather than LOGOFF, NEW to create a new program, OLD to retrieve an existing program, and so on.*

Interestingly enough, BASIC didn't support the INPUT statement until its third version, meaning that for the first two years it wasn't actually an interactive language—obviously making game creation rather difficult!

BASIC TIME SHARING

» The other major factor in BASIC's widespread adoption was the development of time sharing. With a batch computing system, it could take hours, or even days, for a program's results to arrive. Less-intuitive languages were easier to make mistakes with, and those

mistakes would waste precious computer access. Time sharing allowed several people to work on one computer simultaneously, maximizing computer time.

Kurtz described the first time-sharing operating system at Dartmouth:

Two Dartmouth students, Mike Busch and John McGeachie, developed a time-sharing operating system for the combination General Electric GE-235 computer and the GE Datanet-30 front-end communications controller. This was done simultaneously with the development of BASIC and, in many ways, was more fundamental to our goals at Dartmouth than BASIC itself. GE immediately took what we had done and developed a nationwide time-sharing network. They used regionally located centers, as the cost of long-distance communication at that time was too expensive. It was in one of these centers in Seattle that Bill Gates was first introduced to computing and BASIC. My best guess is that he was about 13 years old at the time. ■■

Kurtz and the Dartmouth team continued to refine BASIC, adding INPUT, RANDOMIZE, and ON-GOTO statements among many other revisions, until it hit its sixth edition in 1971, which remained largely unchanged for several years. Since BASIC was included with General Electric's time-sharing systems, it continued to spread until it became the most widely used programming language in the world. According to Kurtz, "At one point, around 1974, when ANSI initiated a standards effort for BASIC, it was used on more time-sharing networks, of which there were over a hundred at the time, than any other language—my recollection is more than 80."

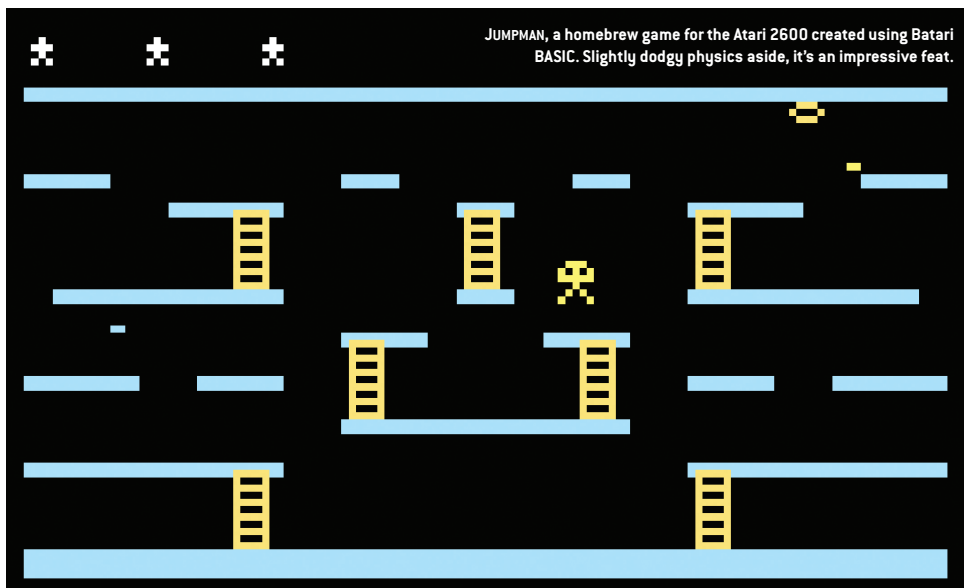
It was around the time of the sixth edition that David Ahl, author of the influential book *101 BASIC Computer Games*, was introduced to BASIC. "I accepted a position at Digital Equipment Corporation in late 1969 in the PDP-8 Group and soon got involved in marketing minicomputers to schools and colleges," he said. "At the time, the only high-level language on DEC's smallest computers was FOCAL. However, because of the pioneering work at Dartmouth by John Kemeny and Tom Kurtz in developing BASIC and related educational applications, most colleges and high schools wanted a computer that spoke BASIC."

Unlike other programming languages, BASIC was completely unregulated, which Kurtz believes was crucial to its success.

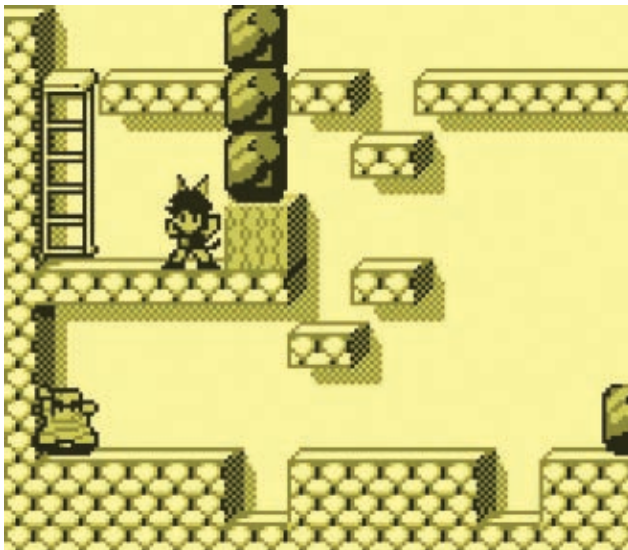
"Dartmouth always took the point of view that what we did was public, and thus we never made any attempt to control or direct the results of our research," he said. Ahl agreed, adding, "DEC's FOCAL language was equal to BASIC in most aspects and even better in some, but it had one huge drawback: DEC was unwilling to license it to other computer manufacturers. FOCAL was fighting an uphill battle against BASIC, which was available on GE, Honeywell, HP, and other computers."

GAMES: BASIC'S KILLER APP

» The early 1970s saw BASIC move away from mainframes and time-sharing systems and to the burgeoning microcomputers of the day. Ahl recalled how quickly they sprang up: "DEC took the unusual step of contracting with outside vendors to develop four versions of BASIC, one for a 4K stand-alone PDP-8 (EduSystem 10), one for a small multiuser 8K PDP-8 (EduSystem 20), one for the Timeshared-8 (EduSystem 50), and one for a batch processing PDP-8 with a card reader (EduSystem 30). The most limited version was that for the 4K PDP-8 as the BASIC



JUMP MAN, a homebrew game for the Atari 2600 created using Batari BASIC. Slightly dodgy physics aside, it's an impressive feat.



interpreter used 3.5K of memory, so programs had to fit in the remaining 500 12-bit bytes.”

To Ahl, games were critical to BASIC’s spread. “We needed to demonstrate that such a limited configuration could run real programs, so I started converting FOCAL demo programs to this low-end BASIC,” he explained. “We also encouraged users who wrote programs, especially games, to submit them. In 1972 I collected together a bunch of games that I had written from scratch or converted from FOCAL along with submissions from users and published it as *101 BASIC Computer Games*. The book was later revised and, in 1983, total sales topped 1 million, and it became the first million-selling computer book.”

As the book included type-in listings for BASIC games, it was likely a major catalyst in BASIC’s proliferation. A large base of games would appeal to anyone considering incorporating a language into a new piece of hardware—like Gates and Altair BASIC, for example. Ahl recalls meeting Gates: “When I left DEC in July 1974 and started *Creative Computing* magazine...I used all my vacation time to attend computer shows and conferences in order to promote *Creative Computing*. One fascinating show was the Altair Conference in Albuquerque in 1975 where I met Bill Gates and learned of his intention to write a BASIC language for the Altair.” Sure enough, in 1975, MITS released Altair BASIC, which was developed by Gates and Paul Allen as the newly founded “Micro-Soft.” It was an interpreter, as opposed to Dartmouth and GE’s implementation of a BASIC compiler, but it suited the limitations of the Altair hardware.

Despite his influence, Ahl is reluctant to take credit for BASIC’s hobbyist popularity: “Did my book popularize the spread of BASIC? Maybe, but the great libraries of BASIC programs at Dartmouth, the

Minnesota Educational Computer Consortium, and Huntington Computer Project along with books like Bob Albrecht’s *What To Do After You Hit Return* were equally influential. BASIC was the only widely available interactive language. Smalltalk? Lisp? Logo? MUMPS? They didn’t stand a chance against BASIC.”

BASIC continued to spread as Microsoft ported its version to new microcomputer platforms and other people developed their own BASIC variant. Apple cofounder Steve Wozniak recalls writing Integer BASIC for the first Apple computers:

I didn’t program in BASIC, but I sniffed the wind. There was a book out, *101 BASIC Computer Games*, and Bill Gates had written a BASIC. So I said: You’ve got to have these games, it’s going to be the heart of a computer that’s worth anything to people.... The microprocessor I used only had machine language. I’d never had any training in writing a computer language, so I pulled out a manual at work, which was Hewlett-Packard, and read through making notes of all the commands. I didn’t realize their BASIC was very different [from] Digital Equipment’s BASIC, which the 101 Games had been written for. They were very different regarding strings.

Technically, Wozniak was probably the only one to hand write a new version of BASIC, explaining:

I could not afford a time-share assembler. That’s where you take a terminal, call a local phone number, connect to some faraway computer, and type your programs in using machine language. So I wrote my programs

Top: Family BASIC for Nintendo’s Famicom isn’t well documented in English, but you can have fun drawing custom backgrounds! Middle: CATRAP on the Game Boy started life as a BASIC type-in listing for the Sharp MZ-700 computer in 1985. The source code is online. Bottom: QUINTY came about after Satoshi Tajiri tinkered with Family BASIC to understand how the Famicom functioned.

in machine language on paper, and then I looked at little charts to figure out what ones and zeroes would be created by them. I would write the ones and zeros down in base-16, or hexadecimal, and type that into the computer I had built. It would take me 40 minutes to type my entire BASIC in. If you go back to Bill Gates, he used a computer to type the program in. I'm sure I must be the only one who wrote a language completely in my own handwriting. 📄

BASIC AND THE NASCENT GAME INDUSTRY

» After that first breakaway in the mid-1970s, BASIC variants continued to propagate across all hardware formats, and as BASIC spread internationally, it would be localized for each new country it spread to. This formed a few distinct microcosms.

