

MASSIVE ATTACK

A NOVEL SERVER ARCHITECTURE FOR 256+ PLAYERS

HOT METRICS

DO-IT-YOURSELF PLAYTESTING ON THE CHEAP

SINGULAR

POSTMORTEM

ITY

*Charleston (S. Carolina)
4 May 1842*

JUL 16
42



Developer

THE LEADING GAME INDUSTRY MAGAZINE



UBM VOL17 NO8 SEPTEMBER 2010

...ALLY STAYED IN THE

- Tim Sweeney, CEO of Epic Games

Scaleform®
GFx 3.3

- Split Screen / Multi Controller Support
- Reduced Memory Usage
- 3Di Rendering - Now in Stereoscopic
- AMP Profiler Tool
- Flash Filters - Shadow, Blur, Glow
- Direct Access API < 1ms HUD Kit
- HD Video & Audio Playback
- CLIK UI Widgets

Scaleform GFx and Scaleform logo are © 2010 Scaleform Corporation. Image of Unreal Tournament 3 © 2010, Epic Games, Inc.



Scaleform®



POSTMORTEM

20 RAVEN SOFTWARE'S SINGULARITY

After creating a number of high-profile licensed titles, the developers at Raven Software had an idea for an original shooter with a unique time shifting mechanic. To convince their parent company, the team rapidly built a demo that made judicious use of pre-existing assets and engine technology. This approach paid off over the course of the project by putting the design focus for SINGULARITY squarely on gameplay. *By Rob Gee, Brian Raffel, Steve Raffel, Jon Zuk, Dan Vondrak, and Gustavo Rasche*

FEATURES

7 BIG WARS

Massively multiplayer combat is the new frontier for First-Person Shooters. While there are a number of titles that combine fast-paced, FPS gameplay with a persistent MMO-style world, hosting truly large numbers of players at once remains a technical hurdle. Here, veteran networking engineer Lin Luo proposes a new approach to client-server architecture that uses a central server to coordinate the distribution of data across multiple server systems. *By Lin Luo*

15 HOT FAILURE

Large-scale playtesting is usually a luxury that only big studios and publishers can enjoy. While it is a crucial step in tuning gameplay, how can small teams and individual developers gather playtest data? As a one-man Android developer, Chris Pruett approached the problem by building an event-logging system into his game REPLICAS ISLAND that gathered player performance data automatically. After being aggregated and drawn onto a heat map, the resulting metrics quickly showed where gameplay needed to be tweaked in order to provide a smooth user experience. *By Chris Pruett*

DEPARTMENTS

- | | | |
|----|--|---------------|
| 2 | GAME PLAN <i>By Brandon Sheffield</i>
The Power of Mystery | [EDITORIAL] |
| 4 | HEADS UP DISPLAY
HALO 2600 and Hirokazu Yasuhara's design notebook. | [NEWS] |
| 27 | TOOL BOX <i>By Tom Carroll</i>
CrazyBump 1.101 | [REVIEW] |
| 31 | THE INNER PRODUCT <i>By Noel Llopis</i>
Data-Oriented Design | [PROGRAMMING] |
| 34 | PIXEL PUSHER <i>By Steve Theodore</i>
Big Screen Blues | [ART] |
| 36 | DESIGN OF THE TIMES <i>By Soren Johnson</i>
The Chick Parabola | [DESIGN] |
| 39 | AURAL FIXATION <i>By Vincent Diamante</i>
SFX Frequency Mapping | [SOUND] |
| 40 | GOOD JOB! <i>By Staff</i>
Chris Archer Q&A, who went where, and new studios. | [CAREER] |
| 42 | EYE ON GDC <i>By Staff</i>
GDC Online announces new panels and lectures. | [GDC] |
| 44 | EDUCATED PLAY <i>By Tom Curtis</i>
DREAMSIDE MAROON | [EDUCATION] |
| 48 | ARRESTED DEVELOPMENT <i>By Matthew Wasteland</i>
Consensus Reached | [HUMOR] |



THE POWER OF MYSTERY

HOW THIS SUBSET OF LUCK CREATES THAT "JUST ONE MORE" FEELING

"NOTHING IS SO FRIGHTENING AS WHAT'S BEHIND THE closed door. The audience holds its breath along with the protagonist as she/he (more often she) approaches that door." So writes Stephen King in his non-fiction book on horror, *Danse Macabre*. King was not the first to make this point, nor will he be the last—with the right setting, the closed door, with all its possibilities, can be a frightening thing. Contrast that with an open door with light streaming through it, a traditional symbol of hope. But the closed door nags at you—don't you want to know what's on the other side? The real power of the closed door is the mystery behind it. So long as it's closed, the possibilities remain infinite. That constant barrage of mystery is incredibly enticing to players, and is often directly responsible for that "just one more ..." feeling that many games aspire to.

This idea has been used in games for years, occasionally in ways that are analogous to the one King discusses, such as the simple build of tension you might see in the *SILENT HILL* series. As the player approaches a locked door, disconcerting noises increase in intensity, and maybe the world begins to erode, as happens in the series. The player has to go through the door, there's no other way—but they almost don't want to. The player is complicit in the act of approaching the door, which by turns increases or decreases the horror, depending on how much the player already knows. In film, the viewer is less likely to have advance knowledge of what lies beyond—but in games, we might have gone through the same scenario a few times, diminishing the effect.

UNREALIZED DREAMS

» Unfortunately, the payoff is usually not as exciting as that anticipation of terror. Essentially, the imagination usually cooks up something far more exciting than anything we can deliver as developers.

The concept is similar to the classic "Pavlov's Dog" experiment, in which researcher Ivan Pavlov rang a bell (or gave some sort of other stimulus) every time a dog was given some food—over time, the dog would start salivating as soon as it heard the bell, regardless of whether it got any food, because it associated the sound with a reward. In games, as long as there are constantly new things to anticipate, the mind can continue to invent new potential rewards.

This is basically the way we condition players with things like treasure chests and monster drops. They know there's something in there, so they'll fight through hell and back to get to it, even if (in the case of JRPGs especially, but also in Western stalwarts like *DIABLO*) it could be a trap, or a monster in disguise, or have some other sort of ill effect. Over the years, we've come up with a pretty well-accepted formula for this, used (with some variation) by games from *WORLD OF WARCRAFT* to *PERSONA* to *BORDERLANDS*. Chests will generally have

something good in them—the excitement is in not knowing just how good it'll be. This keeps players digging for more chests to get that epic loot. The same applies to items grabbed from felled monsters.

Taking it further, this idea of mystery applies to dialog-heavy RPGs like *DRAGON AGE* or the *PERSONA* series. Whenever the player is given a set of response choices in a dialog scenario, there is an air of mystery—how will the other character respond? You generally have some sense of it—choices generally yield semi-predictable responses—but again, you don't know just how it will affect your relationship with this character in the long term. Therein lies the mystery.

This not only helps strengthen the illusion that you're building a relationship with a character (alongside positive and negative feedback, which both aforementioned games provide), it also keeps players digging to see what will happen. The "just one more" idea returns.

MYSTERY VERSUS LUCK

» This sort of mystery I'm talking about is a subset of luck. It's far more specific, and as a result is easier to control. It can be frustrating in a game like *PUZZLE QUEST* to have your opponent hit you with a huge chain of "random" attacks, or to have your weapon randomly break in an RPG. That sort of luck can be frustrating. With the mystery of a treasure chest that may drop an epic weapon though, the outcome is always positive, which makes for higher engagement and less frustration.

Mystery isn't always good though—choosing a difficulty level before you've started the game, for example, tends to lead to frustration.

Genres other than RPGs seem less able to use the more straightforward tricks. *BORDERLANDS* is an exception with its randomly generated weapons found in treasure chests, but by and large FPS games have to rely on a set group of weapon drops from downed enemies. So how do we get this "just one more" phenomenon in other genres? It's a very "gamey" sort of interaction, which potentially goes against the realism many games strive for, but MMOs and some online FPS use the anticipation of leveling up in a similar way, and fighting and racing games often use unlockable characters or outfits. These are far less of an addictive gameplay element than they are a bonus, but the concept is similar.

In all, I think most games would be well served by including some element of mystery. It adds stickiness, and keeps players playing long after they might otherwise have stopped. It forms a strong link with the player, so that they keep playing "without knowing why." Games that don't do this tend to fall by the wayside. Seems an obvious choice to me!

—Brandon Sheffield

600 Harrison St., 6th Fl.,
San Francisco, CA 94107
t: 415.947.6000 f: 415.947.6090

SUBSCRIPTION SERVICES

FOR INFORMATION, ORDER QUESTIONS, AND ADDRESS CHANGES

t: 800.250.2429 f: 847.763.9606
e: gamedeveloper@halldata.com

FOR DIGITAL SUBSCRIPTION INFORMATION
www.gdmag.com/digital

EDITORIAL

PUBLISHER

Simon Carless | scarless@gdmag.com

EDITOR-IN-CHIEF

Brandon Sheffield | bsheffield@gdmag.com

PRODUCTION EDITOR

Jeffrey Fleming | jffleming@gdmag.com

ART DIRECTOR

Joseph Mitch | jmitch@gdmag.com

PRODUCTION INTERN

Tom Curtis

CONTRIBUTING EDITORS

Jesse Harlin
Steve Theodore
Daniel Nelson
Soren Johnson
Damion Schubert

ADVISORY BOARD

Hal Barwood Designer-at-Large
Mick West Independent
Brad Bulkley Neversoft
Clinton Keith Independent
Brenda Brathwaite Independent
Bijan Forutanpour Sony Online Entertainment
Mark DeLoura Google
Carey Chico Independent

ADVERTISING SALES

GLOBAL SALES DIRECTOR

Aaron Murawski e: amurawski@think-services.com
t: 415.947.6227

MEDIA ACCOUNT MANAGER

John Malik Watson e: jmwatson@think-services.com
t: 415.947.6224

GLOBAL ACCOUNT MANAGER, RECRUITMENT

Gina Gross e: ggross@think-services.com
t: 415.947.6241

GLOBAL ACCOUNT MANAGER, EDUCATION

Rafael Vallin e: rvallin@think-services.com
t: 415.947.6223

ADVERTISING PRODUCTION

PRODUCTION MANAGER

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

REPRINTS

WRIGHT'S MEDIA

Ryan Pratt e: rpratt@wrightsreprints.com
t: 877.652.5295

AUDIENCE DEVELOPMENT

TYSON ASSOCIATES Elaine Tyson

e: Elaine@Tysonassociates.com

LIST RENTAL Merit Direct LLC

t: 914.368.1000

MARKETING

MARKETING SPECIALIST Melissa Andrade
e: mandrade@think-services.com



United Business Media
WWW.UBM.COM

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



yet Arabic game development
is still in its infancy.

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

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

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

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

we are twofour54°. are you?

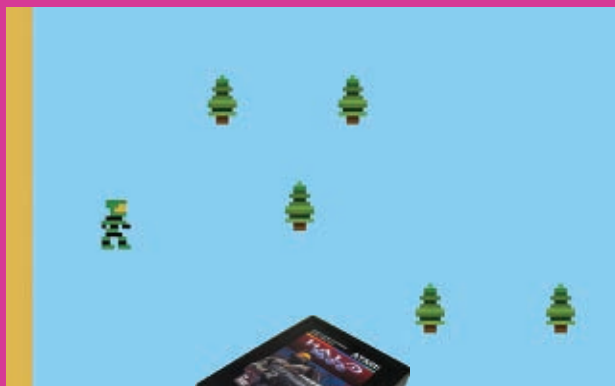
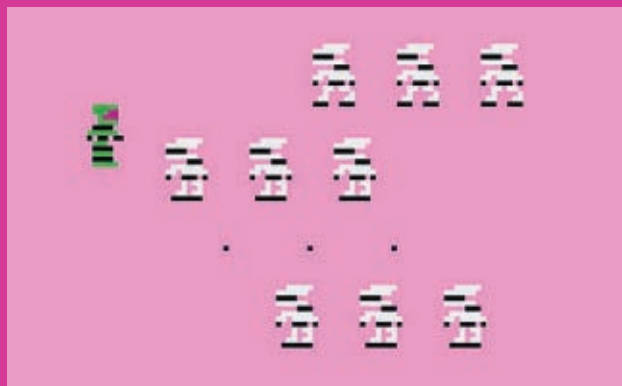
find us. join us. create with us.

+971 2 401 **2454** twofour54.com

twofour54°
Abu Dhabi

content creation community

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



HALO 2600 RELEASED

THE CLASSIC GAMING EXPO, held in Las Vegas at the beginning of August 2010, was host to the usual museums, swap meets, and panels, all surrounding gaming's storied past. Particular attention is paid to Atari systems, so it's no surprise that the largest buzz at the show surrounded a new homebrew game for the Atari 2600, the first truly mainstream game console.

That game is **HALO 2600**, coded by Ed Fries, who was vice president of the Xbox division during the lifetime of the original console. He had gotten his start programming Atari 800 games in high school and college, and was a very early Microsoft employee, taking over the game

division out of personal interest (and against the warnings of several other execs, he says).

Fries left Microsoft in January 2004, but as he got further from game creation, that itch to make something new returned. After reading the book *Riding the Beam*, he decided to try his hand at programming a 2600 game, in spite of not having written 6502 assembler code in almost 30 years. For lack of a better idea, he decided to make a little Master Chief run around in an environment—and the game was born. But not without difficulty.

"The thing you need to realize about the Atari 2600 is that it is an incredibly limited

machine," said Fries in a post on the Atari Age forums. "It has only 128 bytes of RAM and without bank switching, the maximum program size is just over 4,000 bytes. There are just two 8 pixel wide monochrome sprites, two one pixel bullets, a 'ball' and a 40 pixel wide background (and even that is exaggerating). There is no memory to store the screen image like any modern console or PC; instead it has to be drawn a line at a time by changing the values of the registers that control the sprites and background. The processor is so slow that only 76 clock cycles occur while a line of the screen is being drawn, and the simplest 6502

instructions take at least 2 clock cycles. So just to draw an image of the Master Chief is pretty tough. To create a complete game while living within these constraints is much harder."

But create it he did, thanks to the multitudinous homebrew tools that now exist for the console, as well as the help of the vibrant hobby community. The game is currently available for download from freeware rom sites, and a limited number of 2600 cartridges were sold at the expo. And for those adventurous souls who find themselves completely absorbed by the game, you might want to investigate a few of the buggier areas a bit more closely. Fries

explains how he found himself in "the magic land."

"I was working on a bug with the boss encounter and accidentally found myself completely outside the 64 room map," he writes. "I was wandering through memory that was never intended to be interpreted as part of the map but the code was doing the best it could to interpret what was being thrown at it. Strange, misshapen monsters attacked me in even stranger ways as I wandered through this bizarre land that I had unintentionally created. I left a bug or two in the final game to allow others to find and explore this strange landscape as I did."

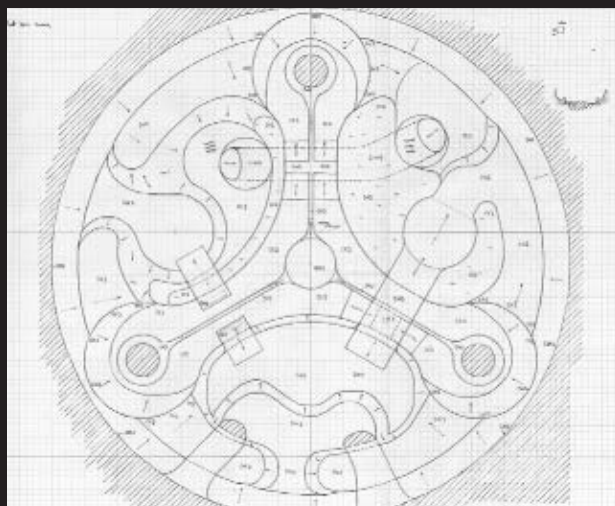
—Brandon Sheffield



DEMOSCENE-DOKUMENTTI CHRONICLES FINLAND'S DEMOSCENE HISTORY

The demoscene in Finland is the root of that country's rich game development heritage, from **STARDUST** to **MAX PAYNE**. Thomas Puha, editor of Finnish consumer game magazine *Pelaaja*, has put together a series of interviews that preserve the memories and histories of those involved in the scene from its origins in the early 1990s up until today. The series will spread across seven 10-minute episodes, beginning with a segment on the legendary Future Crew. The documentary will be released in full at the Alternative Party Finnish demoscene event, but will also be available online for free. See <http://demoscenedoc.com> for more info.

—Brandon Sheffield



DESIGNER'S NOTEBOOK: HIROKAZU YASUHARA

SONIC THE HEDGEHOG LEVEL DESIGNER HIROKAZU YASUHARA DREW COMPLEX PEN AND PAPER DESIGNS FOR THE MAPS IN NAUGHTY DOG'S RACER JAK X. HE HAS USED PAPER PROTOTYPING AS PART OF HIS DESIGN PROCESS THROUGHOUT HIS CAREER, INCLUDING HIS WORK ON THE UNCHARTED SERIES.



SEAPINE AGILE EXPEDITION™

Join the Adventure Today!

eXplore Agile in an eBook Adventure

The Seapine Agile Expedition is a **free** eBook-based learning series that provides a fun and informative overview of **Agile development**.

New to Agile? Discover how Agile boosts the performance of your entire team—developers, quality assurance, product owners, and ultimately, your customers—through timely communication and collaboration.

Already using Agile? Learn how to meet deadlines, respond to change, reduce risk, and deliver what you promise through Agile development powered by Seapine ALM.



www.seapine.com/gamebook

© 2010 Seapine Software, Inc. Seapine Software, Agile Expedition, and the Seapine logo are trademarks of Seapine Software, Inc. All rights reserved.

 Seapine Software™

SERVER ARCHITECTURE FOR 256+ PLAYER REAL TIME MULTIPLAYER FPS GAMES

BIG WARS

CONTEMPORARY REAL-TIME MULTIPLAYER FIRST-PERSON SHOOTER (FPS) GAME SERVER SYSTEMS FACE TWO problems that, at the practical level, limit the maximum number of players that can be hosted during any one game session: the CPU intensive real-time world simulation for all the in-game objects on the server, and the Input/Output (I/O) intensive network bandwidth consumption for synchronizing in-game object states to all the connected clients in a real-time fashion. This article will introduce an innovative server architecture that aims to defeat the above limitations, and enable 256 or more players to engage in real-time multiplayer FPS game sessions. However, please don't confuse this with existing MMO server cluster technologies that are able to host thousands of players but do so with a much slower non-real-time pace.

BIG WARS

FPS games that aim to provide a persistent world like that found in the MMO EVE ONLINE can face steep technical hurdles when using traditional client-server architectures.



//// THE CURRENT LIMITATIONS

A typical real-time FPS server system works like this: the server, being authoritative on the game world, simulates all the in-game objects, based on the latest input information from each client, at a predefined frame tick rate (typically 30 frames per second). For each connected client, the server also synchronizes those object states that are relevant to the client at a predefined network tick rate (typically 20 times per second). This high synchronization frequency for each connected client is necessary to maintain the accuracy of any real-time FPS game session. The server-side authoritative world object simulation is necessary to defeat client-side cheating and avoid inconsistent world object states that could happen if the simulation were distributed on the client side.