In the United Kingdom, microcomputers became wildly popular. Machines such as the ZX Spectrum, Amstrad CPC, and Commodore 64 all had one or more variants of BASIC interpreter. With one of these affordable, comparatively easy-to-learn machines, a teenager in the United Kingdom could write a game in his spare time using BASIC, have a parent sign a publishing contract, and have the game be distributed on audio cassette across the country. Two examples are GHOST TOWN and ZIGGURAT by John Pickford, both made while he was still at school (the first netting him £500). From this humble start on the ZX Spectrum, he would go on to develop NES games for Rare under contract. And thanks to the BBC's Computer Literacy Project, almost every British classroom had a BBC Micro—complete with BASIC.

Japan adopted BASIC dialects on various home computers, often ported by Microsoft. These computers included NEC models such as the PC-6001 and PC-88, as well as the MSX and X68000 formats. Some of Japan's most prominent developers learned to program using these BASIC variants, including DRAGON

QUEST creator Yuji Horii. He first encountered BASIC on his PC-6001, reading books on the language and modifying the games he had. Using BASIC he would program Japan's first adventure game, the astoundingly successful and genre-defining PORTOPIA RENZOKU SATSUJIN JIKEN, which subsequently influenced an entire generation of Japanese players and developers.

Even Nintendo created a Famicom BASIC variant (called Family Basic) in June 1984, which POKÉMON creator Satoshi Tajiri of Game Freak used to develop his understanding of the Famicom. In an interview with the Tokyo Metropolitan Museum of Photography, he described the creation of his first Famicom/NES title, Quinty (aka Mendel Palace):

📄 It became possible to see what was actually going on inside the Famicom, when software for beginners called "Family Basic" was released. When I completely understood its mechanism, I went to Akihabara to buy a multiuse circuit board, added the terminals from my Famicom, and ran my programs over it. That was our first step. Then I made a long-lasting battery, to save the memory on the circuit board. It was all a handmade developing environment. 📄

But how important were games to the BASIC founders? Very important, as it turns out. Kurtz explains:

📄 The early personal computers were primarily game machines for hobbyists. So, their versions of BASIC had instructions for dealing with joysticks, and so on. Remember, at the time that Gates [and others] did their things, BASIC was the most widely used programming language in the world. The microcomputer folks adopted it for their teeny machines, which at first

were good only for game playing. It was ideal for hobbyists who could not afford to pay for time-sharing services. Serious business users couldn't imagine giving up their big machine computers to put their applications on these "toys." It's clear that Microsoft did see the potential and managed to benefit from it, but most of the other small personal computer companies did not. 📄

Even now, you'll find the roots of BASIC in a lot of popular or significant games. Richard Garriott would use Applesoft BASIC when programming AKALABETH, and again when developing the first ULTIMA. WIZARDRY was started (though not completed) in BASIC. Henk Rogers' THE BLACK ONYX, a major influence on early Japanese designers, was partially programmed in BASIC. PITMAN, also in Japan, started off as a magazine type-in before it was ported to the Game Boy as CATRAP. Julian Gollop, famous for the X-COM series of strategy games, started out programming in BASIC on the ZX81 and then ZX Spectrum micros. One of his earliest BASIC games was REBELSTAR RAIDERS on the ZX Spectrum, which itself spawned several sequels right up to 2005.

IS BASIC STILL RELEVANT?

» BASIC's significance is fading as it nears its 50th anniversary. Anyone interested in game development can use utilities like Game Maker, which don't require any programming knowledge, while more ambitious developers these days target popular platforms with better earning potential, such as iOS. Even if you want to use BASIC, where do you start? Microsoft no longer includes QBasic with Windows, and the older DOS-based versions are incompatible with modern Windows versions.

BASIC isn't popular in academia anymore, either. "Two things happened over the years," Kurtz said. "First, many of the applications used to demonstrate the use of simple BASIC programs could be handled, for example, by spreadsheets. Second,



Yuji Horii's BASIC source code for PORTOPIA RENZOKU SATSUJIN JIKEN, Japan's first adventure game from 1983. It would later influence Hideo Kojima's SNATCHER.

the computer science profession insisted on introducing ridiculously complicated languages, including those that went with the secondary school computer science curriculum."

Wozniak feels it's still useful for beginners. "It was really a great language with a great purpose," he said. "People have to start somewhere, and I think that's the best starting language you could ever have. ...Newcomers learn things a lot better when they understand what every statement does. It's a much better starting point."

Ah! feels optimistic, saying, "Will BASIC go on? Yes, but there's a danger in saying that it will last forever like the alphabet or wheel or pliers. Nevertheless, the BASIC language has had a profound influence for 50 years, and Kemeny and Kurtz deserve a great round of applause from all of us...for the last five decades, it's been one of the towering achievements in technology and education."

Where does BASIC go from here? Maybe only BASIC itself can answer that:

GOTO 10
RUN



JOHN SZCZEPANIAK is a freelance journalist and novelist, with a penchant for yachting and obscure video games. He writes for Hardcore Gaming 101, GamesTM, Retro Gamer, and several other publications.

Gateway to Asia

Spreading Smiles through GAMES

TOKYO GAME SHOW 2012

9 / 20 21
[THU] [FRI]

BUSINESS DAY

22 23
[SAT] [SUN]

PUBLIC DAY

Venue: **Makuhari Messe** (Chiba, Japan)

■ **Business Days: 10:00-17:00, September 20(Thu) & 21(Fri), 2012**

Admission Fees: Business Days -Ticket sold in advance: 5,000 yen

Online Ticket Service is available at our official website

■ **Public Days: 10:00-17:00, September 22(Sat) & 23(Sun), 2012**

Public Days -Advance Ticket/1,000 yen per day, On-the-day Ticket 1,200 yen per day.

Online Ticket Service is available at our official website. Children (elementary school age and under): free of charge

■ **Official Tour:**

Official Tour Packages departing from North America is available.

Please access to our official website for more information.

TOKYO GAME SHOW
Organizer: Computer Entertainment Supplier's Association (CESA)

Co-Organizer: Nikkei Business Publications, Inc.
Supporter: Ministry of Economy, Trade and Industry

Official Website:
<http://tgs.cesa.or.jp/english/>

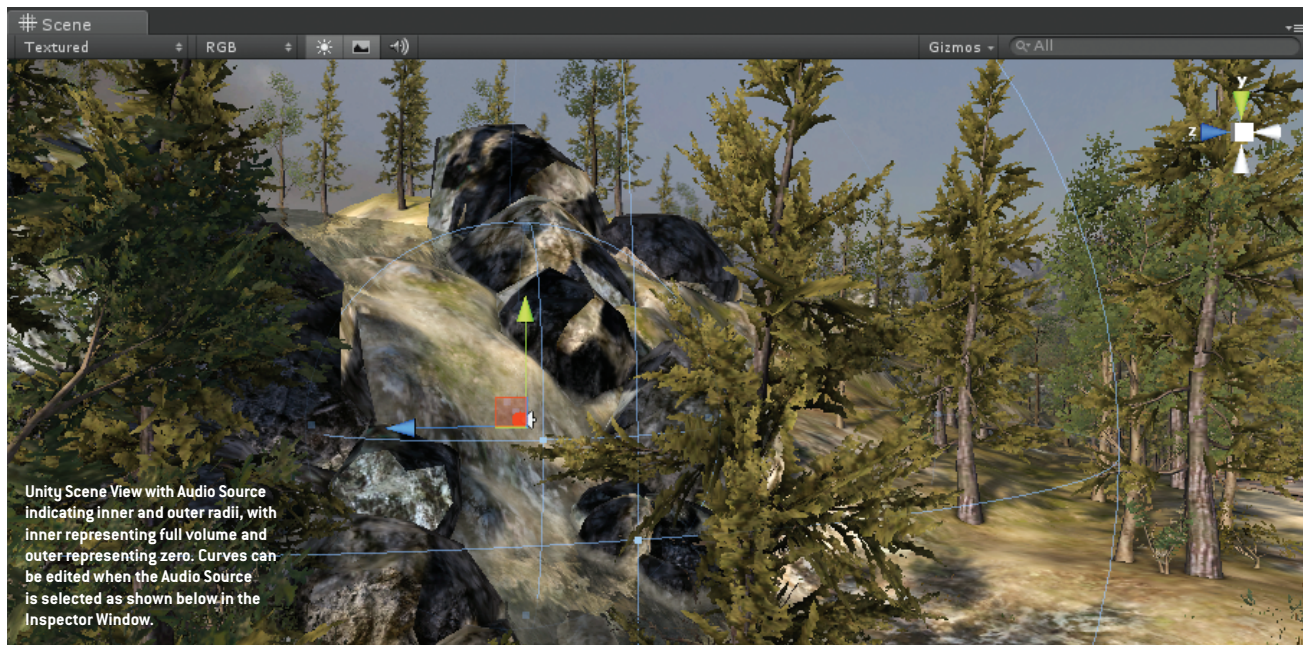
 **CESA** 日経BP社
Japan Business Publications



AUDIO IN UNITY 101

WELCOME TO UNITY SOUND DESIGN AND SCRIPTING

Game audio designers want control. In the film industry, you can easily drop audio into a film and see exactly how it will play out, but in games it's not so easy. But things have been getting better (we don't have to integrate with text files any more), thanks in part to better audio tools in commercial game engines like Unity. Unity is getting increasingly popular among game developers across all ranges of expertise, so I'm going to walk through a few of Unity's basic audio concepts, both in the GUI and on the scripting side, with tips you can't get from reading the manual.



STARTING WITH THE UNITY GUI

» Unity has limited audio capabilities out of the box when compared to seasoned engines (such as Unreal) or audio-specific middleware such as AudioKinetic's Wwise, Firelight Technologies's FMOD, or RAD Game Tools's Miles Sound System. Those audio-specific tools have been around for years, and they offer far more GUI-based integration than Unity, which mostly limits you to placing sounds in the world and assigning sounds to animations. Let's take a look at the basics of what the engine can do.

Your Unity work will start in the Scene Editor, where you can create Audio Clips (the sounds themselves, such as .wav files), Audio Sources (audio objects with properties such as 2D or 3D, and any additional functionality you may wish to script, such as randomization and arrays of Audio Clips), reverb zones, distance attenuation curves, low-pass filter curves and spread curves for your Audio Sources, and ambient sounds. You can also specify a Listener, which is where the sound gets picked

up in the game (usually where the player's view is, attached to a first-person camera controller).

You can also attach an Audio Clip to an animation frame within Unity's Animation window; just hit Control + 6 to bring up the Animation window, then click on a character with animations. This will allow you to select the animations and play them. From here, just click on the vertical line icon to add an Event and map it to play a sound.

These are all basic methods of inserting your audio into a 2D or 3D game scene, but it's enough to let you create a full and rich environment of ambient sounds, and have the sounds that play within that environment behave realistically, with appropriate reverb. (Got a bathroom in your world? Yep, there's a Bathroom reverb setting.) But to dig much deeper with Unity audio, you'll need to start scripting.

BETTER LIVING THROUGH SCRIPTING

» All advanced audio tasks in Unity are scripted, usually in JavaScript or C#, and it can be complex, confusing, and time consuming to

an audio professional without at least a year of coding or scripting training. Scripting lets you be very flexible, but it also doesn't give you one standard way to integrate your audio into the game, meaning each project may need a unique approach.

With Unity and a decent scripting background, you can sort of brew your own "do-it-yourself" graphical user interface approach. Unity translates script concepts such as variables and arrays into GUI elements instantly—which is the first time we've seen this in a commercial game development tool. Previously, there was an entire code layer that stood between scripting and the ability to create GUI elements such as buttons or drop-down menus, but most creative types aren't also programmers, and prefer a simpler interface.

This system opens up a world of options for audio designers who don't want to wait for a programmer. If you take the time to learn the script (which is detailed in Unity's reference documentation), you can learn how to create your own audio functionality and reuse it by



LISTING 1

```
#pragma strict

class MaterialImpact { #PhysicMaterial is like a texture with properties, and within the Material there
will be the means to detect footsteps and bullet impacts, each represented as a variable. When "var" is
used, it generates an editable GUI object in Unity, with values that can change outside the script.
    var physicMaterial : PhysicMaterial;
    var playerFootstepSounds : AudioClip[];
    var mechFootstepSounds : AudioClip[];
    var spiderFootstepSounds : AudioClip[];
    var bulletHitSounds : AudioClip[];
}

class MaterialImpactManager extends MonoBehaviour {
    var materials : MaterialImpact[];

    private static var dict : System.Collections.Generic.Dictionary.<PhysicMaterial, MaterialImpact>;
    private static var defaultMat : MaterialImpact;

    function Awake () {
        defaultMat = materials[0];

        dict = new System.Collections.Generic.Dictionary.<PhysicMaterial, MaterialImpact> ();
        for (var i : int = 0; i < materials.Length; i++) {
            dict.Add (materials[i].physicMaterial, materials[i]);
        }
    }

    static function GetPlayerFootstepSound (mat : PhysicMaterial) : AudioClip {
        var imp : MaterialImpact = GetMaterialImpact (mat);
        return GetRandomSoundFromArray(imp.playerFootstepSounds);
    }

    static function GetMechFootstepSound (mat : PhysicMaterial) : AudioClip {
        var imp : MaterialImpact = GetMaterialImpact (mat);
        return GetRandomSoundFromArray(imp.mechFootstepSounds);
    }

    static function GetSpiderFootstepSound (mat : PhysicMaterial) : AudioClip {
        var imp : MaterialImpact = GetMaterialImpact (mat);
        return GetRandomSoundFromArray(imp.spiderFootstepSounds);
    }

    static function GetBulletHitSound (mat : PhysicMaterial) : AudioClip {
        var imp : MaterialImpact = GetMaterialImpact (mat);
        return GetRandomSoundFromArray(imp.bulletHitSounds);
    }

    static function GetMaterialImpact (mat : PhysicMaterial) : MaterialImpact {
        if (mat && dict.ContainsKey (mat))
            return dict[mat];
        return defaultMat;
    }

    static function GetRandomSoundFromArray (audioClipArray : AudioClip[]) : AudioClip {
        if (audioClipArray.Length > 0)
            return audioClipArray[Random.Range (0, audioClipArray.Length)];
        return null;
    }
}
```

than code. But it's best to understand scripting basics before jumping into these plug-ins, since you will know how nodes work together already.

Each Unity game may be set up differently, which means you need to be well prepared for what might come your way. For example, the code and GUI that control player footsteps may be in JavaScript or in C#. Therefore, as an audio engineer approaching a project for the first time with no audio scripts yet, you should be prepared for either one. ANGRY BOTS, the demo game that is included with Unity, is a good example of JavaScript implementation. The only issue is that its audio scripts are all over the place, embedded within each game rather than clearly delineated from other game elements (footstep sound code is often in the same script file as the entire footstep code routine). This creates a workflow issue where backtracking through the project to find connectivity becomes problematic and time consuming, since you don't know what the audio scripts are, or where they are within the Project Window.

You can make sure this doesn't happen by keeping all your audio functionality and scripts inside an easy-to-find folder. The Project window contains all of your project-based materials, and when they get dragged into a level they show up in the Hierarchy Window. You can organize your Project window to include as many folders and subfolders as you want; I like to have an Audio folder for my .wav/.mp3/.ogg files, a Prefabs folder containing Audio Sources and the raw audio files, and a Scripts folder containing my audio functions, since it's easy to add them to game components from there.

One more note before we start the scripting: If you're just starting to learn scripting, try to focus on C#. JavaScript isn't as optimized or fast as C#, and it has a few debugging issues. Also, if you learn C# well enough, it's not too difficult to switch to JavaScript if you need to. Now on to the meat and potatoes.

SCRIPTING FOOTSTEPS AND FADING

» Before we jump into the sound scripting, you should know about two key concepts: objects and prefabs. Objects are anything from an audio file to an animation. Prefabs are groupings of objects such as a character (mesh, animation, texture, scripts) all wrapped up in one package. A typical project, viewed from the Project window in the Scene Editor, will consist of a Scene folder (can be the entire game or a single level), a Source folder with code and script, a Content folder with all necessary objects, and a Prefab folder where game-specific object groupings are kept and dragged into your scene file.

Now, let's take a look at the JavaScript code that handles footsteps for ANGRY BOTS (see Listing 1). Note that footsteps (metal) outside are inaudible. Each character has a footstep handler attached to it. You specify the FootType (character) and Audio Source.

dragging scripts into your scene editor for functionality that controls anything you need (volume, hooking sounds to particles, footsteps, and so on). It's pretty handy!

Fortunately, Unity's scripting languages are less complex than full-on native code like C++, and eventually someone could even create a set of ready-to-go audio scripts with instructions for the not-so-technically-inclined audio pros to instantly hear their results. Right now, though,

there's no way around some programming without plug-ins. You'll need to plug in somehow to the prefabs, functions, and variables of the rest of the game, similar to how UnrealEd's Kismet script editor uses Remote Events to stub audio connectivity to its game scripts and controllers. Note that Unity does have a few Kismet-style plug-ins, such as Antares VIZIO and Detox Studios uScript, which let designers work with a visual flowchart type of scripting rather

resources

UNITY TUTORIALS

<http://unity3d.com/support/resources/tutorials>
The official Unity tutorials give you a step-by-step explanation of how a few Unity games are made from beginning to end.

UNITY ASSET STORE

From within the Unity Scene Editor, go to the Window menu and choose Asset Store, and you'll find a whole bunch of music and sound assets, editor extensions, and custom scripts, all of which you can easily add to your game.

UNITY SCRIPT REFERENCE

<http://unity3d.com/support/documentation/ScriptReference/index.html> Unity has an official script reference that will tell you everything, but it is incredibly confusing unless you're already an advanced scripter.

UNITY ANSWERS

<http://answers.unity3d.com/index.html>
Unity Answers lets you ask fellow Unity devs questions and get fairly quick answers (usually within 24 hours).