Suppose we are running a 32-player real-time FPS game in a typical client/server setup, and each client is able to request the server to spawn a small number of dynamic physics objects in the world (let's say 4). The server then has to do an authoritative object simulation (mostly physics) for 128 (4x32) in-game objects, which doesn't sound too bad for a real-time multiplayer FPS session. However, when the number of players grows to 256, the number of dynamic objects simulated on the server side balloons to 1,024, which requires a much more powerful server to handle in real-time.

If we attempt to tackle the problem with a client-side authoritative object simulation approach, the server now has to handle much

more intensive incoming data from each client. Each authoritative client also has to tell the server all of the client-side simulated object states, which yields a much higher chance of client-side cheating.

A simplified but quite optimized server synchronization update logic would look something similar to Listing 1 (Note: the code is pseudo, and utilizes some C++0x range-based for-loop syntax to make the reading easier).

As we can see from the code logic in Listing 1, even this quite optimized server update function is still bound to the number of connected clients and the number of objects that are relevant to each client. In a real-time game server system, we have to avoid any lengthy iteration logic inside each tick of the game update loop. Furthermore, when the number of connected clients grows, the server-side outgoing bandwidth requirement grows proportionally. So as the number of clients increases, the server tick rate will drop proportionally while the bandwidth requirement expands proportionally. At some point, when the synchronization frequency drops below an acceptable figure for each client to maintain a reasonably real-time gameplay update, we have to stop adding more clients to the game session. That's why most of the real-time multiplayer FPS games have a maximum number of players (64 is the largest number to date) that can be supported in one game session.

However, a multiplayer game that could support even more users playing at the same time in the same world would be much more

immersive. Just as we push graphics, artificial intelligence, character animation, and physics to their limits, we should be looking at ways to expand player connectivity. Without innovations in these areas, we would probably still be playing text-based multiplayer games!

Contemporary massively multiplayer online games easily host a high number of simultaneous players but with a trade-off: MMOs are usually not real time as far as the in-game characters/objects movement updates are concerned. MMO game servers generally don't perform tick-based object synchronization to each of the connected clients (the number could be huge). Instead, these servers only send reliable object state change updates to relevant clients through Transmission Control Protocol (TCP) connections instead of User Datagram Protocol (UDP), so that no object state changes are lost. Character and object movements in MMO games (on both the server and client side) are also more predictable through deterministic path calculations and kinematics simulation, and they don't require any dynamic rigid body simulation at all. This means less CPU power demand on the server-side for object simulation as well as smaller server-side bandwidth requirements for the changed object states that are synchronized to relevant clients.

We often hear of multiplayer games that claim to be MMO first-person shooters such as SUDDEN ATTACK and CROSS FIRE. However, I believe that to be a misnomer for most of these games. Many MMOFPS games do feature persistent player statistics but there is no real persistent

world similar to the ones provided by EVE ONLINE or WORLD OF WARCRAFT. Also, while their server farms can host thousands of simultaneous players playing inside hundreds of independent short game sessions, each short game session is only able to host a maximum of 32 simultaneous players (16-player game sessions are actually the most common). Each of these short game sessions adopts a traditional client/server architecture that employs a client-side authoritative simulation to minimize server load (with the game server acting only as a message relaying hub). This is why multiple server processes in these games can be arranged to run on a single server machine, resulting in a large number of game sessions that can be hosted simultaneously on a server farm with a relatively small number of hardware units. So as we can see, although thousands of players are able to play simultaneously with a persistent player statistic system, each battle can still only support a maximum of 32 players, which is not really a great technical innovation.

To create a truly massive multiplayer FPS, we have much harder requirements compared to other MMOs:

- The multiplayer game session must be in real time.
- The multiplayer game session must host up to 256 simultaneous players.
- The perceived synchronization latency must be minimal, or as unnoticeable as possible.
- The multiplayer game session must be server-side authoritative to avoid any possible client-side cheating and world state inconsistencies.

With these requirements in mind, is it still possible to create a multiplayer game that can host 256 or more players simultaneously while still running in real time? I believe the answer is yes, and the solution lies in distributing CPU and I/O load across multiple physical server systems with the help of a central coordinator.

////// ADDING MORE PLAYERS

Let's visualize our MMOFPS server system as a distributed server cluster arranged in the following fashion (see Figure 1):

As shown in Figure 1, each Battle Server does authoritative simulation for those objects that are "owned" by that Battle Server. Each Battle Server is responsible for the player characters controlled by the clients connected to that server, as well as all the dynamically spawned objects requested from the connected clients. A Battle Server would authoritatively simulate dynamically spawned objects resulting from its server logic, as well as whatever statically configured objects are generated when it starts up. These are the only

LISTING 1

```
class Server
{
public:
    void Update(float elapsed)
    {
        typedef std::set<Client::ptr> Clients;
        Clients synchronizableClients;

        typedef std::map<ObjectId, ClientSet> ObjectClientsMap;
        ObjectClientsMap objectClients;

        // Build up a processed registry that has the information on
        // the set of clients that each object is relevant for (since
        // each object could be relevant for more than one client!)
        for (auto entry : m_clients)
        {
            const Client::ptr& client = entry.second;
            if ( client->ShouldSynchronize(elapsed) )
            {
                synchronizableClients.insert(client);

                ObjectSet& objects = client->GetRelevantObjects();
                for (auto object : objects)
                {
                    ClientSet& clients = objectClients[object];
                    clients.insert( client.GetId() );
                }
            }
        }

        // Serialize each object's state exactly once, and cache the object
        // state data to each client that this object is relevant for
        // Note: only changed objects are serialized!!
        for (auto entry : objectClients)
        {
            ObjectId object = entry.first;
            ClientSet& clients = entry.second;
            if ( m_changedObjects.find(object) != m_changedObjects.end() )
            {
                Buffer buffer;
                Serializer ser(buffer);

                // Message ID
                MessageTypeParameter p0 = { MESSAGE_UPDATE_OBJECT };
                p0.Serialize(ser);

                // Object ID
                UpdateObjectParameter p1 = { object };
                p1.Serialize(ser);

                // Object State
                const DistributedObjectInterface::ptr& pobject =
                    m_boundObjects[object];
                pobject->Serialize(ser);

                // Cache the state data on relevant clients
                for (auto client : clients)
                    m_clients[client].AddCache(object, buffer);
            }
        }

        // Synchronize object state data cached on each synchronizable client
        for (auto client : synchronizableClients)
        {
            Cache& cache = client->GetCache();
            for (auto entry : cache)
                client->Send(entry.second, false); // unreliable
        }
    }

private:
    typedef std::set<ObjectId> ObjectSet;
    typedef std::set<ClientId> ClientSet;
    typedef std::map<ObjectId, Buffer> Cache;

    std::map<ClientId, Client> m_clients;
    std::map<ObjectId, DistributedObjectInterface::ptr> m_boundObjects;
    std::set<ObjectId> m_changedObjects;
};
```


BIG WARS

owned in-game objects that a Battle Server would need to simulate.

The server system itself is a client/server environment, where each Battle Server is a client connected to the central HUB Server. Within this server-side client/server architecture, each Battle Server acts like an authoritative object simulation client, and periodically tells the central HUB Server in real time what the latest object states are for those objects that are owned, and thus authoritatively simulated on the Battle Server. The central HUB Server maintains the latest copy of all the in-game object states reported in real time from each connected Battle Server. The HUB Server, at a predefined high frequency, then distributes the world object states to all the connected Battle Servers, with the exception of the objects owned by a given Battle Server. Each Battle Server, in turn, distributes the relevant object states to its connected clients, at each server's predefined synchronization rate.

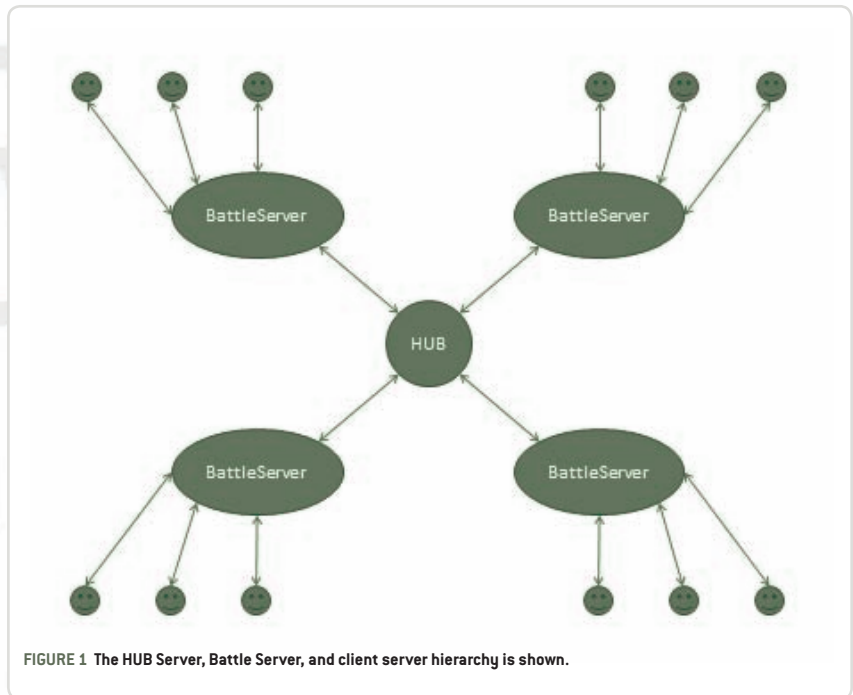
The central HUB Server maintains the object states for the entire world, but does not perform any kind of object simulation. Instead, it merely relays the world object states to other Battle Servers if those objects are not owned, and thus not authoritatively simulated on each Battle Server. Each Battle Server, similarly, will only simulate its owned objects and takes all the other non-owned world object states from the central HUB Server "as is," and then relays the object states to its connected clients.

This way the CPU intensive world object simulation load is distributed to all the participating Battle Servers. The I/O intensive operations are also distributed across the Battle Servers, each of which only handles synchronization with its connected clients. In this new server architecture, the CPU and I/O load for each Battle Server does not increase with the number of clients and objects. Since one game session can now have more objects, the bandwidth consumption and synchronization overhead will increase for each Battle Server (but it's not proportional to the number of clients added). So if each Battle Server can host 32 players simultaneously, and it is theoretically feasible for a central HUB Server to handle 8 connected Battle Servers with a small overhead inside a LAN environment, then we could finally host 256 players simultaneously in a real-time multiplayer game session!

Now that all the high-level concepts are laid out, let's discuss some important implementation details.

//// REMOTE OBJECT SPAWNING

The entire client/server framework is actually a hierarchical distributed object system, where each in-game object requiring some kind of synchronization should have a representation within it. Like all other distributed object systems, there is one master copy of each distributed object on the server side, while there is one proxy



message from the HUB Server, they create an object knowing they are not authoritative on the object (the authoritative flag would be turned off). They will only update the object states from the HUB Server "as is," and will not perform any local simulation on the object. The central HUB Server-side unique object ID is embedded into the object creation message sent out to each Battle Server, and this ID will be used for later state-synchronization.

The object creation requests from the connected Battle Servers and the actual object creation messages sent from the HUB Server are totally asynchronous—the Battle Servers send out the object creation request to the HUB Server and then forget about it. They only assume that an object creation message will be received later so an actual proxy can be created with a HUB Server unique object ID. The object creation requests could either be reliable or unreliable, but the actual object creation messages from the HUB Server to all the connected Battle Servers would always be reliable. An object creation request doesn't have to always succeed based on game logic (a request for spawning a rocket could be lost), but if the HUB Server does receive such a request and creates a master copy, it has to be reliably

reflected on all the connected Battle Servers in a consistent manner; in the end, all the Battle Servers should maintain the same set of objects, so their respective connected clients can have consistent gameplay. If an object creation request is important, the Battle Server should send a reliable request message to the HUB Server.

When the HUB Server receives an object creation request from one of the connected Battle Servers, it simply instantiates a concrete object with necessary parameters, probably through a conventional object factory mechanism. Careful readers will have already concluded that the distributed object system we are discussing features a deterministic object spawning mechanism. Object spawning on the central HUB Server is always by request—there is no implicit object spawning in this system at all!

Object deletion logic resembles that of object spawning in that it is always a Battle Server that sends a reliable object deletion request to the central HUB Server. When the HUB Server processes the deletion request, it deletes the master object, and an object deletion message will then be sent to all the connected Battle Servers so they can properly destroy their proxy copies, and the object deletion message will in

turn be propagated to the relevant clients of each Battle Server. Object deletion requests from any Battle Server should always be reliable since it is an important event that cannot be ignored.

Objects inside the distributed object system must implement the same interface so they can be managed in a uniform fashion (see Listing 2).

The `DistributedObjectInterface` in Listing 2 is the contract that every distributed object must keep in order for them to be successfully registered and managed by the distributed object system.

The `RetrieveCreationParameter` method, when implemented by a specific game object, retrieves the parameters necessary to create a proxy object that has valid starting states from the HUB Server (or a Battle Server, if it is in turn creating the second-level client-side proxy object remotely).

The `Serialize` method is used for automatic object state synchronization. Each concrete game object would implement this method so that a selected subset of the object state could be serialized. See Listing 1 for a typical use case.

The `Bind` method binds the unique object ID to a concrete proxy object after it is created remotely.

The `Invoke` method is used for Remote Method Invocation (RMI), which we will cover next.

BUILD THE FUTURE!

The team that created HALO™ is going to change the world again.

Be a part of it.

BUNGIE

"Create and control some of the most load-intensive server-side game features the world has ever witnessed."

- Bungie Infrastructure

NOW HIRING

Senior Server Programmer
Server Programmer

MORE POSITIONS AVAILABLE:

- Player Investment Designer
- Lead Writer
- Lead UX Designer
- SDET Lead, Server
- Lead AI Programmer
- Senior Simulation Engineer
- Infrastructure Engineer
- Software Development Engineer in Test
- Lead Concept Artist

Send your resumé and/or demo reel to JESSICA OWENS at jobs@bungie.com


WORK AT BUNGIE
GET A JOB AND TAKE OVER THE WORLD
www.bungie.net/jobs

Microsoft and Halo are registered trademarks of the Microsoft group of companies. Portions © 2010 Microsoft Corporation. All rights reserved. Bungie, the Bungie Logo and the "Work at Bungie" logo are registered trademarks of Bungie, LLC. Graphic design © 2010 Bungie, LLC.

BIG WARS

//// OBJECT EVENT PROPAGATION

Besides automatic object state synchronization, there are also many kinds of game object events that need to be distributed across the entire hierarchical distributed object system. The object event propagation is achieved through Remote Method Invocation (RMI) on Battle Servers and clients, and relayed by the central HUB Server.

The RMI system is a natural extension to the traditional Remote Procedure Call (RPC) system that is applied in an object-oriented distributed environment. The RMI system is seamlessly embedded into the broader distributed object system discussed previously, which features deterministic object creation and deletion. Other distributed object systems may have an implicit create-by-call mechanism, but they are not suitable for our needs since we need this kind of determinism.

RMI is always object-oriented in that any remote method invocation is carried out on a distributed object. In local object method invocation, the context of the method invocation is the local object, whether it is a concrete object or a reference. In a distributed environment, however, the context is always remote, so the unique object ID comes into play to act as a remote object context moniker. The way a method is remotely invoked changes from its local form `object.Method(params)` to `server.InvokeRemoteMethod(client, object, signature, params)` or `client.InvokeRemoteMethod(object, signature, params)`. The server version bears an extra client argument because an RMI can be both unicast and multicast, but when a client invokes an RMI, it is always destined to the connected server. There can also be several server-side `InvokeRemoteMethod` variations to support both unicast and multicast RMI, which I leave to the readers to complete.

As mentioned, RMIs can only execute on the Battle Server and client side, while they are only relayed on the HUB Server. RMIs executed on the client side are mainly used for propagating events on the object proxies that are perceivable to the client side, while RMIs executed on the Battle Server side are for those object proxies that are owned by the specific Battle Server. If a client requests an RMI on an object residing on a Battle Server (`client.InvokeRemoteMethod(object, signature, params)`), the receiving Battle Server only executes the RMI if it is authoritative on the object requested. If the Battle Server happens not to have ownership of the object requested, then it would not execute the RMI request locally; instead, it would forward the RMI request to the central HUB Server (`battle.InvokeRemoteMethod(object, signature, params)`). When the HUB Server comes to process this RMI request from the Battle Server, it searches its object ownership registry and locates the owning Battle



Server of the object in question. Then the HUB Server relays this RMI request to the owning Battle Server (`hub.InvokeRemoteMethod(battle, object, signature, params)`). When the Battle Server that is authoritative on the RMI-requested object receives the RMI request, it executes the RMI locally, which finishes the entire calling chain. The object states that were updated on the executing Battle Server, which is authoritative on its owned objects, will be synchronized to the HUB Server in the next Battle Server-to-HUB Server synchronization tick. Those object updates will then be propagated to other Battle Servers in the next HUB synchronization tick.

For illustration purposes, I simplified the prototype of `InvokeRemoteMethod`, which, in a production codebase, utilizes advanced C++ templates and Boost.Function features to achieve a typed variadic parameter marshaling.

Like remote object spawning, RMIs are also completely asynchronous and one-way. They are used to trigger events without concern for possible return values. The result of an RMI to the server will be reflected back to the client through object state updates.

At this point, a good question may have arisen from curious readers: would the added hierarchy on the server side introduce extra lag? The answer is unfortunately yes. The server system would preferably be configured in a LAN environment using high capacity network connections, so the communication latency between the HUB Server and the connected Battle Servers would be negligible (around 1 millisecond [ms]) compared to the communication latency between a Battle Server and its connected clients. The major latency actually comes from the synchronization frequency for the communication between the HUB Server and the Battle Servers.

A Battle Server synchronizes its owned object states to the HUB Server at a predefined frequency, and this frequency is interlocked with the Battle Server's tick rate. It is practical to have

a Battle Server synchronize its owned object states to the central HUB Server at a rate of 20 times per second. This means that the latency in reflecting an object state change on the owning Battle Server to the central HUB Server could be up to 50ms. A HUB Server would also distribute the world update states to relevant Battle Servers at a rate of 20 times per second, so the latency here could add another 50ms. So in the worst case, the server cluster would incur an extra 100ms of delay in addition to the normal client-to-Battle Server communication latency experienced by players.

The biggest impact of any lag is on object interaction across physical Battle Server processes on the client side. Here are some typical scenarios:

A Battle Server-owned client player shoots a remote player that is connected to another Battle Server. The remote player under this scenario merely appears to be a client with some extra 100ms latency. There are existing server-side lag compensated hit detection techniques (like the one found in COUNTER-STRIKE) which would apply here. The tricky part is that we now have to deal with the hit in the hierarchical server environment.