LISTING 2

```
using UnityEngine;
using System.Collections;

public class AudioSourceTimeBased : TimeBased {

    public float fadeInRate = 1f;
    public float fadeOutRate = 1f;

    private AudioSource _target;
    private float _originalVolume;
    private float _volumeTarget;
    private bool _shouldPlay = false;

    public bool introOff = false;
    public void Awake()
    {
        _target = GetComponent<AudioSource>();
        _originalVolume = _target.volume;

        _volumeTarget = 0f;
        _target.volume = 0f;
    }

    public void Update()
    {
        if ( LevelInfo.Instance.CameraSettings.intro.playing && introOff ) return;

        if ( _shouldPlay )
        {
            if ( LevelInfo.Environment.time > timeOn )
            {
                _target.volume = Mathf.Lerp(_target.volume, _volumeTarget, Time.deltaTime *
fadeInRate );
            }
            else if ( LevelInfo.Environment.time > timeOff && LevelInfo.Environment.time < timeOn )
            {
                _target.volume = Mathf.Lerp(_target.volume, _volumeTarget, Time.deltaTime *
fadeOutRate );
            }

            if ( _target.volume <= 0.0001f )
            {
                _target.Stop();
                _shouldPlay = false;
            }
        }
    }

    public void FixedUpdate()
    {
        if ( LevelInfo.Environment.time > timeOn && !_shouldPlay )
        {
            _shouldPlay = true;
            _target.enabled = true;
            _volumeTarget = _originalVolume;
            _target.Play();
        }
        else if ( LevelInfo.Environment.time > timeOff && LevelInfo.Environment.time < timeOn &&
_shouldPlay )
        {
            _volumeTarget = 0f;
        }
    }
}
```

As you can see, this footsteps script is complex enough to support varied materials.

Bear in mind there are numerous ways to hook up footsteps. Unity features a splatmap feature for textures (also known as Terrain Textures) that lets you assign properties to textures, so they may be treated in a similar way to Materials in the Unreal Engine. In a demo created for GDC 2012 called "Initium," I collaborated with dotBunny (a Canadian game development studio run by Matthew Davey) and went a simpler but more work-intensive route by setting up zones that triggered footstep sets when the player would cross them.

Let's try another example. With a default Unity installation, you can't fade in and out from zero to max volume when using a 2D sound. For example, let's say we want a music track to fade out when we've triggered another piece of music. With Wwise and FMOD, you can do this natively, but in the meantime if you need to fade you can use this simple C# script (see [Listing 2](#)) that enables a piece of music to fade in or out at a specific time of day. Just plug it into an Audio Source, and it should provide GUI functionality to enter the time of day and length of fade from 0-1.

AUDIO DESIGNERS, UNITE!

» That's all we're going to cover in this article, but don't let that stop you from learning more. Unity has a huge online presence and support system that is somewhat fragmented between official and community channels, so here are some sites that should help you figure out how to solve your Unity audio problems and find more assets and scripts you can use in your games. 🎧

ALEXANDER BRANDON is a composer, sound designer, voice actor, producer, and director for games with 18 years of experience in the video game industry. He is currently president of Funky Rustic, an audio outsourcing group in Georgetown, Texas.



WHEN DIGITAL MEETS PHYSICAL

BOARD GAMES AND VIDEO GAMES: BETTER TOGETHER

The line between video games and board games is blurring. Consider the recent mobile games *CABALS* and *HERO ACADEMY*; both contain the trappings of board games (turn-based play, a shuffled deck of game pieces, a visible board divided into tiles, and transparent rules with no hidden modifiers), even though these games exist only in digital form. Mainstream video games are also starting to include select board game elements, such as the “Rage Frenzy” collectible card mechanic in *RAGE*. Essentially, designers are discovering that familiar board game conventions (cards and dice, for example) can be just as useful as any video game convention when helping players feel comfortable with the design.

But the connection between board games and video games is not a one-way street; board games are changing, too. More specifically, the iPad is revolutionizing the board game industry as digital translations of physical games are finally viable. The iPad’s features—a large, high-resolution screen, a touch-based interface, and (perhaps most importantly) a robust infrastructure for selling digital apps—are the perfect combination for digital board games. Eric Hautemont, the founder and CEO of the board game publisher

Days of Wonder (*TICKET TO RIDE*, *SMALL WORLD*), expressed his enthusiasm for the device:

“The beauty of the iPad is that you could forget about it. Meaning that when you put an iPad between two players, the screen is so well done that you almost forget there are electronics behind that. When you sit down to play *SMALL WORLD* on the iPad, you stop thinking about it as an iPad game and just think of it as *SMALL WORLD*. In the future, the question of whether something is a ‘board game’ or an ‘iPad app’ or whatever it will be

in the future becomes a meaningless question.”

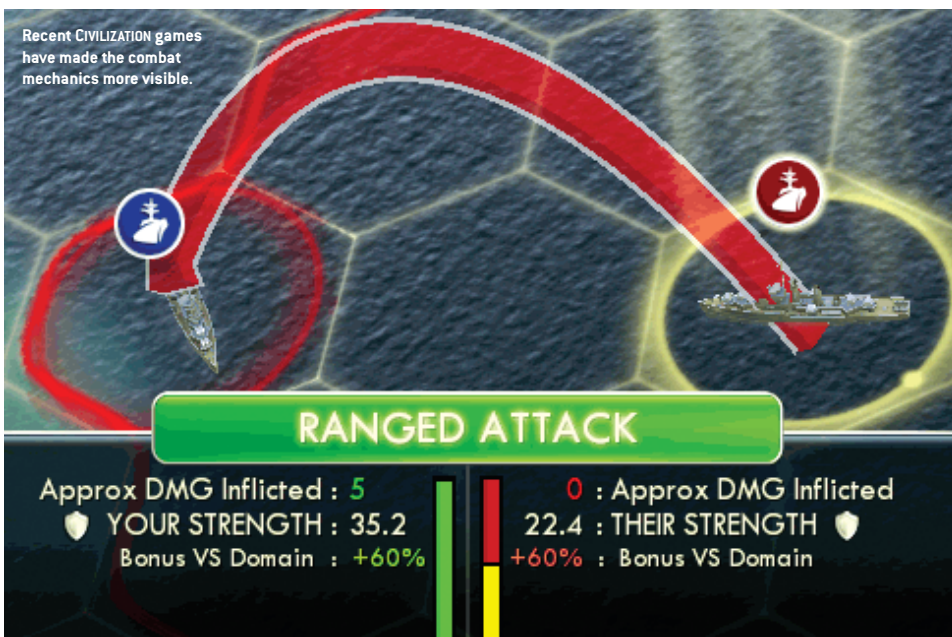
Days of Wonder’s business experienced a significant bump from mobile. Since the release of *TICKET TO RIDE POCKET* on the iPhone, the boxed version of the game began selling more copies, by a sustained increase of 70 percent. Meanwhile, the iPad version is consistently a top-100 app, selling for a healthy \$6.99. (One sign of the healthy iOS market for board games is how well they have maintained a high price point in a sea of 99-cent games; *CATAN* and

SAMURAI both sell for \$4.99, while *CARCASSONNE* still costs a whopping \$9.99 two years after release!) Indeed, since release, the digital versions of *TICKET TO RIDE* have outsold the physical one by 3-to-1—perhaps we ought to call Days of Wonder a video game company instead of a board game one.

TRANSPARENT GAMES

» As board games become increasingly digital, we are discovering that the essence of board games might not be their physical nature, but instead a set of shared design elements, foremost of which is their absolute transparency: the idea that all a game’s rules should be visible. You can’t make a board game mechanic or process ambiguous, nor can you hide them from the players, because you need the players to actually execute those mechanics (instead of just showing them the output of a CPU’s calculations). Designers should understand the role it plays in making board games fun—and how they can be integrated into a video game as well.

For example, the *CIVILIZATION* series is essentially a giant board game that can be played only with a computer to handle all the calculations and record-keeping. The majority of game mechanics are clearly transparent to the player, from how much food a city produces during each turn to how much time is needed to discover the next technology. The combat system, however, was originally not quite so transparent, so earlier *CIVILIZATION* players worried that a tank could lose to a spearman under the wrong circumstances. *CIVILIZATION IV* took steps to fix this problem by providing players with the exact probability of success for



each possible battle. CIVILIZATION V went even further, with a detailed graphical widget to show the estimated damage.

The combat systems of these games were still opaque to the average player (the hardcore players ended up reverse-engineering the formulas, of course), but these features still honored the ideal of transparency by making the results of combat clear; the designers understood that transparency was an important virtue for the series, and the changes were well received by the fans.

WHEN DIGITAL BEATS PHYSICAL

» One of the most exciting aspects of the digital-physical union is that some board games are greatly improved in the transition to video game. First, digital board games require no set-up time or record-keeping, which means that games can be played much faster and in new environments; suddenly, you can play MEMOIR '44 in a coffee shop without worrying about whether your little army men are scaring away the other customers.

Being able to play a digital board game tens (or even hundreds) of times transforms the experience. One person will probably play a heavy, card-driven historical simulation game like 1960 only a handful of times in person, since a typical session takes at least four hours to finish—but with the web version you can finish a game in an hour. Making games that last a shorter duration means people can play them more frequently, which means losses sting less and players are more free to experiment with new strategies without fearing they are blowing their one chance to play the game that month.

However, the challenge of such frequent play is that imbalances are found much quicker than ever before. A FEW ACRES OF SNOW, Martin Wallace's 2011 war game on the French-Indian War, gained some notoriety for needing a quick patch to deal with a dominant strategy for the English. This strategy, known as the Halifax Hammer, emerged soon after release because the game was playable for free on the web; water



found a crack that much sooner.

Another advantage of digital board games is asynchronous play; it's hard to find a long, uninterrupted block of time where you and your fellow board gamers can get together and play in the same place. Asynchronous play circumvents this issue by letting people run games at their own pace; the program simply waits for the next player to make her move.

One iOS game, ASCENSION, owes much of its success to getting this format right. As a card game, ASCENSION is merely a competent variant to the seminal deck-building card game DOMINION. On iOS, however, ASCENSION was very successful because the developers focused on asynchronous play as a core feature of the game, enabling players to easily manage multiple concurrent games. Not every board game is ideal for asynchronous play (each turn needs to feature a significant number of decisions), but ones that are should find new life on mobile devices.