When a local client player fires an instant hit weapon (including those weapons that fire bullets which could take some time to travel, but follow a deterministic path), the first level hit detection would be carried out on the player connected Battle Server. The Battle Server detects the hit by rewinding the object states it maintains back in time, taking into account player lag and the lag incurred by the server hierarchy, together with interpolation techniques. If the Battle Server detects a hit, but the hit target is an object that is owned by another Battle Server, the current Battle Server would send a hit RMI with necessary firing information to the HUB Server, which will in turn forward the hit request to the owning Battle Server. The Battle Server that owns the hit target would perform the same check, and if the hit is on the

owned object, the respective object states will be updated (force applied, health decreased, etc.), and the updated states will be propagated to other Battle Servers on the next synchronization tick.

A Battle Server-owned object “bumps” into a remote object owned by another Battle Server. Since the original Battle Server does not own the bumped target, it sends a “bump” RMI to the central HUB Server. The request is then forwarded to the owning Battle Server, which in turn processes the bump message in its authoritative physical environment. The bump target will then be updated if the bump is actual. Because a bump is bilateral (object A bumping into object B is also object B bumping into object A!), the original Battle Server would also receive a bump request from the remote Battle Server. It will then update the relevant object state should it decide a bump actually happened based on its local object states.

A Battle Server-owned C4 charge explodes. On the owning Battle Server, it adds force to all the locally owned objects that are in the range of the explosion. For those objects that are not owned by this Battle Server but are also within the explosion range, it sends an explosion RMI on each of those potentially affected remote objects to the central HUB Server. The HUB Server then forwards the requests to their respective owning Battle Servers. Each owning Battle Server would then apply respective explosion effects to the affected objects (adding forces, etc.).

Of course, the above case analyses may not be 100 percent complete, but they at least give a good starting point on object interaction problems in the hierarchical server system, with the concern for added communication latency.

NOTE: Remote objects are not simulated at all on local Battle Servers and clients, so all remote objects appear in the local physical environment as “static” objects—they move according to the updates from a remote server. On the client side, it only performs a physical simulation on client controllable objects, and only does interpolation (no extrapolation at all!) on remote objects. By not simulating those remote objects locally on a Battle Server, we lower the CPU power consumption.

///// EXCHANGING OWNERSHIP

Now let’s briefly discuss how object ownership transfers from one Battle Server to another. In a game session, when an object owned by one Battle Server is somehow bound logically to an object owned by another Battle Server (for instance: a player connected to a Battle Server picks up a weapon dropped by another player connected to a different Battle Server), the object

ownership is transferred to the binding object’s owning Battle Server to minimize latency and inconsistency issues.

Ownership requests are a special type of system message that are always processed on the HUB Server exclusively. Any Battle Server can claim ownership of a remote object through the HUB Server if the Battle Server’s game logic sees fit. The HUB Server would unconditionally respect the request and update the ownership registry. It would then send an ownership change message to the old owning Battle Server and the new owning Battle Server to confirm the ownership transfer. It is the game logic’s responsibility to prevent object ownership competition, and whichever Battle Server claims ownership of a remote object first is assigned ownership through the HUB Server.

///// SERVER START UP

Finally, we can discuss how this hierarchical server cluster could be configured and bootstrapped, and a typical sequence for handling client connections.

We need to first understand that the server architecture we are discussing is still designed for short-sessioned real-time multiplayer games. We are lucky in that we don’t have to deal with issues such as dynamic load balancing, fault tolerance, and other concerns that are fundamental to normal MMO server architectures.

With that in mind, the server cluster startup sequence is fairly straightforward:

- Due to the added complexity, it is preferable for the server cluster to be started by a master lobby server, rather than being started up by individual players.
- There is a server spawner process residing on each physical server machine in the server farm, and all of them are connected to the master lobby server.
- The master lobby server maintains information on which physical machine is free to spawn a new server process through information advertised by each connected server spawner process.
- Game clients always connect to the lobby server first, and a client can decide to start a new game session with a HUB Server involved.
- Per client request, the lobby server picks a free server machine based on the information advertised by the connected server spawner, and first notifies the spawner to start a central HUB Server on a preconfigured port.
- The lobby server then, based on the total number of players requested in a game session and the maximum number of physical clients that can be supported for each Battle Server, picks the right number

of free server machines, and requests them to spawn one Battle Server on each machine. At the same time, the lobby server, through the respective spawners, gives the HUB Server’s IP address and port number to the newly spawned Battle Server processes, so each Battle Server could then connect to the central HUB Server.

- The lobby server then advertises the least connected Battle Server IP/port information to each new client requesting to join the game session, so new players will be added to the game session in a round robin fashion (i.e. static load balancing during bootstrapping).
- The lobby server will coordinate the game session start once all the connected clients are ready.
- The spawned Battle Servers will communicate server status to the lobby server through the server spawner residing on the same server machine, while in-game messages are only communicated among connected clients and the central HUB Server.

This concludes the server cluster startup sequence in a typical use case; other sequences are possible based on specific needs and configurations.

///// PUSHING THE LIMITS

At the beginning of this article, I mentioned client-side authoritative simulation as an alternative means to offload server load (with pros and cons). It is actually feasible to combine client-side authoritative simulation and the server architecture discussed in this article to potentially achieve even better results, but I’ll leave the brainstorming to interested readers.

As you can see, with the innovative server architecture discussed in this article, we could theoretically achieve a much higher number of simultaneous players in a real-time multiplayer game session. I hope this article will provoke some brilliant ideas to further push the limits of real-time networking in games, just as we have always pushed the limits of graphics, AI, animation, and physics. ☺

LIN LUO has many years of professional experience in the field of multiplayer networking programming. Email him at lin.luo@live.com.

RESOURCES

Unreal Networking Architecture

<http://unreal.epicgames.com/Network.htm>

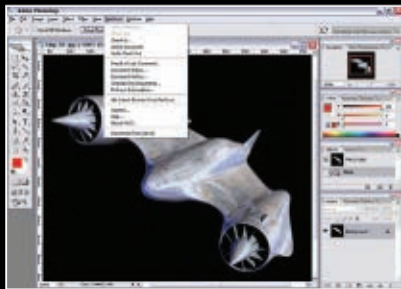
Source Multiplayer Networking

http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

Perforce | Fast Software Configuration Management



Introducing P4GT, a productivity feature of Perforce SCM.



P4GT

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

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

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

PERFORCE
SOFTWARE

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

All trademarks and registered trademarks are property of their respective owners. Adobe screen shot reprinted with permission from Adobe Systems Incorporated.

HOT

FAILURE

CHRIS PRUETT

TUNING GAMEPLAY WITH SIMPLE PLAYER METRICS

THERE'S NOTHING LIKE WATCHING SOMEBODY ELSE PLAY YOUR GAME. Over the course of development, you've played the game daily, and have, perhaps unconsciously, developed a particular play style. But putting your work into the hands of a novice gives you a chance to see what happens to your design when it's played without the benefit of daily practice. Every collision pop, animation snap, confusing tutorial message, and intermittent bug seems amplified when a beginner plays. No matter how much you polish or how many bugs you fix, your play style and intimate familiarity with the content can bias you away from problems that other users will immediately encounter. That is why playtesting is a vital part of making a good game. In order to truly get the most from playtesting, you're going to have to take some data from these sessions—this article chronicles my experience with gathering gameplay metrics.

////// STARTING SIMPLE

I got my start in the industry writing Game Boy Advance games. Back then, our idea of playtesting was pretty straightforward: we would get some local kids to come in, hand them a special GBA that was hooked up to a VCR, let them play for a bit, and then go back and review the tapes. This procedure yielded immediate, dramatic bugs. Areas that the team took for granted were often sources of tremendous frustration for our testers. When a member of the target audience fails continuously in a specific area, it is usually a clear message that something needs to be fixed. A couple iterations with real live kids and the side-scrollers we were making would be vastly improved.

Nowadays, I work on and advocate games for Android phones. My first Android game, *REPLICA ISLAND*, is a side-scroller, not so different from the GBA games I was making ten years ago. But some things have changed: I'm no longer working for a game studio; I wrote *REPLICA ISLAND* on my own,



with the help of a single artist, mostly on my free time. I also no longer have access to a pool of young playtesters, and even if I did, my target audience is a bit older. Finally, there's no easy way to record the output of a phone while somebody is playing—the only way to really

see what's going on is to stand over their shoulder, which is awkward and can influence the way the tester plays.

What is an indie phone game developer to do? As I reached feature completeness for *REPLICA ISLAND*, I realized that I really had no way to guarantee that it was any fun. The game had been developed in a vacuum, and I needed to get more eyes on it before I could feel confident releasing it.

The first thing I tried was user surveys. I put the game up on an internal page at work and sent out an email asking folks to play it and give me feedback. I even set up a feedback forum with a few questions about the game. This approach was pretty much a complete failure; though many people downloaded the game, very few (less than 1 percent) bothered to fill out my five question survey. Those who did fill out the survey often didn't provide enough information; it's pretty hard to tell if "game is too hard" indicates a failure in the player controls, or the level design, or the puzzle design, or the tutorial levels, or what.

THINKING ABOUT METRICS

After that setback, I remembered reading about the player metrics system Naughty Dog developed for the original *CRASH BANDICOOT*. The system wrote statistics about play to the memory card, which could then be aggregated offline to find areas that took too long or had a high number of player deaths. These problematic areas were reworked, and the data was also used to tune the dynamic difficulty adjustment system in that game. One of the most interesting principals that fed into the design of this system was Naughty Dog's idea that the game over screen must be avoided at all costs. Their end goal was to remove "shelf moments," moments in which the player got stuck and could not continue.

I thought this was a pretty cool idea, but I wasn't sure how feasible it would be on a phone. I asked around a bit to see what the current state of metrics recording is on big-budget games, and found that many companies have some way to report statistics about player actions. Several people told me that while they collect a lot of information, they have trouble parsing that data into results that suggest specific design changes. On the other hand, some studios have tools that can recreate a player's path through a level, and produce statistics about which weapons users prefer, which enemies are particularly tough, and which parts of the map are particularly visible. It seems that collection of player metrics is applicable to a wide variety of games, but that it only benefits the studios who also take significant time to build tools to crunch all the data that they collect. (For an example of how this kind of system can be taken to the extreme, see Goerg Zoeller's talk about the crazy system they have at BioWare.) It turns out that collecting the data is the easy part—rendering it in a way that is useful for designers is much harder.

That sounded discouraging, as my goal was to keep my tool chain as simple as possible. But I decided to experiment with some metrics recording anyway, starting with just a few key metrics. My Android phone didn't have a memory card, but it did have a persistent internet connection. Maybe, I thought, I could log a few important events, send them to a server, and get results from players that way. My goal was to try to understand as much as possible about my players while keeping the system as simple as possible.

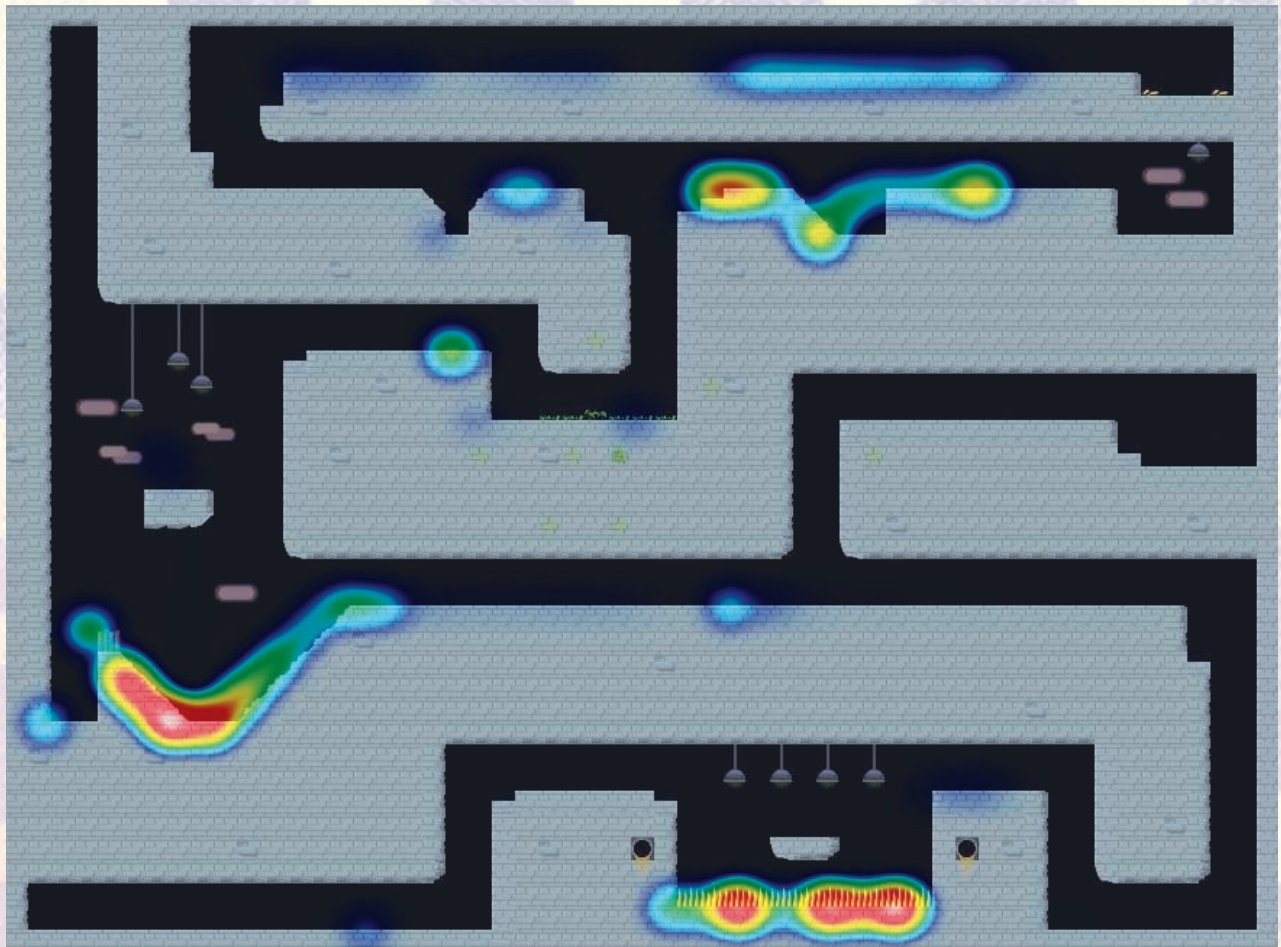
THE BASIC SYSTEM

The event logging system that I wrote has three parts: a thread in the game runtime that collects player events and sends them to a server; the server itself; and finally a tool to parse the data recorded by the server. "Server" is a strong word in that second component. My server is actually a PHP script that, in about 30 lines of code, validates the HTTP Get query it is sent and writes the results to a MySQL database. The query itself is dead-simple: it's just an event name, level name, xy location, version code, session id, and time stamp. These fields are recorded to the database verbatim. The actual processing of the data is also done in PHP (a poor choice, in the long run; more on that later), though only on demand when a special dashboard page is loaded.

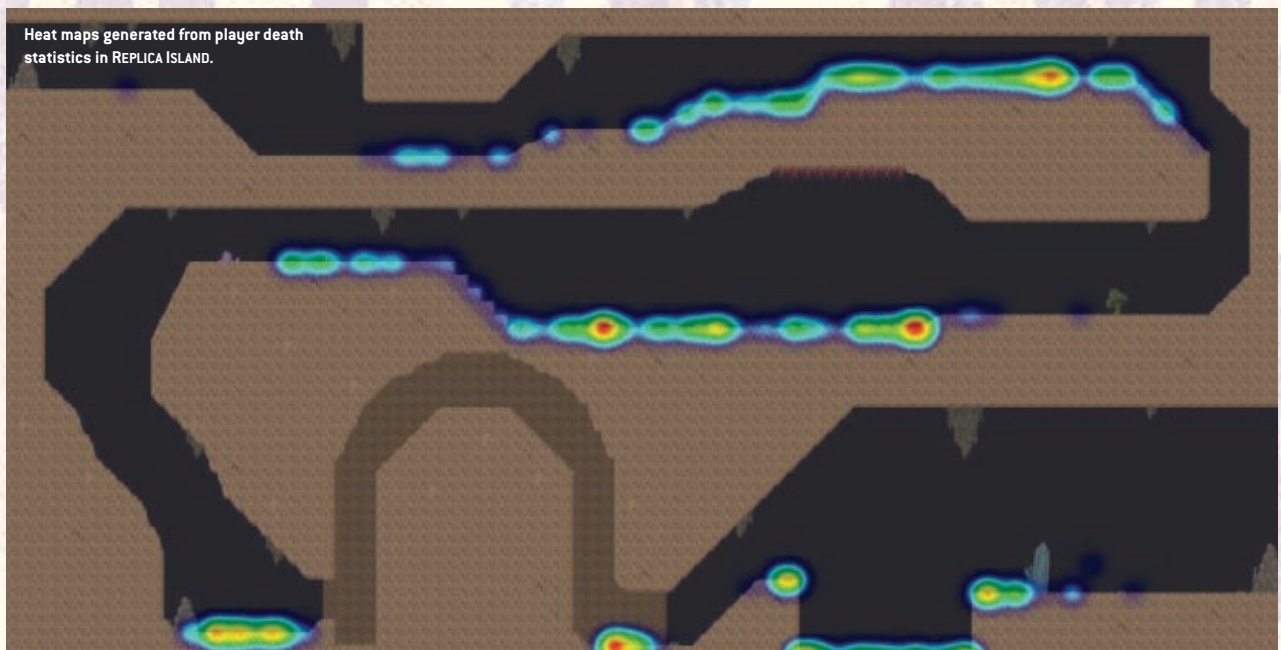
I started with just two events: player death and level completion. Each time a player dies or completes a level, the game reports that event to the server. From this data, I was able to construct a pretty detailed overview of the game flow. I could see which levels took the longest, which had the most deaths, and which were unusually short. By dividing my values by the number of unique players, I could also see what percentage of players died on certain levels, and the average number of deaths for each player. By looking at the spatial location of the event, I could tell the difference between a death from an enemy and a death from a pit. As a first-pass implementation, my simple metrics system proved to be pretty detailed.

HIGHLIGHTING FAILURE IN BRIGHT RED

Once I had the basic reporting system up and running, I released an update to my testers and watched the data flow in. Very quickly, patterns emerged; there were some levels where almost 100 percent of players



Heat maps generated from player death statistics in REPLICA ISLAND.



died at least once, and other levels in which players were getting stuck for hours (indicating a pretty major failure for a level designed to take five minutes). Just by looking at the numbers, I had a clear picture of which levels needed the most work.

But identifying problematic levels wasn't enough. Sometimes I couldn't tell why a particular level was a problem.

So I went a step further. Using the same data, I wrote a tool to plot the death positions on top of the level art so that I could see exactly where users were dying (and where they were not). The first pass of this system just drew a little dot on the level art when a player died, but once the number of players grew to be large, I switched to rendering heat maps of death locations over the levels, which was much easier to read (see sidebar).