ANALYTICAL FUN

» Whether played asynchronously or in single-player, digital

translations can eliminate the waiting time associated with meaty board games. Certain types of Eurogames with little randomness and no hidden information (CAYLUS and PUERTO RICO, for example) are painful to play with "optimizers"—people who are unafraid to slow the game down to a crawl to ensure they make just the right decision. As people wait for the "optimizers" to take their turns, they often conclude that optimizing itself is not fun, but this is mitigated when you're playing asynchronously. Optimization while under social pressure to finish faster may not be fun, but finding just the right move to handle a tricky situation is exactly why these types of games are so rewarding. For a multiplayer game, we'd call this phenomenon "analysis paralysis," but for a single-player game we'd call that "intense engagement!" The problem with playing in person is not wanting to slow down the game, while also fearing that rushing will lead to the wrong move.

Both asynchronous and single-player versions of board games solve this problem by giving the player all the time he needs

to perfect his plan. Indeed, PUERTO RICO comes alive on the iPad, shining as a tight, elegant game that can move at a comfortable speed when a single person gets to make all the decisions at her own speed. Indeed, the popularity of cooperative board games in recent years, such as PANDEMIC and GHOST STORIES, suggests a healthy market for solitaire video games with a board game soul.

This revelation underscores the value of decoupling the physical characteristics of board games from their defining feature—absolute transparency. The lesson for all designers is that transparency can be a virtue in almost any genre or format. Consider the natural tile-matching patterns in TRIPLE TOWN, or the predictable enemy behaviors in PLANTS VS. ZOMBIES, or the simple physical elements in CUT THE ROPE. These games don't appear to be board games, but they all share the virtue of transparency. 🎮

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.



GAME DEVELOPERS CONFERENCE™ CHINA

SHANGHAI, CHINA
SHANGHAI INTERNATIONAL CONVENTION CENTER
NOVEMBER 17-19, 2012

2012

www.GDCCHINA.com





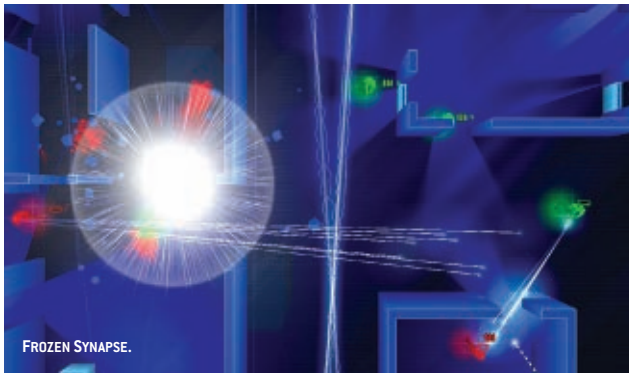
IN DEFENSE OF PAYING ONCE

WHY RISING INDIES CAN'T BE IGNORED

In an era where free-to-play is rapidly establishing itself as the dominant business model, is “pay-once” an ungainly lumbering dinosaur best thought of as an amusing historical footnote?

The data is there for all to see. Industry commentator Nicholas Lovell of Gamesbrief says, for example, “On a 10-year view, I don’t believe it will be possible to charge for basic access to content at all.”

I’m certainly not a free-to-play naysayer. It’s clear that F2P offers many significant advantages, such as access to a much wider player-base, the opportunity for essentially uncapped spending within your game, and the ability to reward players for positive behavior. These are all things that any game developer should care about. Recent successes like *TRIBES: ASCEND* and *LEAGUE OF LEGENDS* have shown that F2P can work for a core audience as well as a casual one.



FROZEN SYNAPSE.

I personally believe that we’ll see many more traditional mainstream triple-A IPs switching to free-to-play within the next two years. The bottom line—both literal and metaphorical—is that an F2P game is capable of making significantly more money than a pay-once title.

However, as large developers and smaller indies alike rush into the bright free-to-play future, it’s worth remembering that traditional pay-once games can still turn a healthy profit.

A DIFFERENT KIND OF FREEDOM

Pay-once offers one clear, unambiguous advantage: There are no design constraints.

While F2P is certainly expanding its repertoire in terms of genre, pay-once knows no bounds. Some of the most innovative games being created right now require only one payment: Who would have thought that an experimental title like thechineseroom’s *DEAR ESTHER* would be profitable for its creators within the first five hours of sales?

When you’re free from the requirement to constantly direct players to their wallets, you can allow them to relax and enjoy the scenery. This is an experience that will never go out of fashion. People will always be willing to pay for escapism.

Similarly, the F2P business model generally presumes that a

game is inherently modular: items, character slots, aesthetic options, and all kinds of other features can be plugged into the core game at will. While this is great for some titles, it’s impossible for others (think about simple-yet-beautiful experiences such as *LIMBO*), and not all players are interested.

A high number of continuous active users is vital for an F2P title. Player retention is all-important, so skill curves and progression have to be meticulously designed and metricated. However, there is nothing to say that a game has to constantly retain players to be considered a creative success—there are many highly regarded games that players enjoy for a few weeks, leave, and then pick up again later. It is doubtful that those designs would be suitable for F2P monetization.

THE AESTHETIC DEBATE

What about F2P games where payments have no effect on gameplay? Traditionally, purely aesthetic items have always performed much worse than their gameplay-modifying counterparts. David Ederly recently acknowledged this when he indicated that cosmetic items in *REALM OF THE MAD GOD* comprise only 17% of the game’s total revenue.

This has implications for certain types of competitive games where paid “crossgrades” (as in *TRIBES*) are not an option. Valve’s forthcoming *DOTA2* will be monetized entirely with aesthetic items and will prove to be a deeply interesting test case. To my mind, this is an area where F2P has yet to fully prove itself to a broadly Western, “hardcore” audience.

THE FUTURE IS FREE

Does the inevitable dominance of F2P mean the death of pay-once? Given the rapid increase in the rate of smaller indie games being produced in the last three years, and the very recent emergence of a variety of new price points for such games, such a proposition seems ludicrous.

F2P proponents are often keen to write off titles with what they see as insignificantly low revenues, but this means that

they also write off an emerging horde of new indie developers. With many talented people leaving the mainstream industry, and many younger developers taking advantage of the vast resources which have only recently become available, this sector of the industry is booming.

Valve in particular is brilliantly placed to take advantage of this situation, thanks to the market share of Steam, and the company’s willingness to include indie titles that don’t earn so much. While the promise of a living wage (or much better) awaits many independent creators who manage to acquire Steam distribution for a pay-once game, the model will continue to thrive.

Our tactical game *FROZEN SYNAPSE* transformed *Mode 7* from a struggling start-up to a fully fledged microstudio. Its gameplay, aesthetic, and feature-set all pointed toward pay-once for us, and we’re very happy with our choice. We wanted to make a game where units had no health bars and no “equipment”—like chess pieces. Once we were happy with the core gameplay, we didn’t see much room for expansion in terms of unit types (even coming up with one for our DLC was a challenge!) and our pared-down aesthetic doesn’t really leave room for hats or other visual flourishes.

I have a soft spot for pay-once as a consumer, but I also acknowledge the power and benefits of free-to-play and fully intend to make use of it in the future. Pay-once isn’t going anywhere, so as developers we need to take the time to understand when it’s the right choice.

It is far from an evolutionary dead-end. ☹

PAUL TAYLOR is the co-founder of *Mode 7 Games*. You can follow him on Twitter: @mode7games.



TREAD LIGHTLY

SUSTAINABLE STEWARDSHIP FOR OUR FILES



ILLUSTRATION BY JUAN RAMIREZ

As artists, we pride ourselves on our unique creative abilities—on being able to see things in ways that nobody else can.

Unfortunately, we tend to take the whole roadless-traveled a bit too far, as anybody who ever has to work with other people's files on a regular basis knows. Ever opened up "armchair.max" and found not just a chair, but a fully rigged, textured character because somebody needed scale reference? How about trying to pop in and tweak some collision geometry only to confront 57 missing file dialogs pointing at textures in a temp directory on somebody's laptop? If you're lucky, you may even have experienced the perennial classic "file name says 'run' but the actual animation is 'stop running'" because a contractor used the run file to generate a transition and saved it under the wrong name. Annoying sure, but proof of our brilliance—only truly creative geniuses could find so many different ways to make each other miserable. This, my friends, is art.

Collaboration is a fact of life in the modern game business. The artist who gets to own a file exclusively for its entire life span is a rare bird

these days. The increasing technical complexity of our assets means that few of us can model, rig, animate, and provide physics or gameplay markup to a file at every stage of development. Even if you're a generalist who could do every one of the dozens of tasks needed to bring a model to life, there's always the chance that somebody else will have to take over your precious asset in the hectic weeks before shipping to cut down on polys, replace textures, or fix bugs. No asset is an island.

Even if you're perfectly happy with your own personal level of chaos, someday you will have to pick up where somebody else has left off. The odds are good that you'll spend half a day just poking around—hiding and unhiding meshes, resetting display layers, hunting for lost references, or simply marveling at the quirky leavings of your predecessors instead of, you know, actually working. We can all roll our eyes about it, but it's an entirely avoidable—and unprofessional—occupational hazard.

Since it's a fact of life, let's devote a few minutes to the ethics of shared work. It's a simple application of the Golden Rule—do unto your files as you would have done unto the files

you have to work with. Before your next check-in, take a few seconds and ask yourself if the file is really in a fit state to be seen or used by others. To help you along as you ponder, here's a little guide to how to tread lightly in shared files.

ORGANIZATION

» When you're busting out polygons in a creative frenzy, it matters little that every object in the scene is named "pCubeXXXX," or that your character's arms are called "R_leg" because they started out as duplicates, or that the shader for your floor is called "ceilingTemp." As long as you're actively modeling, your short-term memory and context cues—like where things live in outline view, or the fact that you can select them all with a particular swipe in one viewport—should provide enough information to keep the creative process going. Few artists are willing to step out of the flow of a good work session just to make sure that everything is neatly named and properly grouped.