GAME DESIGN FAILURES AS OBJECT LESSONS

The combination of high-level play statistics and plotted death locations was illuminating. I learned, for example, that a huge number of players were dying at the very first enemy. This was not because the enemy was particularly hard; after considering the problem, I realized it was because the enemy appeared in a spot where the main attack—a crushing butt stomp, performed from the air—was difficult to accomplish due to a low ceiling.

I also learned that my simple dynamic difficulty adjustment system needed adjusting itself. This system secretly increases the player's life and flight power after a certain number of consecutive deaths, and by looking at the data, I could see that it needed to kick in a lot earlier.

I also made sweeping changes to my level geometry. I had a few levels with very high completion times but very few deaths, and I realized that players were simply getting lost. I reworked these levels to make the paths through them clearer; in one or two cases, I scrapped an entire level and made a new one from scratch.

But the biggest problem that I identified was with pits. REPLICAS ISLAND is a platformer, and as you can guess, it involves a lot of jumping over pits. But unlike certain spinning marsupials and pipe-dwelling plumbers, my character's main mode of transport is flight. I needed a control system that did not require a D-pad, so the protagonist in REPLICAS ISLAND, the green Android robot, flies using rocket thrusters on his feet. The basic movement model involves getting momentum up while on the ground before jumping into the air and using that momentum, along with the thrusters, to fly around. The thrusters run out of juice quickly but refill when you land, so the idea is that a player will jump into the air and then carefully expend his fuel to reach distant ledges or line up a precision butt stomp.

All that is well and good, but when I looked at the death data coming back from my playtesters I found that they were dying in bottomless pits en masse. Drove of players were falling down even the smallest of holes. And of even greater concern, the death-by-pits numbers did not decrease over the course of the game; players were not getting better at making jumps as time went on.

With this information in hand, I reviewed my core game and level design and came up with a number of theories. The basic problem, I decided, was that players could not see the pits they were jumping over. First of all, there was no visual indication that a pit of death is a pit of death; since my levels are often very tall, it's hard to tell which pits lead to some underground level segment and which lead to a grisly demise. Second, and most important, my camera was not doing a good enough job of keeping the floor visible when the player jumped into the air. Almost as soon as the player leaps into the air the ground would scroll off the bottom of the screen, making it hard to judge where to land.

Master platformers like SUPER MARIO BROS. almost never scroll vertically; Mario has a whole set of complicated rules dictating which specific circumstances allow the camera to move up and down. In REPLICAS ISLAND, however, the flight mechanic meant that I had to allow vertical

HOW TO MAKE HEAT MAPS

Generating heat maps isn't hard, but information on the exact procedure can be hard to find. I used a method similar to the one described here: <http://blog.coronet.com/how-to-make-heat-maps>

The basic procedure is as follows:

- ✕ Prepare a grayscale image of a circle that goes from black in the center to transparent on the edges in a radial gradient. This is your event spot image.

- ✕ Prepare a color gradient image. The bottom should be white, or red, or whatever color you choose to indicate "most intense" on the heat map. The top of the image should be black, and with several other colors in between. This image will be used as a lookup to colorize your output later.

- ✕ Generate a list of event positions.

- ✕ Calculate the maximum number of overlapping data points (i.e. the number of events that occurred at the most common xy position). This is the value of maximum heat.

- ✕ For each unique location on the event list, draw the spot image to a canvas at the location of the event. Draw the image at $[(\text{number of events at this$

location] / [maximum heat] * 100%)) opacity. Use the multiply transfer mode (src * dest) to blend each spot to the canvas.

- ✕ When finished, you should have an image with a bunch of black spots on it of varying shades of darkness. This is the intermediate output image.

- ✕ Take the output image and remap its color table using the gradient image. Take the alpha value of each pixel and use it to look up a Y offset in the gradient image to find the color value for that pixel.

- ✕ Take the resulting image and blend it over your level art. The event hotspots in the level will be shown as colored areas, with the intensity of color increasing in areas where more events occurred.

When doing this work, make sure that you keep your color space (particularly the opacity calculation in step 5) within regular 8-bits-per-channel ranges (or consider using a format that supports floating point pixels). It is easy to introduce precision bugs that will only manifest when there are so many data points that the contribution of a single event falls below 1 percent. Tools like ImageMagick (see Resources) can help you do this.

scrolling in the general case. After a bunch of tweaking, I came up with a smarter camera that does not begin to scroll vertically unless the player is close to leaving the visible space themselves.

After making these changes, I shipped another update to my beta testers and compared the results to the previous version. The deltas were very reassuring; deaths were down overall, level completion times were, for the most part, back into normal ranges, and pit deaths dropped by a pretty huge margin. I iterated several more times with these testers before I was ready for release, but with the metrics reporting system in place, it was easy to see whether my changes were having an influence on how my testers were playing.

HELLO WORLD

After several iterations with my test group, my graphs started to align to the bell curve I was looking for. It was time to ship the game, and I decided to leave the metrics system in place. I wondered if the data I collected from live users would look different from the data produced by my test group. There was only one way to find out.

Of course, any time an app reports data back to a server, it's best to let the user know about it. The first time REPLICAS ISLAND is launched, a welcome message appears that details the latest game improvements. That message also informs the user that anonymous, non-personal play data will be uploaded to a remote server in order to improve the game, and that players who do not wish to participate may turn the reporting system off in the options menu. This approach seemed like the best solution: though the code is open source and anybody can look at the content of the data packet itself (and I ensured that nothing about the metrics data can be tied to any specific user or device), allowing users to opt-out gives them an opportunity to say "no thanks." By comparing my Android Market installs with the number of unique users reporting in, it looks like less than 20 percent of my users chose to opt out of metrics disclosure.

As a result, I have a huge amount of data now—over 14 million data points, close to a gigabyte of event information generated by my user base (which, as of this writing, is about 1.2 million players). In fact, the volume of data broke my data processing tools pretty quickly; I have a snapshot of statistics from the first 13,000 players (which I have published on the REPLICAS ISLAND website—see Resources), but after that, a lot of my tools failed. The good news is the first 13,000 players produced aggregate data that was very similar to the smaller test group, which probably means that the test group results can be applied to much larger groups of players.

SOMEHOW, THIS PLAN WORKED OUT

I have been extremely satisfied with the event reporting system in REPLICAS ISLAND. For very little work, almost no cost (the server back end that records events costs less than an Xbox Live account), and using only two types of events, I was able to quickly and effectively identify areas where players were having trouble. Furthermore, once I started collecting this data, I was able to compare the aggregate result of my metrics between versions, which made it easier to see if my design changes were effective.

Using PHP and MySQL as my back end server language was a good choice; the actual recording of events is so trivial that I'm sure any language would have worked, but with PHP, the whole server took less than 30 minutes to put together.

Using a separate thread to report events from the game was a good move as well. I didn't want any sort of UI to block HTTP requests, and moving the web communication to a separate thread made sense, but I initially had some concerns about overhead. I needn't have worried; the overhead is so small, I can't even get it to show up in my profiler.

Finally, keeping the system as simple as possible was a really positive decision. I considered a lot of potential event candidates, but for my game, tracking player death and level completion provided more than enough information. More statistics would have complicated the processing of the data, and possibly made it harder to reduce the feedback to a concise view. Now that I've had some experience with automatic metrics reporting, I'll probably increase the volume of data that I send back in the future, but starting simple was definitely a good move.

BUMPS ALONG THE WAY

Not everything about the event reporting system worked out well, however. I made a few decisions that ultimately turned out poorly, or just wasted time.

The decision to use PHP for the reporting server was a good one. It was a mistake, however, to use PHP to do the processing of the data. My idea had been to do everything via a web dashboard (I even wrote my level editor in PHP and Javascript), but PHP fell down hard when the amount of data I needed to manage exploded. PHP runs in pretty strict memory and speed requirements, and I found myself hacking around these limitations almost immediately. Once I passed 20,000 users, most of my PHP-based tools simply stopped working.


Bitmap processing was particularly painful in PHP. I did all of the heatmap generation in PHP, but I should have just written something that could run locally instead of on a web server. I ran into a number of bugs in the PHP GD interface (compositing bitmaps with alpha is pretty broken), and ended up having to reduce the size of my level art images in order to do the processing. For this article, I rewrote this tool using Python and ImageMagick, and the results are far superior. I've provided the code for this implementation, which can be found at www.gdmag.com/resources/code.htm

Finally, though this data tells me all about where players die and how long it takes them to complete levels, it doesn't help me identify shelf moments that are not related to death. I ended up shipping with a few key level design failures that my metrics never caught; in the most egregious case, players get stuck at a puzzle where they do not understand how to progress, and end up giving up before they complete the level. This never shows up in my metrics because an event condition is never reached; I only learned about it when users started complaining about being stuck in the same spot. Automatic metrics are super useful, but they can't show you a complete view of the game. In my case, the metrics were good at finding problematic level layouts but were particularly ineffective at identifying design failures related to rule communication.

THE FUTURE

For my next game, I'll definitely employ automatic metrics reporting again. In addition to death positions, I may add events based on different forms of death; it'd probably be useful to know how exactly a player died, not just where. And, depending on the game, it might be useful to report a history of positions before the death so that an individual player's path through a level can be traced. However, the key to this kind of system is simplicity; collecting data isn't useful unless I also have reliable tools to process it later. For the next title, I'll probably leave the basic reporting and storing mechanism alone and focus most of my time on writing better tools for crunching the numbers.

I'm also wondering whether aggregated output from this form of player metric can be used to inform runtime dynamic difficulty systems. If the game were capable of reading aggregated output back from a server, it could change itself based not only on the play of a single player, but on the average habits of millions of players. The availability of this data opens up all sorts of interesting possibilities.

Player metrics are not a perfect replacement for user testing, but they are a pretty useful approximation. And because they allow you to test a much larger group of users than would be possible with individual testers, metrics can tell you more about your game in the long run. The cost to benefit ratio was extremely positive for REPLICAS ISLAND; by keeping the runtime and server dead simple, I learned much about my level designs and the habits of my players, and my game got a lot better as a result. My only regret is that I did not employ this kind of system on earlier games—it seems applicable to pretty much any genre on pretty much any platform. 

CHRIS PRUETT is a game developer advocate at Google, focused on Android. Under the cover of night he writes indie games and blogs about horror game design. The views expressed in this article are his alone and not those of his employer. Chris lives in Yokohama, Japan with his wife and daughter.

R E S O U R C E S

Goerg Zoeller's BioWare telemetry talk

<http://gdc.gulbsoft.org/talk>

Replica Island player metric snapshot

http://replicaisland.net/index.php?view=en/player_metrics.php

ImageMagick

www.imagemagick.org





postmortem

SINGULARITY

BY ROB GEE, BRIAN RAFFEL, STEVE RAFFEL, GUSTAVO RASCHE, DAN VONDRAK, AND JON ZUK

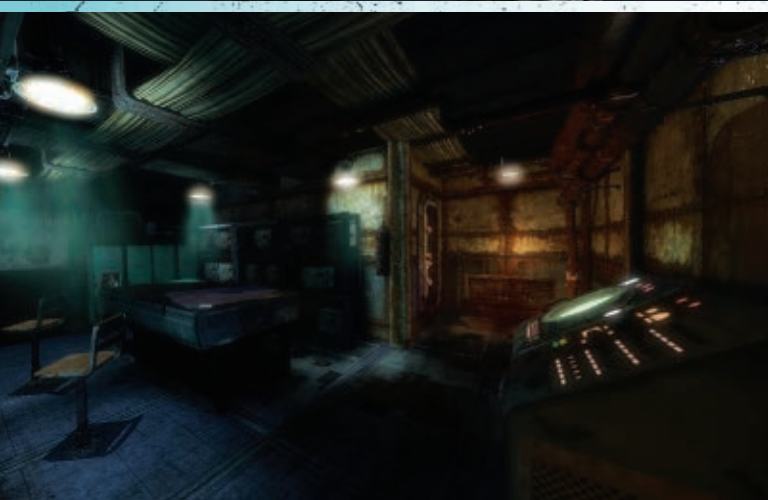
AFTER WORKING FOR YEARS on licenses like *Star Wars*, *Star Trek*, and Marvel, the team at Raven Software decided it was time to return to our roots: to create an original IP as we had with games like *HEXEN*, *HERETIC*, and *SOLDIER OF FORTUNE*. A new license is not an easy proposition due to the much larger budget. Today's games often require teams of more than 70 people;

compare that to the less than 15 people we had when Raven first began. We knew we would have to create a great prototype to convince Activision to give us our shot, and that this would take some "under the radar" work. We put together a small team and had them work for three months behind the scenes, creating a demo that encapsulated the core concepts of

SINGULARITY. When it was done, it looked better than we had hoped. We were so confident in the prototype that we set up a meeting with upper management and sat them down to demo our vision. We turned off the lights and showed them the game. There was nothing but silence. We thought it was all over; our hopes for creating a new IP were dead, and Activision management was going

to lynch us for wasting their time. But when the lights came back up, we were surprised to find that they loved it—they wanted to see this game get made. We left Activision confident the toughest part was over. Naturally, that wasn't the case. As we began our journey creating *SINGULARITY*, we discovered a new IP brings out challenges from all directions. >>>

SINGULARITY



[WHAT WENT RIGHT]

1) FOCUS ON GAMEPLAY WITH REUSE OF ASSETS. The initial SINGULARITY prototype team comprised a small number of experienced developers along with a group of enthusiastic new hires working together under a tight deadline to prove out a new game concept. This smaller team structure inspired better communication and allowed for quicker feedback and iteration time. Almost daily reviews allowed us to track our progress, and gave everyone on the team a chance to contribute ideas and provide constructive critiques. The smaller, more agile team allowed us to try things that we wouldn't have been able to do with a larger team, like changing direction easily and quickly.

At the center of SINGULARITY is the TMD (or Time Manipulation Device) and the time shifts that transport the player between the past and present. The TMD primarily allows the player to control the age of specific objects in the game. These ideas were the genesis of the game and came on-line early in the design and development prototype. It was our main "hook," and in order to prove out the mechanics for it in terms of puzzles and combat, we decided we needed to create examples of each TMD function in order to get a handle on the final gameplay possibilities. These functions included removing barriers to create new paths, discovering hidden assets (that old rusty barrel could be "renewed" and used in battle), and using time manipulation directly against enemies in combat.

The short development time required that we be creative when using pre-existing assets from previous Raven games, and to really pick our battles when it came to creating new assets and code. We chose an engine we were familiar and comfortable with as well as models and textures we had created in previous games to quickly dive into gameplay creation. A large gun from another project was used as a TMD stand-in and music from appropriate film scores was inserted as placeholder to set the mood. Most of the new assets were created to demonstrate the effects of the TMD and the different time periods on the environment.

2) SWITCHING THE PERSPECTIVE BACK TO FIRST-PERSON. SINGULARITY shipped as an FPS, and the initial demo was an FPS as well, but at a certain point, the decision was made to investigate the game's potential as a third-person shooter (see the what went wrong section for more on this). After working with various third-person camera systems, alongside art assets and gameplay design, as well as animation, rigging, and so forth, we decided to return to the first-person camera system used in the prototype. This decision was a hugely positive change, allowing us to greatly reduce the number of development challenges we faced. It also allowed us to take advantage of our collective decades of FPS development experience, while focusing on a smaller set of robust features instead of a wider range of shallower features. With the challenges of creating and shepherding a brand new IP, defining and developing a new range of time manipulation gameplay, learning a new engine and tools (Unreal), and ramping up a third development team at Raven, the challenge of making a third-person game on top of everything else was just one risk too many.

The first-person camera returned the focus of the game back to the TMD, instead of on the main character. We figured that the best way to use our limited development time was to reduce our scope and depth while achieving higher quality. We stopped spending time trying to perfect a third person camera or a main character with a full set of interactive animations. Instead, we put our efforts into presenting the TMD as our main character, and put more attention and detail to time-manipulating dynamic objects and enemies, environmental time shifts, and creating a completely immersive environment to bring the setting and back story of SINGULARITY to life.

3) SHORT TERM GOALS WITH PURPOSE. The first playable build is generally where everything is figured out—all the main features of the game are functional, and a vertical slice of the game can be shown in a

polished state. From a solid first playable build, everyone should know to “just do more of that” and build the rest of the game. One issue with getting to that milestone is the distance from the prototype phase. When a project is longer than two years, six months can pass before the game needs to be shown again. To keep the team moving forward, we needed to set some short term goals.

The first thing we did was sit down and hash out a plot for the rest of the game and nail down what levels we wanted. The team was forced to wait this out for a week or two but it was an important foundation. From there, we split the team into two separate groups and gave each of them a level with a goal to have it planned and blocked out in two weeks, refined in another two, and ready for polish in a few after that. We thought that this would flow gracefully into the next set of levels. The plan from there was to choose the best section of the game, add a little polish, and show off our vertical slice. Although production had a long term plan for the whole project, we kept the team thinking more about the next few weeks ahead of them.

As we worked through this plan for the next few months, an opportunity arose to possibly get a major magazine cover. This was great because it would let people know what the game was about, build buzz, and generally just get the game into people’s consciousness. Because we had the

team operating on short term goals, it was easy to adjust and insert this new goal into the pipeline.

We decided to continue working on normal production but peeled some people off to add polish to the Freighter level for our presentation to the magazine. We worked for a solid four or five weeks to tighten up game mechanics, optimize the level, work on the HUD, and more. At the end of that time period, we presented the game to the magazine and found out a few weeks later that we would get the cover. Having that short term goal with a solid purpose (get magazine cover) really spurred the team into action.

4) FOCUS ON PLAYER EXPERIENCE. When SINGULARITY went into its final phase of production, we focused on the player experience, making sure the player felt attached to the world around them. We removed almost all third-person cutscenes, and replaced them with first-person moments. It was important to us that the player felt the story and events were happening to them.

To that end, we removed the main character’s voice entirely. The voice was in the game for some time, but there were a couple problems with this. The biggest was the “disembodied voice talking to you” factor. It was hard for people to understand that it was the main character talking, and we’d see focus testers looking around as the voice played, wondering where it came from—we even caught ourselves

doing it from time to time. Plain and simple, it was just awkward.

We also decided to insert bigger action moments to help pace out the slower story-driven areas and TMD puzzles areas. These included areas like “the escape” at the end of the first level, where your hands are bound and you’re being hunted down. There were other areas like the “Rescue Kathryn” chase, “the Grom” (a big boss) throwing you around in a train car and fighting him on the bridge later—all of these were created so we could spike our intensity pacing at select points in the game.

5) FPS VS. THIRD-PERSON MULTIPLAYER. SINGULARITY’s multiplayer pits soldiers against creatures. One of our goals was for the creatures to control and play as though they were in a third-person action adventure game, while the soldiers would play in typical FPS fashion. The creatures had a pulled out third-person camera, melee moves, and special attacks. In essence, we were pitting two totally different game types against each other in asymmetric combat.