Unfortunately, when things slow down, that context information is all too easy to forget. Coming back to a complex file after a long weekend is disorienting enough; opening up

somebody else's unholy rat's nest of a file with no prior knowledge at all is usually an appalling process. So it's important to take advantage of creative downtime to tidy things up while your creative faculties take a break. It's a kindness both to your teammates and to your future self. Never underestimate how quickly you'll forget exactly what "fix_2_final" is supposed to be, or how to tell the difference between shaders named "concrete," "concreet," and "Concrete_copy."

Descriptive names and a logical hierarchy don't take a lot of work to maintain, and they repay that minimal effort many times over the life of a file. You don't need to go overboard—it's probably okay to group the 10 futuristic forms that make up your starship's command pod together as "command_pod" rather than trying to cook up individual names for every sci-fi doodad. Just ask yourself these three questions before you close your file and hand it off to your colleague:

* If I had to get to some part of this model quickly, could I?

* If somebody hits "unhide all," will the file still be usable?

* Is the file exportable in the state I've left it?

If the answer to all three questions is "Yes!" then your organization is probably sufficient. More elaborate schemes with color-coded wireframes, layers, references, or asset containers can be useful (especially in very complex files, like big environments or complex rigged characters) but they can also become a source of conflict with your co-workers. Not everybody uses (or even knows how to use) the full gamut of organizational tools, so a minimalist approach is probably the best. When in doubt, revert to defaults—you may be comfortable working with triple-thick borders, visible face normals, and X-ray mode on everything, but most people aren't. Obviously, if your studio develops and sticks to a more elaborate approach, that's great (and very, very rare), but otherwise just stick with simple naming and default view settings.

DEPENDENCIES

» Another important aspect of being a good citizen is making sure to manage your dependencies. Our art programs aren't really designed from the ground up for collaboration and sharing; they're oriented around a single artist working off a local hard drive. But our models, characters, and animations are never a self-contained unit. Every texture, every referenced model, every shader file or reference image used in your file is supposed to be on the local disk—and woe betide the poor joker who

opens your file while it still references dozens of textures from your local system.

Obviously, proper file etiquette means making sure you check in all the relevant dependencies along with your files. Before you check a file in, prune out matters that the file doesn't really need—like photo image plates, or other models referenced in to provide scale comparisons, or textures that were considered and then discarded. Maya's Optimize Scene Size function is handy for cleaning out things like unneeded materials, but in Max you'll have to make sure that your scene materials and material library are in sync by hand. Don't forget to check for file references too, including dormant or unloaded references that don't show up in the viewport.

Naturally, the necessary stuff should be checked in—not only does that ensure that other people can see it, but it also will force you to trim out any references to quick-and-dirty stuff like files on your desktop or your thumb drive, since these aren't likely to be included in your source control setup. This process is purely mechanical, so it's also a great candidate for automation—talk to your friendly neighborhood tech artist about a tool that makes sure you check in textures, referenced models, and other dependencies together with your files.

EXPORTABILITY

» Art files exist only to export the assets that will end up in the game. For that reason, leaving your file in an exportable state is a key part of good file-handling manners.

Files frequently contain lots of extraneous material, for reasons ranging from the eminently practical to the entirely ridiculous. Scale reference or style comparisons may include extra models alongside the asset you're actually working on. Some artists like to keep visual variants on hand as they grope for a style. Sometimes, it's easiest to steal an existing shader from another model. And, occasionally, things just get weird. Over the years I've found everything from a collection of severed arms, to not-safe-for-work .WAV files, to a lovingly modeled 40,000-polygon dinosaur stashed away inside of seemingly innocent asset files.

All that is fine as long as you're iterating on the file locally. If it helps your creative process to have the complete fully modeled Eiffel Tower on hand when modeling café tables, then *à chacun son goût*. But when you're getting ready to check in, you need to clean up the souvenirs of your artistic journey. At the very least, make sure that non-game material is clearly segregated from the main content of the file. The intern who is working through a spreadsheet of files adding sound markup may not know or care enough to hide that Eiffel Tower before re-exporting, and the consequences will be *très comique*.

Ideally, you should just get rid of reference, construction kit pieces, and inspirational architecture when you're done with them. Never forget that the next person to open your file will probably hit "Unhide all" at some point. If you think the pieces are still needed, group them together in a single place for easy hiding—a nice clear group name like "DONT_EXPORT" or "HIDE_ME" is a good idea.

Along the same lines, you should also delete your construction history or collapse your modeling stack once you're done iterating. Although construction history can be very powerful (see "The History Channel" in the December 2006 issue of *Game Developer*), it's also going to bulk up your file and slow down the loading process. Including your construction history also makes it possible for a new user to inadvertently mess up a model by touching an upstream node or deleting what looks like an empty transform. Checking in a file is a great opportunity to simplify and streamline it.

The most important element of a good file, though, is that it exports correctly. Always make sure the files you check in correspond to the exported assets in the game. Animators, for example, frequently create variants using existing files—a crouched run based off of an ordinary upright run, say, or a transition starting from an existing cycle. That's all well and good unless the altered data gets checked in to the original file, so that the Run.MA now exports only a crouching run. This kind of thing is unprofessional and dangerous—not only does it mean there is content in the game that can't be reproduced, it will cause all sorts of chaos if some poor tech artist has to do a batch re-export. So please, make sure your files contain what they say they do—no more, and no less.

TAKE ONLY PICTURES, LEAVE ONLY FOOTPRINTS

» One of the toughest things about life as a game artist is that the old-fashioned stereotypes of art as a lonely, individualistic, and very inward process simply don't match up with the extremely collectivist nature of our work. It's hard enough just to make pretty pictures; add in the need to reconcile varying styles and clashing egos and it all gets pretty touchy. Observing the Golden Rule of art files helps to ease the inevitable stresses of working closely together—and if it catches on, we'll all spend less time tabbing through those blankety-blank missing texture dialogs. So pay it forward, tread lightly, and do the right thing—until next month, anyway. 🙏

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*.

GDC EUROPE 2012

GDC EUROPE ADDS JOURNEY AND GOLDENEYE 007 POSTMORTEMS, MAJOR PUBLISHER PANEL

In the latest update for GDC Europe 2012, show organizers have unveiled several new talks, including a postmortem of thatgamecompany's JOURNEY, a postmortem of Rare's GOLDENEYE 007, a panel on how publishers need to react to the latest industry trends, and a look at SOE's approach to social media.

These talks all fall within GDC Europe's Main Conference, which takes place Monday through Wednesday, August 13-15, 2012, at the Congress-Centrum Ost Koelnmesse in Cologne, Germany.

As part of the Game Design track, former thatgamecompany producer Robin Hunicke (now of Tiny Speck), will look back at one of the most highly regarded downloadable titles of this generation in "THE LONG JOURNEY." During this postmortem, Hunicke will outline how the PSN-exclusive JOURNEY embraced the unknown to create meaningful social gameplay without relying on traditional video game tropes. The game took three full years to make—double thatgamecompany's original goal—and Hunicke



will reflect on its production to determine whether the title lived up to its initial vision, while sharing some key lessons learned along the way.

Also in the Game Design track, former Rare game director Martin Hollis (now of independent studio Zoonami) will debut the first-ever GDC Europe classic postmortem, chronicling the development of seminal Nintendo 64 title GOLDENEYE 007. Hollis led the creation of this hit movie tie-in game, and during this session he will tell how the game came to be, detailing its roots as


a VIRTUA COP-influenced on-rails project all the way to its eventual release in 1997. It'll be a rare chance to get an inside look at the much-loved 8-million-unit-selling title that paved the way for the future of console-based first-person shooters.

Another new session in the Game Design track is developer Don Daglow's "5 Things About American Online Gamers that Surprise European Developers," in which he will explain why European online game developers need to prepare for some unique challenges when creating a title for players across the Atlantic. Daglow helped create the original NEVERWINTER NIGHTS (the AOL-graphic MMO), and has been working on online games for more than 25 years. In this session he'll detail some of the counterintuitive behavior patterns he's observed in North American game players to help European studios prepare for their own overseas launches.

Over in the Business, Marketing and Management track, a handful of major publishing executives will discuss their thoughts on the biggest trends facing game development in "Ask the Publishers: Adapting and Succeeding in a Changing Games Industry." In this panel, speakers from major companies including Microsoft, Konami, and Capcom, moderated by Remedy (ALAN WAKE) head Matias Myllyrinne, will discuss the latest digital-distribution models, new funding options like Kickstarter, and much more. The panelists will examine these trends, offering their take on how they will impact the industry and what developers and publishers need to do to survive in the years ahead.

Also in the Business, Marketing and Management track, Sony Online Entertainment's Linda Carlson will host

"Surviving Social Media: Experimentation Leads to Innovation," detailing how companies can improve their online community outreach and better take advantage of the latest online developments. Carlson will share SOE's own social media efforts, noting that the best way to find success with these initiatives is to experiment, abandon unsuccessful tactics, and constantly seek out new ideas. Attendees will walk away with a better understanding of how to leverage online media, regardless of their studio's size, structure, or ambition.

Keep an eye out for even more exciting keynotes and sessions; GDC Europe organizers have plenty more in store before the event opens its doors this August. For more details, see the GDC Europe web site at www.gdceurope.com. 





who went where

Catharina Mallet has left her position as executive producer at EA social studio Playfish to head up a newly formed development team in London for European publisher King.com. Mallet had been working on an undisclosed project for EA when she left, and had previously worked as a producer on *THE SIMS SOCIAL*.

Sony Corporation president and CEO Kazuo Hirai has stepped down from his role as representative director and chairman of Sony Computer Entertainment to focus on his leadership role in the company's consumer electronics division. Hirai will continue to serve as a member of the SCE board in a part-time role.