This was a challenging undertaking with a fair share of hurdles, but the result is fun and unique and in the end, proved to be successful. The main contributor to this success is the fact that part of the production team had just come off a third-person action game—WOLVERINE: UNCAGED. We were able to leverage a good amount of existing





SINGULARITY

technology and expertise from that project, including several important tools for creating character moves and melee attacks, to quickly get a proof-of-concept up and running. This allowed us to quickly gauge how successful this feature would be.

From there, we went about actually designing the game mechanics and match modes. Creatures were picked from the single player portion of the game. Each creature is unique, so a great deal of thought went into what defines the class, and what its abilities and attributes would be.

On the FPS side, we designed four soldier classes to each have their own TMD power. Considering the asymmetric nature of the combat, and how hard (or impossible) it would be to achieve true balance, we decided early on to go with a two round match. The teams switch sides from round to round, giving players a shot at each scenario. This allowed us some wiggle room in terms of how truly balanced the factions are against each other. The biggest fear was that no matter how balanced we were able to make it, the chance would still be there for players to find exploits and tip the scales toward one faction or the other. The two round matches eliminate that chance, and keep the game fresh and interesting.

[WHAT WENT WRONG]

1) NOT ENOUGH FOCUS ON STORY OR ADDITIONAL GAMEPLAY BEYOND PROTOTYPE. Since the prototype was primarily intended as a proof of concept of the TMD, not as much time as we had hoped was put into developing the story and supporting fiction beyond some general points. We knew what type of gameplay we wanted and that the game would involve a storyline that jumped between two time periods, but we were too short on specifics such as characters, creatures, key locations, etc. This made for a difficult transition into pre-production as people came off other projects and SINGULARITY ramped up rather quickly.

Also, when we had the main character Renko's voice in the game, it was really difficult to find the right voice actor, and to nail the dialogue. We tried a number of variations; Renko always seemed to come across in a negative way—too funny, too cocky, too serious, or too whiny. Once we removed his voice, we had the new challenge of rewriting the dialogue for other characters, since they had no one to play off. But overall, dropping his voice really helped the player experience and made the moments in the game more impactful to the player.

2) INDECISION AND BLURRY VISION. SINGULARITY was not only an important title for Raven—Activision had a big stake in the game as well. There were many different ideas from all sources in terms of what would make a triple A hit. As mentioned in "right" number 2, one thing that came out of early

FEEDER BOSS
RENDERED FROM
A CHART AND IN
THEORY PROTECTED BY THE
FEEDER BOSS



GAME DATA

PUBLISHER Activision

DEVELOPER Raven Software

NUMBER OF DEVELOPERS 65 core team, 50 additional

LENGTH OF DEVELOPMENT 4 years

RELEASE DATE June 29, 2010

SOFTWARE ZBrush 3.5, 3DS Max 2010, Photoshop CS4, CrazyBump, MotionBuilder, Maya, Vegas Movie Studio, Visual Studio 2008, Perforce 2009

PLATFORM Xbox 360, PS3, and Windows PC

focus testing was the idea of making SINGULARITY a third person shooter. Other popular games had moved in that direction, so we took a long look at it.

When we finally got a prototype of the game running this way, it was accepted internally and at Activision, but there were still some questions regarding whether or not this was the right move for the game. We had a gut feeling that sticky cover wasn't the right "fit" for SINGULARITY, but at the time, we weren't exactly sure why.

We knew it felt better to play our original prototype than our current game, and we began to compare differences. It was a really tough time of reflection for us, but ultimately, we came to the conclusion that all the changes we made to the game design during early pre-production left us with an assorted mix of semi-similar game features, many of which had little to do with what made the original prototype so cool—the TMD.

Armed with that knowledge, we proceeded to list the current game features and evaluated whether each one supported or detracted from TMD use, and how it was presented to the player. When put in such a severe light, it was clear which features were right for SINGULARITY and which weren't. Cutting sticky cover was just the beginning. Even though it meant we would be losing a lot of work, we knew we had to make the essential change of going back to a first-person shooter. That's when things really started going right during SINGULARITY's pre-production phase.

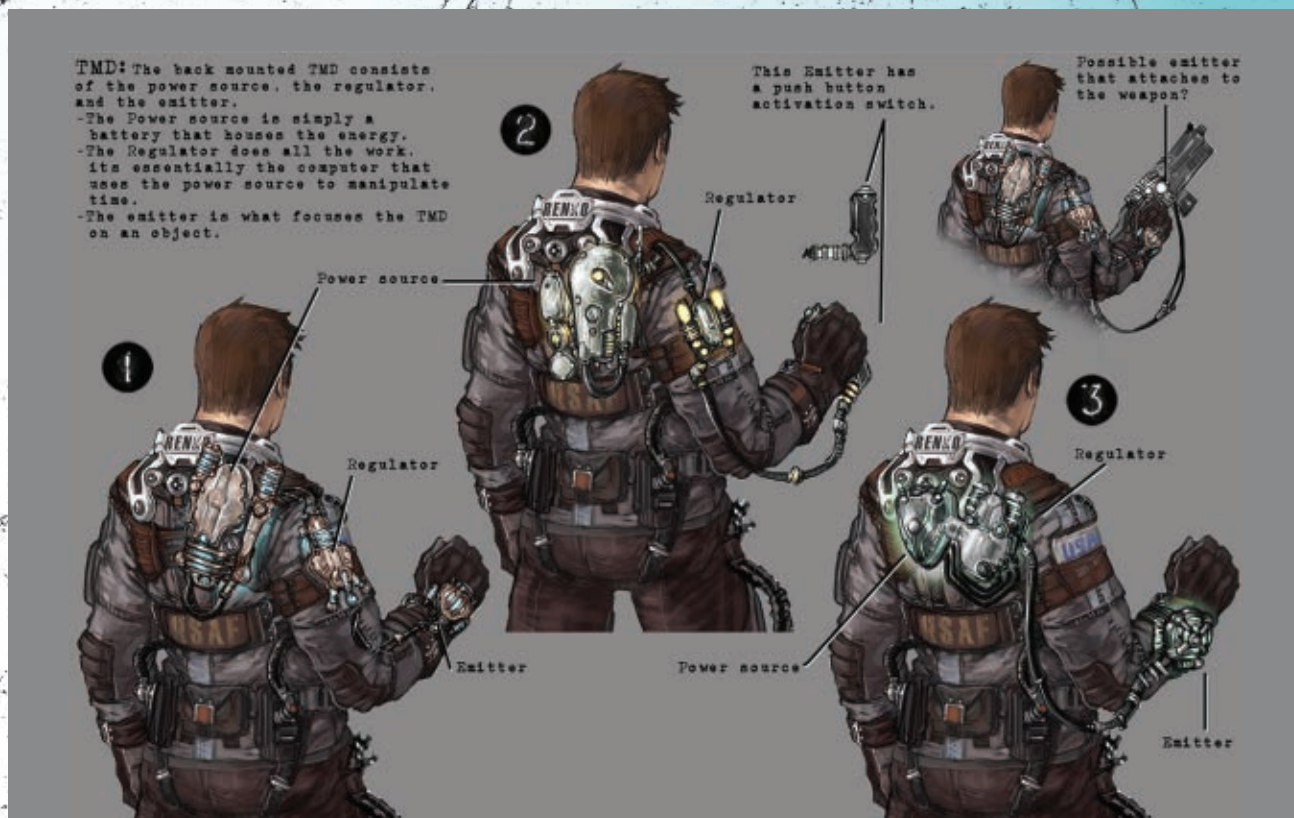
3) TRYING TO PLEASE EVERYONE LEAVES NO ONE HAPPY. Filtering out which feedback and suggestions to listen to and which to ignore is an extremely difficult challenge, we learned. With the heightened importance of SINGULARITY as a new IP, we found that feedback, suggestions, and direction came from multiple directions, both from inside the team and out.

While gathering feedback and suggestions is a crucial part of the development process, the team has to be strong in its convictions and push back on things we don't agree with. We also found it important to make sure that everyone has accountability for their direction and feedback. It's a painful lesson to learn, but once Raven learned it, we were able to move SINGULARITY toward completion.

4) TOO MUCH STUFF, TOO LITTLE TIME. SINGULARITY was an ambitious project from the start, and going into that final phase of production, we bit off quite a lot, ultimately more than we should have given the time we had. For example, at one point, we re-modeled and re-textured all the weapons. We also changed functionality for many of them, and of course tweaked out all the weapon data, such as damage, aim friction, and turn speed. We expanded TMD and weapon upgrades, layering on a light RPG system to provide more choice and customization (which was a good thing, but still added considerable work). Lots of other work went into the final phase—we added more enemy types, reworked the UI, added the all-new multiplayer with our third-person action versus FPS environment, and then of course had to polish everything that was already in the game.

This really left us short-handed at the end. We did cut features to help ourselves out, though. We had a time teleport system, for example, which allowed players to retry levels and would ultimately allow them to go back and solve puzzles with their upgraded TMD that couldn't be solved the first time through the level. But we had to drop it because there was no way we could have gotten it tested for all the possibilities players could come up with. There is internal debate to this day regarding which features we never should have started, or should have cut earlier. As SINGULARITY's development was winding down, we already started talking about building more polish time into the next project—and doing a little more pre-production early to test out the ideas we have—so we can cut the bad or overly-complex ideas earlier.

5) MULTIPLAYER MATCH A LA MODE. Multiplayer match design is often postponed until the later stages of the development cycle. The main reason is that by that point, mechanics are more solid and MP modes can be more easily established. But due to the asymmetrical nature of SINGULARITY's MP concept, we knew that the



primary mode had to be something special, and that we needed to design it alongside the mechanics.

We could still have the standard team deathmatch type mode, but we also needed something that would cater to the differences between our two MP factions and the asymmetrical nature of the combat in SINGULARITY's MP. "Extermination" is the primary mode we ended up shipping with, and it accomplished our goal fairly well. The problem is how long it took us to get there. We went through three or four different mode design iterations, and too much work was thrown out at every transition. Even after we had decided on and fleshed out this mode, we still spent a lot of time iterating exactly how the mode mechanics would work.

The root of the problem goes back to the asymmetrical nature of our combat: FPS soldiers, with their ranged attacks, have the advantage in open areas with long lines of sight. Creatures on the other hand, thrive in close quarters. To further complicate matters, the creatures are primal by nature, and look very odd performing "smart" actions such as carrying objects, interacting with mechanical objects, or anything else of that nature.


The first modes we tried were centered on control points, but felt too symmetrical and not appropriate given the factions. Also, battles often degenerated into chaos with creatures cluttered around the control points while the soldiers stood back and fired into the mob. We quickly realized that having multiple control points fractured the action, which is a bad thing when you have a maximum of 12 people playing at any one time. Teamwork was almost non-existent.

A long time and several re-designs later, including one mode where teams had to destroy each other's primary targets, we arrived at what is now known as Extermination, which much better caters to the asymmetry in combat: creatures are always on defense, and soldiers on offense. This way, creatures aren't forced to expose themselves in open areas if they don't want to. Instead, they can keep to the areas that benefit them, and strategically defend their turf. There are still control points, called beacons, but only one is available at a

time. When creatures lose that beacon to the soldiers, the action moves on to the next area and the subsequent beacon becomes the focus.

It worked out in the end, but ideally, we should have gotten there much quicker and with less wasted work. It is easy to say that we should have done better with the initial design, and should have foreseen these issues before implementing them, but the truth is that it's often very difficult to predict exactly how something will play. Sometimes it's not until you have the controller in your hand—and in this case, with 11 other people playing—that you can see whether an idea is fun, and what the issues might be. If we were to do it over again, perhaps we would focus more on identifying the strengths and possible issues associated with each of the game's mechanics early on. Furthermore, presenting the idea to a much wider audience before proceeding with implementation would help as well.

[A STITCH IN TIME]

Now that all's said and done, we are very proud of SINGULARITY, and feel we delivered on the core concept for the most part. There are things we would have loved to do differently, but that's the case with almost every game we've made. Emotionally, this was one of our most difficult titles, but also one of the most important for Raven, as everyone at the studio had a hand in creating SINGULARITY and it went through many changes. The biggest lesson we took away from this process is to stick with our gut on the vision. Questions will always be raised internally and externally, but the initial gut vision of the game and your first thought is usually correct. Our final lesson was to focus our talent on one project rather than spreading too thin by planning to release three titles within three months of each other. The order of the day for Raven Software has got to be quality over quantity. 

By Rob Gee (project lead), Brian Raffel (studio head), Steve Raffel (creative director), Gustavo Rasche (MP lead), Dan Vondrak (senior project lead), and Jon Zuk (project lead).

CELEBRATING 25 YEARS

GDC

Game Developers Conference®

February 28-March 4, 2011

Moscone Center | San Francisco, CA

Visit www.GDConf.com for more information.





CRAZY BUMP

CRAZY BUMP 1.101

SOME SAY THAT CONFESSION IS GOOD FOR THE SOUL. If that's true, then I'd better hit the game developer confessional two or three times in quick succession. You see, I've been attempting to write this review of CrazyBump, a straightforward 2D map-making tool, for some months now, but really, it seems like years. To put this in perspective, I ran through the NextEngine 3D scanner in about a month, broke apart Motionbuilder in only a couple of weeks, put 3DS Max into the "Istanbul Twist" and took that mutha down in a little more than 10 days (though I'd already been using a previous version of Max not long before). But CrazyBump is a different kettle of fish altogether. It's so ... well, simple; so simple that it got under my skin. But within that simplicity is a very powerful range of tasks that can be accomplished to generate or fine tune useful textures.

THE BASICS

» Just to cover the basics, mapping is a way to use 2D textures to add sophistication to 3D models. In all subsequent discussions, I will use a basic sphere as the 3D model unless otherwise specified.

Diffuse: The most basic map is the diffuse, or color map. It gives a 3D model the color information necessary to simulate an object in real life. In our test case, let's assume that the diffuse map is the orange and bumpy looking color information that describes a navel orange. Appetizing, isn't it?

Bump: Now, to add additional sophistication, the Bright Person Consortium, or BPC (an imaginary organization made up of all the people who came up with all the cool stuff in

the world), invented the bump map. To make a long story short, bump mapping changes the brightness of the pixels on the surface in response to a particular height map specified for the surface. For instance, our sphere, now covered in a diffuse map that looks like a navel orange, renders fine because the surface looks like you'd expect an orange to look, but animate a light in the scene and you instantly see that it's still just a smooth sphere. Add a bump map to your diffuse texture and you get complex surface depressions, all without increasing the complexity of the sphere itself. Move the light now and the shadows in the pockmarks shift with it. Rendering the finished orange with a bump map takes much less time than it would if all the deformations were created using geometry.

Normal: The BPC went into overtime when they created the normal map. Instead of a simple grayscale map used to create the bump mentioned above, a normal map is a 2D image used to replace or modify the normals of a 3D surface. A surface's normal is essentially the direction that the surface faces.

Specular: Specular maps use a simple texture to simulate a complex process, in this case: shininess. Specular maps are 2D textures used to determine how "shiny" a surface is, how discriminating the shine, and what color that shine should be at any given point.

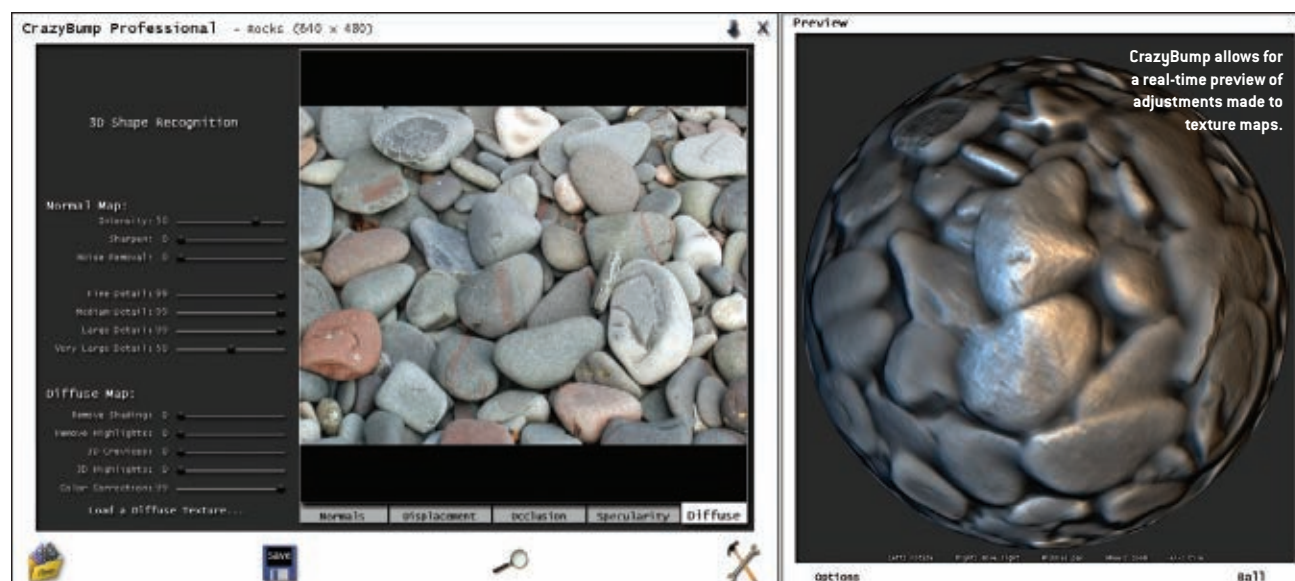
Ambient Occlusion: Another set of maps that help add realism to local reflection models by taking into account attenuation of light due

to occlusion. Ambient occlusion attempts to approximate the way light radiates in real life, especially off what are normally considered non-reflective surfaces. In short, it makes shadows more realistic, while simplifying the process of making them.

OLD DAYS/OLD WAYS

» It seems hard to imagine, but not that long ago, none of the maps described above were used at all. In the world of 2D side-scrolling video games, the idea of using 2D maps to obtain 3D results probably was bandied about around the water cooler, but it wasn't actually being done.

Then, because of advances in graphics tools and display cards, it became possible to make the maps, but they still weren't much use to





anyone except owners of top-of-the-line graphic workstations. In fact, if you could make a normal map in, say 1993, you wouldn't know what the hell to do with it except maybe hang it in an experimental art gallery somewhere in San Francisco.

But technology marches onward, and it was soon possible to make practical use of maps using consumer-level hardware. Now new types of mapping schemes are appearing every day. Parallax maps are a good example ... very sexy.

WORKFLOW TAKES A HIT

» Now that even the most modest system can employ numerous shader textures, it becomes necessary to be able to make them quickly and efficiently, and in that respect the workflow champion is CrazyBump.

CrazyBump is a simple to use but deep utility that replaces the Nvidia Photoshop filter for turning diffuse maps into normal maps, and also creates specular and ambient occlusion maps from either a diffuse map or a normal map.

Just get CrazyBump running and you'll see your first example of its simplicity, though in this case, it's not deceptive. In the lower left corner of the main screen is a line of text that says, "Click This Button To Begin" ... Aaah, simplicity.

You're then given the option of three types of files to open: a photograph (aka: diffuse map), a heightmap, or a normal map.

Let's start with a photograph/diffuse map. After the map is loaded, you're fully into the CrazyBump interface. If it's your first time in, experiment by tweaking some settings and watch how the live preview is changed by each of your actions. Notice that all the maps you can create are listed as tabs along the bottom of the preview window.


Click on each one and look in the slider section: the normal

map sliders stay the same, but below them are new sets of sliders that reflect changes you can make to displacement, occlusion, specular, and diffuse. Again, in incredibly simplistic terms, once you are satisfied with the look of your texture, hit the save button and you can save all the maps to your hard drive. From there, just apply them to an object in the game engine of your choice.

You can also start the process with a normal map and there are a lot of reasons why you should. For instance, if you create a complex model in ZBrush (or perhaps you enhance your simple model within ZBrush), you can export a normal map from ZBrush and load that normal map into CrazyBump. You can then preview the results and spit out specular and ambient occlusion maps based on it. Just so you're not seeing apples and oranges in the preview window, you can also load your model into CrazyBump so the maps lay over the surfaces they're intended for. I ask you, what could be easier than that? The only hiccup in the process occurs when the tool's graphics card compatibility can sometimes disable the preview window (while CrazyBump does support pretty much every graphics card out there, it needs up-to-date video drivers).

Within the game development field, shaders can be constructed that will use all the maps generated as separate elements, but for other work, you can use Photoshop to blend the ambient occlusion map into the diffuse map so as to cheaply and directly obtain some of the depth an ambient occlusion map will provide, but within the diffuse.

Your first duty is to adjust the Normal map parameters using seven slider bars that are prominently displayed. The sliders are: Intensity, Sharpen, Noise Removal, Shape Recognition, Fine



Crazy Bump Version 1.101
www.crazybump.com

PRICE

Professional: \$299
Personal: \$99
Students: \$49

SYSTEM REQUIREMENTS

1GHz processor (Dual cores recommended), 1GB RAM (2GB recommended), DirectX 9-compatible video card, Windows 7/Vista/XP/2000

PROS

- 1 Common sense, easy-to-understand instructions, and labels throughout.
- 2 Real-time previewing.
- 3 Friendly pricing structure.

CONS

- 1 The sliders can give you too much freedom to tweak.
- 2 Compatibility with all models of graphics cards can disable the preview window unless up-to-date drivers are present.
- 3 The online training video could use an audio track.

Detail, Medium Detail, Large Detail, and Very Large Detail. The sliders run from -100 to +100, with 0 being neutral or no change. Right next to the slider bars is an interactive display shape that shows the texture you've selected mapped naturally on a column (other shapes, such as roller, sphere, and box, are available from a pull-down menu). For instance, if you move the slider for Very Large Detail from 0 to +100, you'll see it reflected immediately in the lit shape next door. Move the +100 to -100 and the shapes that formerly pushed out in the display will push inward by an equal amount. Be careful when working with the sliders because they can give you a lot of freedom to tweak, sometimes making it difficult to know when you're really done.

You can use the same map to adjust for specular/iridescence. As we discussed before, a specular map indicates how shiny various parts of your object will be in the

final render. The darker the map, the shinier that part will be. But you don't have to worry about dark or light, because you're making adjustments to the sliders and watching the results in the preview window with the knowledge that, if you like it in the window, you'll like the results in your game engine just as much. Finally, you can make similar adjustments to create an ambient occlusion map, too.


Perhaps simplest of all, CrazyBump's interface actually has words on it that make sense. I love that. Instead of bizarre text like "Write Config," you've got a big disk icon that says "Save."

CrazyBump is a middleware tool that fills an old art pipeline void very, very well. Sure, we all know that the big modeling suites like Max and Maya can render shadow and specular maps to texture, but not with the ease of use of CrazyBump. And you don't get to preview your results in real time, like you do in CrazyBump.

And really, I ask you, why boot up Max or Maya when all you want to do is load a normal map and spit out the ambient occlusion and specular?

WRAPPING UP

» I hear that Albert Einstein, the techie, once said, "Everything should be made as simple as possible, but no simpler," and that Leonardo Da Vinci, the artist, once said, "Simplicity is the ultimate sophistication." Saying stuff like that is okay if you're a certified genius, but for the rest of the game artists pumping polygons and pixels to simply make a living, there's CrazyBump. The easy-to-understand wording, sliders that have definable ranges that make sense, and a preview window that reflects your changes within the flick of a whisker makes this the best tool that money can buy.

With apologies to Leo and Einie, CrazyBump really is as sophisticated as it needs to be. Try it yourself; I think you'll agree. 

TOM CARROLL is a video game artist currently with a prominent game studio. He is a contributor to myIPD.com, an intellectual property portal.

[BITS AND PIECES]

You can download a free 30-day trial of CrazyBump at the developer's website: <http://crazybump.com>. There are also student discounts and other offers there. The CrazyBump website also features a support forum and an online demo video that, while effective, could use even the tiniest audio track to help explain things, (even if it's in Pig Latin).



Unreal Technology News

by Mark Rein, Epic Games, Inc.

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

Epic's Unreal Engine 3 has won Game Developer magazine's Best Engine Front Line Award four times and is also one of the few Hall of Fame inductees.

Epic's internally developed titles include the 2006 Game of the Year "Gears of War" for Xbox 360 and PC; "Unreal Tournament 3" for PC, PlayStation 3 and Xbox 360; "Gears of War 2" for Xbox 360; and "Gears of War 3" for Xbox 360.

Upcoming Epic Attended Events:

PAX
Seattle, WA
September 3-5, 2010

Tokyo Game Show
Tokyo, Japan
September 16-19, 2010

GDC Online
Austin, TX
October 5-8, 2010

Please email:
mrein@epicgames.com
for appointments.

UNREAL ENGINE 3 POWERS HOMEFRONT, THQ'S AND KAOS STUDIOS' NEW SHOOTER

In THQ's *Homefront*, North Korea has occupied the United States in 2027 after a series of ripped-from-the-headlines political happenings. It's within this nightmare scenario that a band of freedom fighters takes the war to the enemy stronghold in San Francisco.

THQ-owned Kaos Studios recently elaborated on using the Unreal Engine to bring this near-future America in ruins to life for gamers on PC, Xbox 360 and PlayStation 3. Like Kaos Studios' last game, *Frontlines: Fuel of War*, the Manhattan-based studio built this new shooter using Unreal Engine 3 technology.



Kaos Studios' *Homefront*

"We chose Unreal Engine 3 for *Frontlines* because we were starting our studio and wanted to get the product out within two-and-a-half years," explained Dave Votypka, design director, Kaos Studios.

"Using Epic's tech, so we could focus on building our content, fit into THQ's strategy. Unreal allowed us to hit the ground running."

Votypka said that it was nice to have the Unreal Editor and tools on day one. His team has extended its Unreal Engine 3 technological foundation by adding features like light map streaming and different types of tone mapping.

One of the things Kaos Studios has benefited from since Epic released *Unreal Tournament 3* on PlayStation 3 is the ability to create *Homefront* from the ground up as a cross-console title.

Votypka said that Epic has provided excellent support for the Unreal Engine on PS3, allowing his team to develop the Xbox 360 and PS3 versions of the game hand-in-hand.

The programmers on his team have been keeping up

with the latest updates through the Unreal Developer Network, which allows them to converse with not only Epic, but other Unreal licensees.

During development of this massive new game, which features an epic single-player campaign and an intense multiplayer experience, the team was able to use Unreal Engine tools to bring the war-ravaged American landscape to life.

"As a player runs through this ravaged world, things happen, and we create those dramatic moments using Kismet triggers," explained Votypka.

"We were able to use Matinee, for example, when you're running through a fire scene in a strip mall parking lot and North Koreans are firing at you. When the car blows over you, nearly decapitating you, that's a Matinee piece. We also used Matinee with our mo-cap scenes and authored a lot of our in-game cinematic and storytelling moments using that toolset."

Votypka said *Homefront* will feature online combat on par with the 50-player battlefield chaos from *Frontlines*, replete with flying and ground-based vehicles.

"We're continuing in the large-scale warfare footsteps of *Frontlines*," said Votypka. "We built the foundation for large-scale warfare in multiplayer using the Unreal Engine, so for *Homefront* we hit the ground running. We've focused on polishing and iteration, as opposed to just getting helicopters and tanks working in the engine."

"The places you lived and grew up in have been twisted by this North Korean occupation that occurred after an energy crisis and a financial collapse," explained Votypka.

"We wanted to make that familiar landscape alien. We wanted to explore what would happen if the world's most powerful country was under occupation and what would it be like to fight within that world."

Players will be able to defend America in the first chapter of a transmedia experience that also includes a web-based documentary and a Syfy Channel TV movie.

Thanks to Kaos Studios for speaking with freelance reporter John Gaudiosi for this story.



For UE3 licensing inquiries email:
licensing@epicgames.com

For Epic job information visit:
www.epicgames.com/epic_jobs.html

WWW.EPICGAMES.COM

Epic, Epic Games, the Epic Games logo, Gears of War, Gears of War 2, Unreal, Unreal Development Kit, Unreal Engine, Unreal Technology, Unreal Tournament, the Powered by Unreal Technology logo, and the Circle-U logo are trademarks or registered trademarks of Epic Games, Inc. in the United States of America and elsewhere. Other brands or product names are the trademarks of their respective owners.



MONTREAL TAKES CONTROL

NOVEMBER
8/9 HILTON
BONAVENTURE
HOTEL

Network.
Learn.
Take control.

- More than **80 speakers** to learn from.
- An **expo floor** to meet, greet, discover and play.
- A **Business Lounge** to strike deals and network.
- Many **cocktails** and activities to enjoy and wind down.

Register now and save up to 30% when you purchase
your ticket before **September 25th!**

MIGS.ca



Subscribe to the newsletter!

Visit **MIGS.ca** and look for us
on **Facebook** and **Twitter!**



MONTREAL
INTERNATIONAL
GAME SUMMIT





DATA-ORIENTED DESIGN

NOW AND IN THE FUTURE

LAST YEAR, I WROTE ABOUT THE BASICS OF DATA-ORIENTED DESIGN IN THIS column [see the September 2009 issue of *Game Developer*]. In the time since that article, Data-Oriented Design has gained a lot of traction in game development and many teams are thinking in terms of data for some of the more performance-critical systems.

As a quick recap, the main goal of Data-Oriented Design is achieving high performance on modern hardware platforms. Specifically, that means making good use of memory accesses, multiple cores, and removing any unnecessary code. A side effect of Data-Oriented Design is that code becomes more modular and easier to test.

Data-Oriented Design concentrates on the input data available and the output data that needs to be generated. Code is not something to focus on, as it is with traditional Computer Science, but something that is written to transform the input data into the output data in an efficient way. In modern hardware, that often means applying the same code to large, contiguous blocks of homogeneous memory.

APPLYING DATA-ORIENTED DESIGN

» It's pretty easy to apply these ideas to a self-contained system that already works over mostly homogeneous data. Most particle systems in games are probably designed that way because one of their main goals is to be very efficient and handle a large number of particles at high frame rates. Sound processing is another system that is naturally implemented by thinking about data first and foremost.

So, what's stopping us from applying it to all the performance-sensitive systems in a game code base? Mostly just the way we think about the code. We need to be ready to really look at the data and be willing to split up the code into different phases. Let's take a high-level example and see how the code would have to be restructured when optimizing for data access.

Listing 1 shows pseudo code for what could be a typical update function for a generic game AI. To make things worse, that function might even be virtual and different types of entities might implement the code in different ways. Let's ignore that for now and concentrate on what it does. In particular, the pseudo code highlights that, as part of the entity update, it does many conditional ray casting queries, and also updates a state based on the results of those queries. In other words, we're confronted with the typical tree-traversal code structure so common in Object-Oriented Programming.

Ray casts against the world are a very common operation for game entities. That's how they "see" what's around them, and that's what allows them to react correctly to their surroundings. Unfortunately, ray casting is a heavyweight operation, and it involves potentially accessing many different areas in memory: a spatial data structure, other entity representations, polygons in a collision mesh, etc.

Additionally, the entity update function would be very hard to parallelize on multiple cores. It's unclear how much data is read or written in that function, and some of that data (like the world data structure) might be particularly hard and expensive to protect from updates coming from multiple threads.

If we reorganize things a bit, we can significantly improve performance and parallelization.





BREAK UP AND BATCH

» Without looking at any of the details inside the entity update, we can see the ray casts sticking out like a sore thumb in the middle. A ray cast operation is fairly independent of anything else related to the entity; it's heavyweight, and there could be many of them, so it's a perfect candidate to break up into a separate step.

Listing 2 shows what the broken up entity update code would look like. The update is now split in two different passes. The first pass does some of the updating that can be accomplished independently of any ray casts and decides which, if any, ray casts need to be performed sometime during this frame.

The game code in charge of updating the game processes all AI entities in batches (Listing 3). So instead of calling `InitialUpdate()`, solve ray casts, and `FinalUpdate()` for each entity, it iterates over all the AI entities calling `InitialUpdate()` and adds all the ray cast query requests to the output data. Once it has collected all the ray cast queries, it can process them all at once and store their results. Finally, it does one last pass and calls `FinalUpdate()` with the ray cast results on each entity.

By removing the ray cast calls from within the entity update function, we've shortened the call tree significantly. The functions are more self-contained, easier to understand, and probably much more efficient because of better cache utilization. You can also see how it would be a lot easier to parallelize things now by sending all ray casts to one core while another core is busy updating something unrelated (or maybe by spreading all ray casts across multiple cores, depending on your level of granularity).

Note that after calling `InitialUpdate()` on all entities, we could do some processing on other game objects that might also need ray cast queries and collect them all. That way, we can batch all the ray casts and compute them at once. For years we've been drilled by graphics hardware manufacturers about how we should batch our render calls and avoid drawing individual polygons. This works the same way: by batching all ray casts in a single call, we have the potential to achieve a much higher performance.

SPLITTING THINGS UP

» Have we really gained much by reorganizing the code this way? We're doing two full passes over the AI entities, so wouldn't that be worse from a memory point of view? Ultimately you need to measure it and compare the two. On modern hardware platforms, I would expect performance to be better because, even though we're traversing through the entities twice,

the data is processed. So what before was an `AIEntity` has now become an `EntityInfo` (containing things like position, orientation, and some high-level data) and an `AIState` (with the current goals, orders, paths to follow, enemies targeted, and so forth).

The overall update function now deals with `EntityInfo` structures in the first pass and `AIState` structures in the second pass, making it much more cache friendly and efficient.

Realistically, both the first and second passes will have to access some common data such as the entity's current state: fleeing, engaged, exploring, idle, etc. If it's only a small amount of data, the best solution might be to simply duplicate that data on both structures, which goes against all "common wisdom" in Computer Science. If the common data is larger or is read-write, it might make more sense to give it a separate data structure of its own.

At this point, a different kind of complexity is introduced—keeping track of all the relationships from the different structures. This can be a particular challenge while debugging because some of the data that belongs to the same logical entity isn't stored in the same structure, making it harder to explore in a debugger. Even so, making good use of indices and handles helps this problem become much more manageable (see "Managing Data Relationships" in the September 2008 issue of *Game Developer*).

CONDITIONAL EXECUTION

» So far things are pretty simple because we're assuming that every AI entity needs both updates and some ray casts. That's not very realistic because entities are probably very bursty—sometimes they need a lot of ray casts, and sometimes they're idle or following orders and don't need any for a while. We can deal with this situation by adding a conditional execution to the second update function.

The easiest way to conditionally execute the update would be to add an extra output parameter to the `FirstUpdate()` function that indicates whether the entity needs a second update or not. The same information could then be derived from the calling code depending on whether there were any ray cast queries added. Then, in the second pass, we only update those entities that appear in the list of entities requiring a second update.

The biggest drawback of this approach is that the second update went from traversing memory linearly to skipping over entities, potentially affecting cache performance. So what we thought was going to be a performance optimization ended up making things slower. Unless we're

Data-Oriented Design means making good use of memory accesses, multiple cores, and removing any unnecessary code. A side effect of Data-Oriented Design is that code becomes more modular and easier to test.



we're using the code cache more efficiently and accessing the entities sequentially (which allows us to pre-fetch the next one too).

If this is the only change we make to the entity update, and the rest of the code is the usual deep tree-traversal code, we might not have gained much because we're still blowing the cache limits with every update. We might need to apply the same design principles to the rest of the update function to start seeing performance improvements. But at the very least, even with this small change, we have made it easier to parallelize.

One thing we've gained is the ability to modify our data to fit the way we're using it, and that's the key to big performance boosts. For example, after seeing how the entity is updated in two separate passes, you might notice that only some of the data that was stored in the entity object is touched on the first update, while the second pass accesses more specific data.

At that point we can split up the entity class into two different sets of data. Then the most difficult task is to name these sets of data in some meaningful way. They're not representing real objects or real-world concepts anymore, but different aspects of a concept, broken down purely by how

gaining a significant performance improvement, it's often better to simply do the work for all entities whether they need it or not. However, if on average less than 10 or 20 percent of the entities need a ray cast, then it might be worth avoiding doing the second update on all the other entities and paying the conditional execution penalty.

If the number of entities to be updated in the second pass were fairly small, another approach would be to copy all necessary data from the first pass into a new temporary buffer. The second pass could then process that data sequentially without any performance penalties, which would completely offset the performance hit that comes from copying the data.

Another alternative, especially if the conditional execution remains fairly similar from frame to frame, is to relocate entities that need ray casting together. That way, the copying is minimal (swapping an entity to a new location in the array whenever it needs a ray cast), and we still get the benefit of the sequential second update. For this to work, all your entities need to be fully relocatable, which means working with handles or some other indirection, or updating all the references to the entities that swapped places.

DIFFERENT MODES

» What if the entity can be in several totally different modes of execution? Even if it's the same type of entity, traversing through the modes linearly by calling the update function could end up using completely different code for each of them, resulting in poor code cache performance.

There are several approaches we can take in a situation like this:

- If the different execution modes are also tied to different parts of the entity data, we can treat them as if they were completely different entities and break each of their data components apart. That way, we can iterate through each type separately and get all the performance benefits.
- If the data is mostly the same, and it's just the code that changes, we can keep all the entities in the same memory block but rearrange them so entities in the same mode are next to each other. Again, if you can relocate your data, this is very easy and efficient [it only requires swapping a few entities whenever the state changes].
- Leave it alone! Ultimately, Data-Oriented Design is about thinking about the data and how it affects your program. It doesn't mean you always have to optimize every aspect of it, especially if the gains aren't significant enough to warrant the added complexity.

THE FUTURE

» We might wonder if thinking about a program in terms of data and doing these kinds of optimizations is a good use of our time. Is this all going to go away in the near future as hardware improves? As far as we can tell right now, the answer is a definite no. Efficient memory access with a single CPU is a very complicated problem, and matters get much worse as we add more cores. Also, the amount of transistors in CPUs (a rough measure of power) continues to increase much faster than memory access time. That tells us that, barring new technological breakthroughs, we're going to be dealing with this problem for a long time. We'll have to face it right now and build our technology around it.