Capcom USA strategic marketing director and fighting game specialist Seth Killian has taken a role as lead game designer at Sony Santa Monica. He will be primarily working with third-party studios, and his first project will be *PLAYSTATION ALL-STARS BATTLE ROYALE* by SuperBot Entertainment.

new studios

Yoshiro Kimura, director or designer of experimental games like *CHULIP*, *LACK OF LOVE*, and *LITTLE KING STORY*, has started a new game studio called Onion Games, through which he hopes to make simpler, smaller experiences, inspired by the powerful indie game developers he saw at 2012's Independent Games Festival.

Electronic Arts, the Institute of Play, and the Entertainment Software Association have formed a new nonprofit studio called Glass Lab, meant to focus on making games for students across the United States. Glass Lab is located alongside EA's headquarters in Redwood City, California, and has received \$10.3 million from the Bill and Melinda Gates Foundation and the John D. and Catherine T. MacArthur Foundation.

Former Irrational Games and Harmonix devs Bryn Bennett, Steven Kimura, Arthur Inasi, Aaron DeMuth, and Mallika Sundaramurthy have started Eerie Canal, a new independent studio to focus specifically on "creative and inspired games that are too risky [for] large studios." Eerie Canal's first project is called *DREADLINE*, a hybrid action RPG/RTS due out in early 2013.

Sure Ryu Can!

TAIYOUNG RYU LEAVES ONLINE GAMES FOR CONSOLE GAMES—IN KOREA

SEEMS LIKE EVERY DAY WE HEAR ABOUT ANOTHER LONGTIME CONSOLE DEVELOPER LEAVING FOR GREENER PASTURES—SOMETHING SOCIAL/MOBILE/INDIE/FREE-TO-PLAY. TAIYOUNG RYU DID THE OPPOSITE; AFTER WORKING IN ONLINE GAME DEVELOPMENT IN KOREA, HE DECIDED TO START HIS OWN CONSOLE GAME DEVELOPMENT STUDIO CALLED KUNO INTERACTIVE.

Patrick Miller: *Where were you working before you started Kuno Interactive?*

Taiyoung Ryu: I was working at a small online game company called Kama Digital Entertainment in Korea, where I made two online games and a number of mobile games for the Korean game market, and did research on microtransaction business models. Now I'm working at a console game company called Kuno Interactive, which I started with my friends. Currently, we're making three console downloadable titles. I've been working as a game designer for 10 years, but right now I'm our creative director and CMO.

PM: *Why did you decide to switch jobs? Was there something about developing for consoles that made you want to take the new job?*

TR: After I quit my previous game design job, I went to America to study. I went to USC's Interactive Media Division and studied there for three years. That was my turning point, I think. When I was working in an online game company in Korea, I had a vague perspective on console game development. I had grown up playing console games, such as the *SUPER MARIO BROS.* series and many Japanese RPGs. Making console games was one of my childhood dreams, but it seemed impossible to make console games in Korea because I thought console game developers should be huge like EA, Nintendo, and Activision. None of the big game developers in Korea were interested in developing for consoles. When I studied

in the U.S., I saw many small teams making console games for downloadable platforms in school. Thatgamecompany was one of them. Their games were very inspirational. So, after coming back to Korea, I decided to make console games with my friends.



PM: *Are there many other Korean console game developers?*

TR: As far as I know, there is no console game developer except for my company right now. There were some console game companies in Korea some years ago, but they don't make console games anymore. For example, Phantagram, the developer of *KINGDOM UNDER FIRE*, is making the sequel of *KINGDOM UNDER FIRE* as an online game.


PM: *How are you adjusting to the difference between console development and online development?*

TR: Recently, console games are getting similar to online games. Increasingly console games sell virtual items using in-game shops and update game content over time. What I miss about working on online games is the open-ended development. When making my online games, I was able to easily change various features in those games

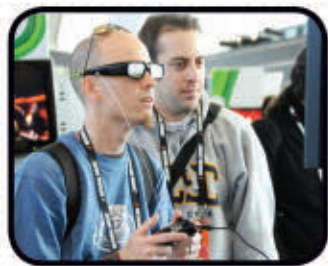
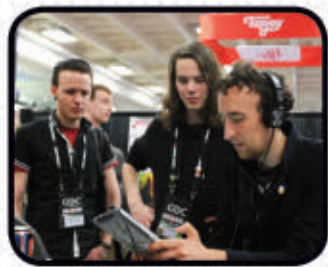
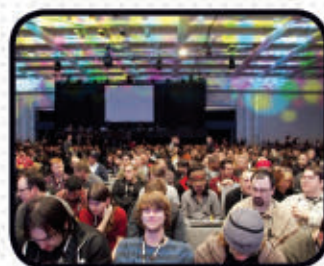
according to user feedback at any time, even after launch. It was fun. Console game development is like making an art piece—your game should be completely finished before you launch. Overall, I prefer console game development because online game development feels more like making a web service than a game. It's too business-oriented. Many online game developers care more about selling items than designing good games.

PM: *With online games, you were developing for a big Korean and Chinese audience, but your new console audience is mostly in America, the U.S., and Japan. Is the transition difficult?*

TR: Everything is different. There is a very small console market in Korea and China, but the online game market in Korea and China is as huge as America's console market. Those players prefer free-to-play games, so it's very important to design monetization models to make users spend money—more important, in fact, than having different gameplay or a creative game mechanic. With the console game market, I think it's more important to give your players a new experience.

American/European console game users usually play a game for less than one or two months, while Chinese/Korean online players often stick to a game for more than a year. With online games, you need to focus on communicating with your players, but for console games, it's more important to read the latest trends in the market. 

THE BEST ON-DEMAND CONTENT FROM THE GAME DEVELOPERS CONFERENCE SHOWS



GDC Vault

Streaming video, audio, and
PowerPoint presentations
from GDC 2012, GDC Europe,
GDC China, and GDC Online.

EDUCATION
GROUP RATES AVAILABLE!

For more information visit: WWW.GDCVAULT.COM



WWW.LOVEPUNKS.COM

LOVE PUNKS

WHAT HAPPENS WHEN YOU MIX GAME DEVELOPERS STU CAMPBELL AND DUNCAN GATES, A BUNCH OF ABORIGINAL AUSTRALIAN KIDS, AND LEFTOVER COSTUMES FROM AN INDIE ZOMBIE MOVIE? THE ANSWER IS LOVE PUNKS, A FUN FLASH GAME THAT TAKES YOU THROUGH A TOUR OF THE ROEBOURNE (WESTERN AUSTRALIA) LANDSCAPE—AND A LITTLE BIT OF THE KIDS' PERSONALITY AND CULTURE.

Patrick Miller: *What is the Yijala Yala project about, and how did LOVE PUNKS fit into it?*

Stu Campbell: The Yijala Yala Project is run by Big hART—a social arts company that has worked with remote and disadvantaged communities for 20 years. The project is supported by Woodside, a liquid natural gas company that has a conservation agreement with the federal government to conserve Aboriginal culture and heritage in the area. Big hART came on board to engage the indigenous community of Roebourne in creative projects to teach them new skills, so that in the future they will be well equipped to conserve and transmit their own culture.

I arrived on the project a year ago to assist filmmaker Telen Rodwell and indigenous actor Trevor Jamieson to make

PM: *Why did you decide to make a game?*

SC: I went into the local school with ideas of creating an interactive comic, but when I arrived in the classroom I realized very quickly that all the kids were game mad, so I benched the comic idea and decided we had to make a game. I wanted to make a game that would reflect the kids' personalities and the environment that helps to shape them. The kids are hilariously, cheeky, and full of energy. The geography surrounding Roebourne is beautiful, diverse, and bloody hot all year round. It reached 52°C [125°F] last year! The Aboriginal kids from the community will happily spend the days inside playing games, but as soon as that sun goes down they'll be outside in a second. So with that in mind I knew I had to make an "outdoor" game.



what areas of Roebourne should be in the game and how they'd interact with those environments. They had so many ideas that the only way to deal with them was to create one big crazy montage: The mudflats blend into the junkyard, which blends into the river, then the sand dune, then the Burrup, and finally the desert.

PM: *Were the kids involved in the game design, as well?*

SC: Yes! They worked with our good old Flash programmer, Duncan Gates. Eleven-year-old Brodie (who holds the highest score in the game) had a big long list of fixes and improvements. He introduced the life meter, and came up with most of the ideas of how you score points, lose points, and get bonus points. He was also persistent about cheats. He kept saying "The game has to have cheats or it sucks!"

Meanwhile, 12 year-old Maverick came up with ideas like "The peacock gives you life." The peacock came into the game when a real peacock walked past our office window. We chased that peacock with the camera, filmed him and imported the frames into Photoshop for clipping. We also added the frogs we found in our office toilet and a bearded dragon we found snooping around our pond. One of the kids grabbed him by the tail and held him in front of the green screen while I filmed. I'm

pretty sure catching that lizard was the kids' favorite part of the process.

Duncan Gates: One of the interesting things about the development of this game was the organic, evolving nature of the creative process. The kids were coming up with ideas and we were adding them into the game on the fly. This presented a few challenges on the coding side of things as it goes against the usual software development process cycle, but ultimately it felt more creative and rewarding to do it that way!

PM: *How'd you come up with the costume design?*

SC: Everything we do here is DIY and limited by what we can find locally. The costumes we designed with the kids. We tore clothes apart, spray-painted them, removed sleeves, tied sleeves around ankles. One of the kids hung a steering wheel from his and neck and another kid has a car light strapped to his shoulder. When Brodie was recruited to the gang, 11-year-old Nathaniel helped him design a costume and sketched his face paint in Photoshop using his Wacom pen.