There are some things I'd like to see in the future to make Data-Oriented Design easier. We can all dream of a new language that will magically allow for great memory access and easy parallelization, but replacing C/C++ and all existing libraries is always going to be a really hard sell. Historically, the best advances in game technology have been incremental, not throwing away existing languages, tools, and libraries (that's why we're still stuck with C++ today).

Here are two things that could be done right now and would work with our existing codebases. I know a lot of developers are working on similar systems in their projects, but it would be great to have a common implementation released publicly so we can all build on top of them.

Language. Even though a functional language might be ideal, either created from scratch or reusing an existing one, we could temporarily extend C to fit our needs. I would like to see a set of C extensions where functions have clearly defined inputs and outputs, and code inside a function is not allowed to access any global state or call any code outside that function (other than local helper functions defined in the same scope—see Listing 4). This could be done as a preprocessor or a modified C compiler, in order to remain compatible with existing libraries and code.

Dependencies between functions would be expressed by tying the outputs of some functions to the input of other functions. This could be done in code or through the use of GUI tools that help developers manage data relationships visually. That way, we can construct a dependency diagram of all the functions involved in every frame.

Scheduler. Once we have the dependencies for every function, we can create a directed acyclic graph (DAG) from it, which would give us a global view of how data is processed in every frame. At that point, instead of running functions manually, we can leave that job in the hands of a scheduler.

The scheduler has full information about all the functions as well as the number of available cores (and information from the previous frame execution if we want to use that as well). It can determine the critical path

LISTING 1

```
void AIEntity::Update(float dt)
{
    DoSomeProcessing();
    if (someCondition && Raycast(world))
        DoSomething();
    if (someOtherCondition && BunchOfRayCasts(world))
        DoSomethingElse();
    UpdateSomeOtherStuff();
}
```

LISTING 2

```
void AIEntity::InitialUpdate(float dt, RayCastQueries& queries)
{
    DoSomeProcessing();
    if (someCondition)
        AddRayCastQuery(queries);
    if (someOtherCondition)
        AddBunchOfRayCasts(queries);
}

void AIEntity::FinalUpdate(const RayCastResults& results)
{
    UpdateSomeOtherStuff(results);
}
```

LISTING 3

```
RayCastQueries queries;
for (int i=0; i<entityCount; ++i)
    entities[i].InitialUpdate(dt, queries);

// Other update that might need ray casts

RayCastResults results;
for (int i=0; i<queries.count; ++i)
    PerformRayCast(queries[i], results);

for (int i=0; i<entityCount; ++i)
    entities[i].FinalUpdate(results);
```

LISTING 4

```
void FirstEntityUpdate(input Entities* entities, input
int entityCount, output RayCastQueries* queries,
output int queryCount);
```

through the DAG and optimize the scheduling of the tasks so the critical path is always being worked on. If temporary memory buffers are a limitation for our platform, the scheduler can take that into account and trade some performance time for a reduced memory footprint.

Just like the language, the scheduler would be a very generic component, and could be made public. Developers could use it as a starting point, build on top of it, and add their own rules for their specific games and platforms.

DATA NOW

» Even if we're not yet ready to create those reusable components, every developer involved in creating high-performance games should be thinking about data in their games right now. Data is only going to become more important in the future as the next generation of consoles and computers rolls in. ☞

NOEL LLOPIS has been making games for just about every major platform in the last twelve years. He's now a one-man band making iPhone and iPad games.



BIG SCREEN BLUES

WITH SOME KEY LEARNING, GAME ANIMATORS NEEDN'T ENVY THEIR FILM COUNTERPARTS

JOB INTERVIEWS CAN BE STRESSFUL, SO YOU

have to cut a little slack for the goofy things candidates say. Here's a free tip for future hopefuls, though: if somebody asks you, "So, why do you want to work in games?" don't say, "Well, it seemed like an easy way to work on my demo reel so I could get a job in Hollywood."

The game industry's relationship with Hollywood is dysfunctional. The conventions of cinema run deep in our DNA, and the last hundred years of film provide a huge wealth of experience to all of us who tell stories with moving images. That said, games and movies are fundamentally different media—a lesson easily reinforced by a glance at the checkered history of film-to-game "convergence" in recent history. When we treat our own work as a subset of Hollywood, we're selling ourselves short.

Nobody suffers from the game industry's inferiority complex more than animators. Compared to most game specialties, animation has a great sense of its own history as an art form. It's strongly rooted in the traditions of Walt Disney, Chuck Jones, and the Nine Old Men, reinforced by the brilliance of studios like Pixar and the powerful influence of animation schools like CalArts and Animation Mentor. Animators who study their craft and its history can't help but measure themselves against the intimidating giants of film animation.

The artistic legacy and accumulated experience of generations of animators is inspiring and filled with useful insights, but can also be a bit oppressive. It's hard not to feel like a second class citizen when comparing your work—constrained as it is by runtime performance, game design, and the tastes of hyperactive teens—to carefully crafted performances designed for only one viewpoint.

LIMB FROM LIMB

» In that very obvious way, it's literally impossible for game animators to compete with their counterparts in film. Interactivity is what makes games so powerful, but interactivity is also the antithesis of authorial control. Most games turn over control of the camera to the player. That's perfect for gameplay but terrible for traditional visual storytelling. Without the camera to frame and direct the audience's attention, game animators are often forced to catch the player's eye with overdone gestures and highly conventional stock poses.

The fixed camera also makes it easier for film animators to cheat. The laws of physics are a lot easier to evade with forced perspectives and carefully chosen angles; heck, Hollywood animators will happily rip an arm or a leg right off a character in pursuit of a golden pose or strong silhouette, safe in the knowledge that the camera won't expose their tricks (and that some poor sucker in post has to paint out the giveaway shadows).



We don't get the luxury of at-will dismemberment (outside of zombie games, anyway). Learning to live without a fixed camera is unpleasant for classically trained animators, but it's the necessary first step out of the shadow of the film tradition. Teams and leads can certainly help the process along with some fairly simple arrangements: The quickest way to escape the tyranny of the too-well-framed shot is to make sure reviews are done in the round. Don't do critiques and approvals on playblasts or AVIs, do them in the game engine or a dedicated animation

preview tool. If possible, view your work with real game characters and lighting—this makes it much easier to catch hitches, weak poses, or clipping that could be hidden with a fixed viewpoint.

If you don't have the ability to see animations in their real context, start complaining. It's vital. Until your team builds the tool you need, though, you can still do reviews in the round using QuickTime's FBX support (www.apple.com/quicktime/resources/components.html) to view 3D animations in real time in the QuickTime viewer. Alternatively, teams that use Granny or Morpheme can use those tools to do reviews. However you do it, viewing everything in the round all the time makes it easier to spot hitches and awkward poses that could be hidden by a fixed camera. Spending more time in an interactive environment (and less time in the safe confines of Max or Maya) helps you take ownership of the real end product: the game character.

Learning to focus on the end result, rather than the process, is critical. Even in Hollywood, animators often feel alienated from the final results of their work. After a shot leaves the animator's hands, it gets passed along through many layers of specialist TDs to add lights, effects, secondary animation, cloth sims, and the other glorious visual paraphernalia we can only dream of. This certainly makes for pretty pictures, but it also diminishes the direct authorial control of the individual animator; it's hard to reconcile the daily business of submitting shots to the busy machinery of a film pipeline with the lofty acting and storytelling ideals of Ed Hooks and Richard Williams. For game animators, though, the path from Max or Maya to living character in the game world has so many twists and turns, it can feel like an endless maze.

It sometimes feels as if everything in the runtime environment—whether it's stuttery transitions, brutal keyframe compression, or designers retiming animations for gameplay—is conspiring to undermine your lovingly-placed keys and carefully-chosen poses. "It looked great in Maya!" is the animator's universal, pathetic lament. Unfortunately, that sense of alienation leads far too many animators to disown their work once it gets out of their hands and focus only on demo reel friendly renders.

Unfortunately, audiences and reviewers will never know how good it looked in Maya. Very few of us get to appear in the "making of" trailers where those carefully-staged playblasts occasionally see the light of day. The only reality in game animation is the game itself—with all the compromises and shortcuts imposed on us by runtime budgets, wonky code, and impossible design directives. Love it or hate it, runtime animation is what most of us do. The question we face as artists is whether we are content to merely suffer through it—taking refuge in our demo reel full of pretty AVIs and the sympathy of our fellow sufferers on the animation team—or whether we can embrace the reality of the medium.

RUNTIME FUNTIME

» Embracing the medium doesn't mean just sucking it up and accepting lousy results. Interactive animation is hard, but it has unique potential to speak to users. Games like *ASSASSIN'S CREED* and *UNCHARTED* show how much can be achieved in the field despite the demands of runtime performance and gameplay. A character whose animations are crafted to work with the gameplay of movement or combat is a vital conduit between the player and the game world. It's precisely the union of interactivity and top-notch animation that literally transforms us into Mario, Kratos, or Shaquille O'Neal when we pick up our controllers. The meshing of AI and animation can create characters that don't simply hit their marks like puppets, but take on a virtual life of their own—an illusion of life that Ollie Johnson could only envy. Game animation is not just film animation with a lower quality bar, it's a vital art form that is constantly breaking new ground and speaking to players in new ways.

The possibilities of interaction are bracing. So, unfortunately, are the challenges. As sandbox games continue to attract customers, the spreadsheet of animation tasks explode in all directions. So many characters times so many items times so many vehicles times so many gameplay states ... the combinatorial math is brutal and it only gets worse over time. Add in the need to provide transitions between all of those options ("put away the sword while dismounting the hippogriff, looking left, wounded") and the future blurs into a Dickensian treadmill of fill-in-the-blanks drudgework (or, just as bad, outsourcing more and more to hordes of cheaper overseas contractors). The relationship between designers who want to add more features and animators who know they'll be doomed to months of cut-and-paste work often spirals into mutual suspicion and acrimony.

The future doesn't have to look that grim, though. The increasing demands of modern games are forcing animators to deal with new tools and techniques, often in ways that don't seem very close to the classic heritage of the discipline. But determination and willingness to experiment can help animators escape from drudgery and concentrate on the fundamentals.

On the lowest level, real-time animators need to understand technologies like IK, ragdolls, and animation blending. The runtime versions of these tools are often simpler and less controllable than the tools we're used to in Maya, but they'll only get better when there's well-informed input from animators. It's easy to grumble, and advocating for the right tools can be extremely frustrating, but this is a crucial skill for the ambitious animator who really wants to push the boundaries of interactive animation. Peek under the hood of any game with cutting edge animation and you won't just see great tech, but also a lot of feature requests from smart animators who

knew enough to negotiate effectively with design and engineering.

On the higher level, systems for managing transitions and animation states also have tremendous impact on the lives of virtual characters. No amount of keyframing mojo can sustain the illusion of life when every change of action or direction interposes a glaring robotic hitch. Thankfully, artist-driven tools like Morpheme and Unreal's AnimTreeEditor put this vital visual task into the hands of artists, where it belongs—at least, in theory. In practice, all too often this critical job gets dumped in the lap of a junior animator or technician because it involves a lot of sliders. In a modern game, good transitions and blends are just as necessary as traditional keying. Animators need to be involved in transitions, blended variations, and physics interactions, even if that means buckling down and learning some ugly tools and intimidating technologies. This may be alien territory today, but there was a time, not that long ago, when "inverse kinematics" sounded like mumbo jumbo to animators too.

TO THE FUTURE, AND BEYOND!

» Does all this invalidate the core of animation? Do modern games really demand that every animator become a coder? Of course not. Timing, posing, and acting remain the indispensable building blocks of interactive animation, just as they were in the days of pegboards and celluloid. The big difference is that we need something a lot more complicated than a pen and a pair of white gloves to realize them. If we want that to happen in an artist friendly way, it's up to us to set the agenda and push for the features we need. The important thing is that the technology is informed by artists who know both where the medium has come from and where they want it to go.

Unlike film animation, our medium is still in its infancy. While we owe a great debt to the giants of the past, we also owe it to ourselves to push the boundaries of what our medium can accomplish artistically. Just as importantly, we must continue to innovate or we'll be swamped with ever-increasing content demands. In the last few decades, we've done a good job of moving from the world of pen-and-ink to a world of polygons and pixels without losing touch with our roots. As we move into a world where IK, blends, and physics are also primary tools, we'll do the same thing. If we can do it right, evolving tools and artistic conventions that empower us and make our characters more truly interactive, we'll find those other guys envying us, rather than the other way around. ☺

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 a consultant helping game studios perfect their art tools and pipelines.



THE CHICK PARABOLA

ON MARCH 11, 2009, DURING THE *Three Moves Ahead Strategy Gaming Podcast*, freelance journalist Tom Chick introduced a phenomenon which has come to be known as the Chick Parabola:

"My experience with *EMPIRE: TOTAL WAR* is this parabola of fondness. At first I don't like it, so I'm at the bottom of the curve. I don't like it because they do a terrible job with their documentation—it's got a terrible manual; they want you to play through this scripted campaign if you want to learn anything; the tool-tips are really screwy. So, I'm hating it.

But I'm playing it, and I'm learning it, and I'm liking it, so I'm climbing up that parabola. At the very top of the curve, I think, 'Hey, I sort of figured it out. I like this game.' But then I start to discover that the AI is terrible, that it's a dumb game, and I start coming down the far end of the parabola, and I am no longer fond of *EMPIRE: TOTAL WAR*.

Commonly, there's this curve where I enjoy a game, and then I master the system, and then—unless it's got a good AI—I lose all interest because I realize that mastering the system is where the challenge ends. Once I reach that point, the game is dead for me, and

I hate that! That's when the game should really start to take off."

Many veteran gamers will recognize this feeling from their own experiences—the rising enjoyment that comes from learning an interesting game system followed by an inevitable deflation as the challenge slowly disappears.

Sometimes a simple technique or exploit can render the rest of the game balance irrelevant. Usually the culprit is a weak adversary, as the artificial intelligence cannot grasp certain core game mechanics to offer the player a robust challenge. The problem is that the game's designers have made promises on which the AI programmers cannot deliver; the former have envisioned game systems that are simply beyond the capabilities of modern game AI.

SYMMETRY MATTERS

» Still, not all games suffer from the Chick Parabola. Many are so fundamentally asymmetrical—*SUPER MARIO BROS.*, *GRAND THEFT AUTO*, *WORLD OF WARCRAFT*, *HALF-LIFE*—that the AI is simply a speed bump that can be easily tuned to provide the right level of challenge. The games which suffer the most are ones where the computer is forced to play the same game as the human.

These symmetrical games—*STARCRAFT*, *STREET FIGHTER*, *PUZZLE QUEST*, *HALO*—have a unique challenge in that each game mechanic must not simply be judged on its own merits but also by asking whether the AI can reasonably understand the option and execute it successfully. Unfortunately, asking this question often disqualifies many interesting ideas.

Artificial intelligence is notoriously poor at handling issues of trust and betrayal, of long-term investments, of multi-front wars, and of avoiding traps obvious to any human. The question of trust, in particular, has torpedoed multiple attempts to make a viable single-player version of the classic board game *Diplomacy*, which relies so acutely on being able to read one's enemies, one's friends, and one's supposed friends.

Thus, to avoid the Chick Parabola, designers of symmetrical games must weigh carefully the implications of various game mechanics. An interesting play option which overtaxes the AI runs the risk of making the game more interesting in the short-term, but less interesting in the long-term once the player masters the system and can use the mechanic to run rings around the artificial intelligence.

Of course, designers of symmetrical games built primarily for multiplayer—such as the *BATTLEFIELD* series or the fighting genre—can choose to sacrifice single-player longevity for multiplayer depth. Non-conventional weapons are fine if we assume that veterans of the game are only interested in playing the game with each other.

The human brain is remarkably flexible, with the ability to easily process novel mechanics which are orthogonal to the rest of the game. This approach has many advantages; Valve has been able to radically change the multiplayer-only *TEAM FORTRESS 2* with each character update (giving the Demoman a sword and shield, for example) without having to worry about toppling over an increasingly rickety AI.

DESIGNING FOR THE AI

» Symmetrical single-player games need to be designed as much for the artificial intelligence as for the humans themselves. Even if it's painful, designers must be willing to leave some of their most orthogonal—and often most creative—ideas off the table for the sake of the AI. Game design is a series of trade-offs, and



Tom Chick's experience playing **EMPIRE: TOTAL WAR** inspired the Chick Parabola.

empowering the AI is important for avoiding the downward slope of the Parabola.

Creative developers can solve this problem at the design stage before it even reaches some doomed AI programmer. One game mechanic that pushed Chick over the edge with **EMPIRE: TOTAL WAR** was amphibious invasion. The AI was simply incapable of coordinating its land and naval forces together to launch a coherent and effective invasion of an overseas target. Smart players would quickly learn that if the AI could not attack amphibiously, the strategic balance could be gamed easily. Maybe England's troops are not such a threat after all?

This problem is not unusual; strategy games with transportation units almost always suffer from ineffective artificial intelligence. Coordinating land and naval units to be ready in the same place and at the same time—along with the necessary escort ships—is a non-trivial task.

RISE OF NATIONS, Big Huge Games's historical RTS, presented a blunt but effective solution to this problem; land forces that approach the shore simply turn into boats to carry themselves across the water. Once they reach

their destination, the boats transform back into the original land units. Thus, there were no transportation ships to build or manage.

With one simple stroke, Brian Reynolds, the game's designer, removed a classic AI problem, ensuring that water maps would remain interesting for veteran players. The design may have sacrificed the "realism" of requiring the player to build transport ships along with other naval units, but the upside was a significant extension of the game's longevity.

Furthermore, many design changes meant to bolster AI by simplification often have the side effect of making the game itself more enjoyable for the player. Quite a few players did not miss having to build and herd transports in **RISE OF NATIONS**. **CIVILIZATION 3** and **CIVILIZATION 4** introduced global unit support and city production overflows, respectively; both changes helped the AI manage its resources but also made the game more enjoyable for the average player by drastically reducing micromanagement.

TOUGH DECISIONS

» The designer's biggest challenge comes when a mechanic which is demonstrably fun or core to

the game's theme needs to be simplified or dropped. Occasionally, a game can get away with assuming that a certain option will be human-only; in the original **CIVILIZATION**, Sid Meier added nukes to the end-game but didn't allow the AI to use them. He reasoned that because the super-weapon came only at the end of a game with such scope, players who used them were not abusing the game; they were simply having a bit of crazy fun at the end.

Further, if the designer wants to maintain a mechanic that the AI can't use, cheating is not a viable solution for balancing away the AI's disadvantage. Allowing too many human-only systems effectively turns a symmetrical game into an asymmetrical one, which will eventually affect the strategic balance.

In **EMPIRE: TOTAL WAR**, once players know that the AI will never launch an effective amphibious invasion, the rest of the game changes immediately. Maybe players don't need to bother defending their coastal territories? Maybe land-based allies are more important than water-based ones? Maybe the AI can be tricked into wasting its resources on futile invasions? Most importantly, the

player is no longer playing like a queen—she is playing like a gamer who knows that the AI doesn't work, one who is on the downhill side of the Parabola.