PM: *What was your goal for LOVE PUNKS?*

SC: My personal goal for the game was to make Roebourne look like the coolest place on the planet—which it is! 🐼



a zombie film titled *Love Sweet Love*. For the film, I created the kids' costumes and designed their face paint, and together we slowly established their overall identity, which became known as the Love Punks. Once we screened the film in the community, every kid in town wanted to be a Love Punk. So when thinking about what project to do next, I knew it had to be another incarnation of the Love Punks and it had to include everyone!

PM: *How were the kids involved?*

SC: We started off by filming the kids in front of a green screen, imported the film into Final Cut Pro, and exported it as a JPEG sequence, then we imported the frames into Photoshop as a layer animation. From there, I taught the kids how to cut themselves out of the background. They also learned a load of shortcuts: **Command + S** quickly became the class mantra.

The kids and I would have regular meetings and discuss

Publisher/Developer:

N/A

Release Date: May 10, 2012

Development time: 4 months

Development budget: \$20–25K

[This is a rough guess as my time is divided between making games, comics, and films. It also includes the cost for employing a sound engineer and programmer]

of lines of code in the game: 5,900

A fun fact: We met the miner zombie at a pub. He was an indigenous fella too, and was keen to be in the game.

LEARN TO MAKE VIDEO GAMES IN
THE HEART OF
HOLLYWOOD!

SCHEDULE A TOUR TODAY

DesignLAFilm.com
800-406-7485



THE
LOS ANGELES[®]
FILM SCHOOL
ANIMATION + AUDIO + FILM + GAMES
ENTERTAINMENT BUSINESS

“Entertainment interchanges from movies to games all of the time. This makes The Los Angeles Film School an ideal place to learn because it presents a great opportunity for those involved in graphics and movies.”

—NOLAN BUNSHALL, *Founder of Atari*



For more information on our programs and their outcomes, visit www.lafilm.edu/courses. ©2011 The Los Angeles Film School. All rights reserved. The term “The Los Angeles Film School” and The Los Angeles Film School logo are either service marks or registered service marks of The Los Angeles Film School. Accredited by ACCSC.

**INFORMING,
ENGAGING, AND
EMPOWERING
THE INDUSTRY**

gamasutra.com

the art and business of making games



GAME DEVELOPER MAGAZINE

the best of postmortems, product reviews, and standout columns

GET THE
PRINT+DIGITAL
ACCESS BUNDLE FOR ONLY

\$49.95 /YEAR

+ DIGITAL ACCESS
TO BACK ISSUES

+ EXCLUSIVE
INTERACTIVE EXTRAS

INCLUDES:



PRINT
SUBSCRIPTION



DIGITAL + GAME
DEVELOPER APP

+

BONUS!



BEST OF
POSTMORTEMS
PRINT ISSUE

SUBSCRIBE TODAY!

GDMAG.COM/SUBSCRIBE



ADVERTISER INDEX

COMPANY NAME	PAGE	COMPANY NAME	PAGE
AUDI SG	6	PERFORCE SOFTWARE	14
BLIZZARD ENTERTAINMENT	30	RAD GAME TOOLS	C4
EPIC GAMES	13	TOKYO GAME SHOW	40
GREE INTERNATIONAL	C2	TWOFOUR54	3
HAVOK	C3	VANCOUVER FILM SCHOOL	33
LOS ANGELES FILM SCHOOL	54	XSOLLA	33

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. All content, copyright *gd Game Developer* magazine/UBM LLC, unless otherwise indicated. Don't steal any of it.



VOX POPULI

GAME DEVELOPERS SPEAK OUT

Ever wonder what game developers really think? Well, we asked them! Check out these exclusive, unedited opinions from game developers of all stripes that give us the straight talk on the issues that concern them most.

MOST FAMOUS GAME DESIGNERS AREN'T THAT GREAT

by A Not Famous Game Designer

» I think this "famous game designer" thing is totally out of hand. Look at Peter Molyneux, for example. What all has he done, anyway? If you look at him and all his big awards, and going on TV, and being in magazines, and so on, and then you look at a guy like me—it's like, hey, I've worked hard and done some pretty amazing work, too. My student game, COLORED DOTS, only shipped because I crunched for close to a week on it to get it done. I even

rewrote the whole dot-coloring algorithm in a single day. And now, thanks to that effort, I'm rocking it in the big leagues—staying up all night getting free food and booze to work on a scripted artillery shell sequence for the next big triple-A game! So have fun with your airy thought experiments, Mr. Molyneux—I've got a game to ship here.

SWAG THESE DAYS JUST AIN'T THE SAME

by Grizzled Industry Veteran

» You kids all comin' back from E3 with your bags fulla trinkets

and [garbled], well that's neat I suppose, if a fella likes junk, I say. Time was, when you'd go to E3 and you'd get giveaway stuff so great we'd be hootin' and hollerin' all the way back to the Figueroa. It was so high quality it'd go for a bundle of cash on the old eBay, which we'd turn around and use to buy [unintelligible] and some of those anime pencil boards.

I remember in particular I had one of those purple foam Nintendo GameCubes from—2002? 2001? People were waiting in line to play the GameCube and it was just taking so darn long. I almost left, but my friend wanted to see ROGUE SQUALID, or whatever it was called. The Nintendo man came around asking, "You get your foam GameCube?" and fwip!—he'd hand you one, right there. They just don't do that anymore.

[sighs] I really liked that old foam GameCube, but you know how it is. We used to horse around at the office some during all the late hours, and it got dirty and a little tore up. Then one day it got dropped it into a vat of sour cream when we had the taco bar for crunch food. Yeah, I think that was the end of the line for that little guy.

THIS NEOGAF THREAD PROVES I'M RIGHT

by Design Lead

» Pretty sure I just successfully blocked some bad influences from the marketing team a moment ago. There's been this ongoing thing where they want to make our game more accessible to a larger audience by having it be more about "simple fun" and so on. You know, Kinect and all that.

Thankfully, I think I was able to sway their opinion when, during a meeting, I quickly called up a random NeoGAF thread and showed it to the marketing people. "Look—this is what gamers think," I told them. They looked at the screen for a long time. I think the message sunk in, because they pushed the laptop away and ended the meeting right then and there. So I'm real happy about that; I'm

going to tell everyone on NeoGAF about it tomorrow.

THIS JAUNTY MAIN MENU MUSIC LOOP HAS BEEN STUCK IN MY HEAD FOREVER AND IT WON'T STOP

by Timmy the Night-Shift Tester

1. Boot game
2. Scream, tear hair out aggggakkalkal

I'M JUST AN ARTIST, MAN

by A. Artist

» I'm just an artist, man, I don't turn on my computer, hey I came in and my computer was off, what do I do? I'm just an artist, I just draw things, I don't know about all this machine language, binary logic stuff, I don't get the polygons and megabytes and all that stuff, man, I just do art, you know what I mean? Hey, I need some of those propeller heads to come help me, I need help with this crazy system you guys have here, I made something I wanna use but I just get this error that says, file not found, what does that even mean, man! I'm just an artist! Can you come over here and press the export button on the exporter for me? I never really got into pressing buttons, 'cause it's like, it just feels so corporate and button-down, you know, pressing buttons to do things, it's like this dystopian society just totally became real. Like, oh, here, press this button to do your job, it's so corporate, man. Why don't you guys get out of the way and just let me work, anyway? I'd have all the art for the game done in like, two, two and a half hours if I could just like concentrate on doing art, instead of answering emails or being in meetings or whatever, that's not even like my job at all because I'm just an artist, man! 🎨

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

The Havok Strike Program

For Small Projects and Mobile Development

The industry's best technology

in your pocket

The Havok Strike Program offers teams who are working on small projects and/or mobile games to have access to the same award-winning tools and technology as the biggest titles in the industry at a cost that fits their budget.

Havok technology is fully supported on all major consoles, mobile devices such as Android, iOS, and Windows 8, and handheld consoles such as PlayStation Vita®.

- Platform optimized runtime technology
- Project budget-based licensing model*
- Industry leading support
- Cross-platform compatibility

Havok™ Technologies Include:

Havok Vision Engine • Havok Physics • Havok AI • Havok Animation • Havok Behavior • Havok Cloth • Havok Destruction • Havok Script

*Some licensing and support restrictions apply



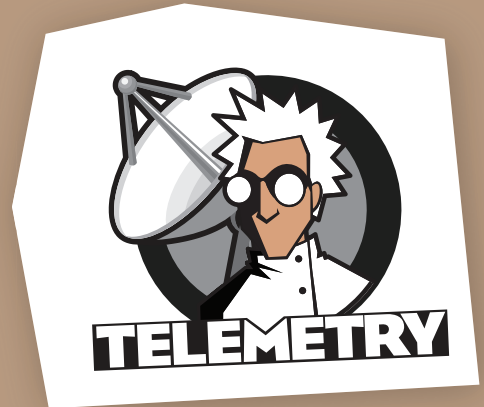
Learn More: www.havok.com/strike

THE NEWEST RAD TOOL IS

yes, it's **NEW**

NOW SHIPPING

OUT
OF THIS
WORLD



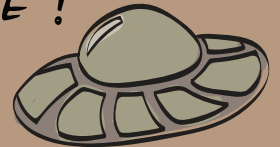
TELEMETRY

42

is a programming library and set of tools for instrumenting, profiling, tuning and visualizing application **PERFORMANCE!**



VISUALIZE real-time game performance,



see **WHEN** things happen — not merely **WHAT** happened!

PROBE the hierarchical display —

see thread interactions, context switches, and mutex locking!

THIS ISN'T JUST ROCKET SCIENCE, THIS IS **rad!**



www.radgametools.com/telemetry
(425) 893-4300