Ultimately, the designer may have to make a tough choice: drop a beloved mechanic, or risk shortening the replayability. Many options do exist to extend a game's longevity outside of pure balance—scripting a variety of scenarios, supporting procedural content generation, providing robust mod support, developing post-release content, and so on.

However, for robust replayability, nothing compares to pure strategic depth with a competent computer opponent. Sacrificing the game's longevity to provide a few moments of fun for the human is essentially eroding the design at the foundation. As Chick puts it, when the player finally learns a system, "That's when the game should really start to take off." The joy of learning is a big reason why games are fun, but no one wants to study for a test that doesn't exist. 🎮

SOREN JOHNSON is a designer/programmer at EA, working on browser-based strategy games at www.strategystation.com. He was the lead designer of **CIVILIZATION IV** and the co-designer of **CIVILIZATION III**. Read more of his thoughts on

www.GDCOnline.com

GDC Online

Game Developers Conference® Online
October 5-8, 2010 | Austin, TX

Visit www.GDCOnline.com for more information



SFX FREQUENCY MAPPING

WHY YOUR AUDIO MAP SHOULD BEGIN WITH A HARD LOOK AT FREQUENCY

THE DESIGN DOCUMENT. THE DEVELOPMENT wiki. The game bible. Whatever you call it, in your video game production, there's probably some sort of repository of design decisions made well before anyone on the team creates a production quality asset. In this document, game designers might put in things like: whether certain levels are hub-and-spoke models as opposed to funnels. Artists might note that the main character is primarily green and rectangular as opposed to the antagonist's purple trapezoid of a body.

And sound designers ... well ... they usually just have lists of things that need to get done. Gunshots here, switches there, screams here, and so forth. Seems like there's a dearth of information there compared to the other disciplines, right?

There are plenty of reasons why this is the case. Often the sound department is significantly smaller than art and design, so there's less need to have essential design concepts communicated in a document. Also, audio so often follows the visual execution, that it makes it hard to decide early on the exact aural ideas to execute. And then there's the old standby: relative to other components, all that sound pre-production work becomes useless once you're in the thick of actually making sounds. Still, this doesn't mean that we can't try!

HIT THE DOC

» Other than the sound effects that need to be filled, what else do you put in that early audio design document? Just as the art director might decide beforehand on basic colors and shapes that comprise the game's visual aesthetic, the basics of frequency (low or high) and shape (simple or complex) should also make their presence known early on. Where the art director might suggest that the hero of our cartoonish fantasy DS adventure game looks like a rotund green elf, the sound designer would decide that his main action of casting attack magic should have a fundamental frequency around 500 Hz and be pretty harmonically thick, like a square wave tapered temporally, with a bit of a presence peak around 3 kHz but absolutely no energy past 5 kHz.

You might consider that to be a lot of arbitrary sound design decisions based on the fact that he's a fat green magician. But making decisions like the above allow us to start putting together the puzzle that is the game's audio design. And

besides, it's not that arbitrary; attacking is the main way for the player to communicate with the game, so having the attack magic carry properties of the human voice isn't a bad idea.

So we've got a place for the attack magic; where do we put the other stuff? Let's start placing the puzzle pieces next to each other and see what works out. Maybe put the UI sounds a bit above the voice. Let's give enemy sound effects a place just slightly below our main character in the frequency spectrum, but make them less harmonically rich, more of a sine wave than a square wave, so that they don't feel as powerful as the player. The cartoonish, sparkly explosions? How about placing them right at the upper limit of musical sounding, about 4–5 kHz with a cool little delay effect.

How do we know that these sound effects are in the right place? It's mostly guess work, but it's well-informed guess work. If we want a particular sound to stand out, giving it its own dedicated space on the frequency spectrum—away from other sound effects—will help. If we're dealing with a group of creatures that are visually different but effectively the same from a gameplay perspective, having them occupy approximately the same frequency range might help to convey that.

Are there existing sounds that you know have to be in the game? Things that the designer/publisher/fan expects? Go ahead and analyze it, then throw it into the frequency spectrum. Then you can build around that with everything else the game needs. Is there a particular voice actor that the producer is keen on bringing in to the project? Figure out where he sits on the frequency spectrum and how much space he takes, then work around that. Got a composer that has a predilection for those high french horns? Make a note to be careful what you put into the 500–1000 Hz range.

OPINION NOTED

» So, you've made some decisions about sound design; how are you going to notate them? Maybe it's just a matter of making a slightly more elaborate table in the design wiki. I personally like to put things on a traditional piano grand music staff. Besides offering me a familiar space, it has lines and spaces where I can just write where sound effects should go, providing a way to cleanly translate aural ideas

into a visual aid. If you're not adept at reading music, know that you can draw musical notes pretty much anywhere on the staff. The bottom staff holds stuff from 100 to 500 Hz, while the top staff holds it from 500 to 2000 Hz (see Figure 1).

You can go lower than 100 Hz; just place whatever you want even lower on the staff. You can go higher than 2000 Hz; just place it even higher, in the white space way above the staff. The important thing is using this tool to make sure no one is stepping on anyone else's toes ... unless that's exactly what you're going for.

Having all of this preparation done before getting into the messy business of actually crafting sounds should help make the job a

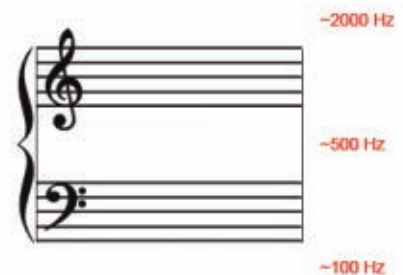


FIGURE 1 Using a traditional music staff is a good way to note the frequency spacing of sound effects.

bit less messy. Knowing some specifics of the sounds we need to make from scratch keeps us from being paralyzed by creative choices. Keeping those early design decisions in mind will help us trawl more quickly through library sounds for the needed sound effect. As we know all-too-well: you can never be too efficient in game production.

The next time you see some pre-production game documentation, you might want to try your hand at designing a bit more extensively well before you touch Pro Tools or Sound Forge. A little bit of info on those basic components of sound goes a long way. 🎧

VINCENT DIAMANTE is a music composer and educator in the Los Angeles area. He penned the score for ThatGameCompany's *FLOWER* and currently teaches at USC. Email him at diamante@gmail.com.



ARCHER AIMS AT SOE

EX TREYARCH EXECUTIVE PRESIDENT LEADS SEATTLE DIVISION

Chris Archer, former game director at Treyarch and executive producer at Activision on the publishing side, has recently made the move to Sony Online Entertainment; he was picked to run the company's Seattle division, which is currently working on the shooter MMO THE AGENCY. We spoke with Archer about his transition from the console to online space, and what it takes to lead a studio.

Game Developer: You were an executive producer and game director at Activision for quite a long time. Which of these best prepare one to run a studio, or is it the combination?

Chris Archer: As a game director at Treyarch, I was responsible for the creative vision of the project from conception to completion and ultimately responsible for one thing: making a great video game.

When asked from people outside the industry, I would compare my role of executive producer as a combination of the director and producer on a feature film. As an executive producer for Treyarch and Activision, I was tasked with leading a 150+ person team, and was responsible for making sure the game was on budget, on time, and of the highest quality.

I think both roles are really critical experiences for running a successful studio. You need to understand what makes a game fun, how the games are built, how to lead a large scale team and keep all of those developers happy, on time and on budget.

GD: What are the new challenges you face as your role changes to studio head at SOE?

CA: The biggest challenge for any person coming in to a new team is gaining the trust and respect of their team.

My previous positions both on the developer and publisher side have provided me with a lot of insight and experience in not only the day to day operations of managing large teams, the business side of making great games on time and on budget, but also what publishers' expectations are and how to communicate on both sides of the fence. The biggest challenge in terms of the role change has been the transition from traditional console and PC development to MMO development. I have much to learn about these games, but I am really excited about what is ahead in the MMO space and I truly believe that online play is where all development is headed.

GD: How has the shift to a much more online-focused organization worked? Did you have to study up for this at all?

CA: The cool thing about the video game business is there is always something new and exciting to sink your teeth into. This was no exception when I moved over to SOE. Not only have I learned a lot about the MMO business and how it works, but also how differently these games are built.

Is it really studying when you are having fun doing it? My studying consisted of playing a lot of different MMO titles, as well as nearly all of the FPS products both current and past.

GD: Console games have made a consistent drive toward better visuals, while online games generally put that second to "just one more"-style of addictive play. How do you see that progressing in the future?

CA: The parity between MMO games and top tier console games is narrowing quickly. I can tell you I see it on my screen every day: the next wave of MMO titles that you will see come out will not only look as good as those top console titles, but provide hundreds more hours of play comparatively, with little to no difference in fidelity of gameplay. THE AGENCY, in particular, I believe will be one of those titles. Our goal, and we are proving it right now, is to provide graphics and game fidelity that rivals or beats the top shooters on consoles, while at the same time still providing hundreds and hundreds of hours of compelling MMO type "just one more" gameplay. Customization, socialization, and twitch gameplay all in one place, in one game. I can't wait.



new studios

Games industry veterans Brenda Bailey Gershkovitch and Kirsten Forbes have founded Silicon Sisters Interactive, a Vancouver-based studio focused on creating games for a female audience. Silicon Sisters is the first female-owned development studio in Canada, and has an all-female design staff.

Pitbull Studio has opened its doors a year after the closure of Midway Studios Newcastle, with the intention of carrying on the shuttered studio's legacy of developing racing games.

Scott Martins, former head of Sony Online Entertainment Denver, and other SOE veterans have opened Dire Wolf Digital Game Studio, a Denver-based studio specializing in online trading card games, digital collectibles, and social titles.

Ex-Game Developer columnist Alexander Brandon, who worked on series such as UNREAL and DEUS EX, has announced the formation of Funky Rustic, a Texas-based multimedia audio production studio.

who went where

Michael Rawlinson, director general of UK trade body the Entertainment and Leisure Software Publishers Association, has been appointed vice chairman of the Alliance Against IP Theft, a coalition of professional associations and enforcement organizations that aim to protect intellectual property rights in the UK.

Activision Blizzard has announced that advertising executive Eric Hirshberg will take the reins as the CEO of the company's publishing division, following the resignation of Mike Griffith.

HAPPY AQUARIUM developer CrowdStar appointed four game industry veterans to its management team: Pete Hawley as product development VP, Mark Hull as product marketing and community VP, Mike Ouye as monetization and merchandising VP, and Robert Einspruch as business development director.

Cryptic Studios announced that Daniel Stahl has been promoted to executive producer of STAR TREK ONLINE, after working at the company as a producer on the game, as well as on CHAMPIONS ONLINE.

Programmers & 3D Artists: Are you the trailblazer for our new frontier?

Friendly and small-feeling
\$1B company (on NYSE)
HIRING experienced
team members full-time
for our Naples, Florida studio

Apply online: dreamtools.com

... oh and the answer to your first question for us... Ninjas.

DreamTools

If it doesn't look real, we didn't make it.



GET YOUR Game on!

EVERYTHING YOU NEED TO KNOW TO
GET INTO THE Game INDUSTRY!

- news and features for students and educators
- GETTING STARTED section – an invaluable how-to guide
- message boards



DIGITALCOUNSELOR

I'LL MATCH YOUR INTERESTS AND
GOALS WITH THE RIGHT Game
RELATED programs and schools
FROM AROUND THE WORLD.

www.gamecareerguide.com



Download your FREE digital
edition of the 2010 game developer
Game Career Guide online at:
www.gamecareerguide.com



GDC ONLINE ANNOUNCES NEW PANELS, LECTURES

GDC ONLINE ADDS ZYNGA, DISNEY, PLAYDOM, TENCENT LECTURES

As momentum builds for GDC Online (formerly GDC Austin) in Texas this October, organizers have announced key new lectures from the world's leading online game firms, including Zynga, Disney, Playdom, Tencent, and more.

The Austin, Texas-based GDC Online conference and expo—taking place October 5–8, 2010—is focused on online games of all kinds—including social network titles, free-to-play web games, kid-friendly online titles, large-scale MMOs, and more.

With a leading advisory board guiding the evaluation and choice

of lectures, and the newly announced GDC Online Awards honoring the leading games in the space, the conference is a must-attend for those working in online games.

Some of the highlights from newly announced sessions—as the August 4th alumni registration deadline for GDC Austin 2008/2009 attendees and speakers approaches—include:

Playdom VP John Donham, most recently at Metaplace and a 20-year veteran of online games, discussing why “developing games for social networks is a dramatic shift from making titles for PCs, consoles, or even the Internet.”

The session will “provide you with a solid basis for revising your strategy as you approach social game development.”

/// In “Scalability for Social Games: YOVILLE, MAFIA WARS, and FARMVILLE,” Zynga’s Robert Zubek expands on his GDC 2010 Summit talk from the leading Facebook game firm to “describe architecture and proven techniques for building scalable server

odds of winning,” alongside “a summary of some of the top publishers and games in China and a broad overview of some of the regulations and regulatory bodies in China.”

/// A practical lecture called “MMO 101: Building Disney’s Server System,” Disney Online Studios’ director of architecture and R&D, Roger Hughston, presents on the system underlying titles like WORLD OF CARS and PIXIE HOLLOW ONLINE. This technical discussion—“with heavy emphasis on server systems”—will describe approaches that have been taken by Disney Online Studios in the development of their MMO environments, including “candid discussion about what is good, bad, and ugly.”

In addition to the main conference content, GDC Online will present specialized summit programs with in-depth business and technical advice on major up-and-coming facets of the game industry, including the 3D Stereoscopic Games Summit, the iPhone Games Summit, the iPad Gaming Summit, and the Game Narrative Summit (formerly the Game Writers Summit).

GDC ONLINE ANNOUNCES “LIVE” TRACK SESSIONS

Organizers of GDC Online (formerly GDC Austin) have announced the first set of lectures for this October’s pre-eminent conference related to online games, including a “Live” track featuring Sony Online, WIZARD101, and IMVU speakers.

While there are already over 25 confirmed lectures across the entire event, organizers are focusing on the “Live” track, which will discuss the vital topic of successful strategies for online games post-launch.

The rise of swiftly iterated social games and microtransactions have led to a wide array of new techniques and technologies that can help increase fun, profitability, and retention; the “Live” track will deal with many of these.

Some of the highlights of the GDC Online “Live” track, as announced thus far, include:

/// In “From SHADOWBANE to WIZARD101: Strategies for Expanding Player Communities and Sustaining Enthusiasm After Launch,” J. Todd Coleman and Josef Hall of KingsIsle Entertainment will reference their 10-million-registered-user online game and previous experience. They will focus on “sustaining a community that transcends genre and generation, the importance of always having new content in queue, and strategies for communicating milestones, and methods for remaining engaged in public conversation.”

/// “Surviving Social Media: Advice From The SOE Playbook” sees Sony Online Entertainment’s Linda Carlson discussing how the FREE REALMS and upcoming DC UNIVERSE ONLINE creators “identify the best outlets for ROI, implement efficiencies in time and money, involve all departments for the common good,

craft audience-specific messaging, execute proven strategies to engage and motivate fans, manage multiple games in social media, maintain brand strength”—with “plenty of war stories and scars to prove points!”

/// IMVU’s Brett Durrett will present “Building A Successful Business After Launch Through Rapid Iteration,” showing how, by “creating systems that utilize A/B testing, instant customer metrics, and a sophisticated deployment process, IMVU is able to shorten the iteration loop and deploy new software to production up to 50 times per day”—leading to an avatar-heavy online environment with over 100 million registered users.

In addition to the main conference content, GDC Online will present specialized Summit programs with in-depth business and technical advice on major up-and-coming facets of the game industry, including 3D stereoscopic games and iPad development.

The event is also hosting the first ever Game Developers Choice Online Awards—honoring the accomplishments of the sometimes overlooked creators and operators of persistent online video games—from large-scale MMOs through free-to-play titles to social network games.

To learn more about the newly announced lectures across all tracks for GDC Online, for which registration is now open, please visit the official GDC Online web site: www.gdconline.com.



of lectures, and the newly announced GDC Online Awards honoring the leading games in the space, the conference is a must-attend for those working in online games.

Some of the highlights from newly announced sessions—as the August 4th alumni registration deadline for GDC Austin 2008/2009 attendees and speakers approaches—include:

/// “AAA To Social Games—Making the Leap” sees

infrastructure, particularly for social web games, operating on the web and social networks.”

/// In a talk called “The Chinese Game Market: Planning To Win,” Tencent Boston VP and GM Jeff Goodwill discusses an overview of Chinese game opportunities from the U.S. division of one of the biggest Chinese firms in the space. The talk will include “practical strategies to improve your



ACADEMY *of* ART UNIVERSITY

FOUNDED IN SAN FRANCISCO 1929 BY ARTISTS FOR ARTISTS



Gaming Artwork by Rob Arnold



Gaming Artwork Collaborative Student Project



Gaming Artwork Collaborative Student Project

TAKE CLASSES ONLINE OR IN SAN FRANCISCO

Advertising
Animation & Visual Effects
Architecture*
Art Education*
Fashion
Fine Art
Game Design
Graphic Design
Illustration
Industrial Design
Interior Architecture & Design
Motion Pictures & Television
Multimedia Communications
Music for Visual Media
Photography
Web Design & New Media

ENROLL NOW

EARN

YOUR AA, BA, BFA, MA, MFA OR
M-ARCH ACCREDITED DEGREE

ENGAGE

IN CONTINUING ART EDUCATION COURSES

EXPLORE

PRE-COLLEGE SCHOLARSHIP PROGRAMS

WWW.ACADEMYART.EDU

800.544.2787

79 NEW MONTGOMERY ST, SAN FRANCISCO, CA 94105

Accredited member WASC, NASAD, Council for Interior
Design Accreditation (BFA-IAD), NAAB (M-ARCH)

**Degree program not currently available online.*



dreamside maroon

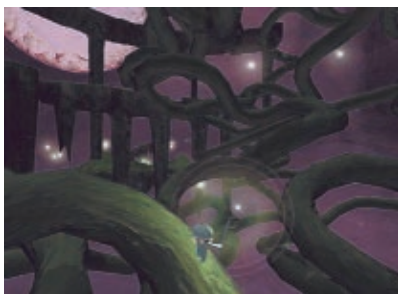
THE TWISTS AND TURNS OF INDIE DEVELOPMENT

CREATED AS A SCHOOL PROJECT BY FOUR STUDENTS FROM THE DIGIPEN INSTITUTE OF TECHNOLOGY, DREAMSIDE MAROON ALLOWS PLAYERS TO EXPLORE A SURREAL LANDSCAPE WHILE RIDING A MAGICAL VINE. THE GAME WAS FEATURED IN THE 2010 IGF STUDENT SHOWCASE AND WAS A FINALIST IN THE 2010 INDIE GAME CHALLENGE. WE TALKED TO TERRACED, THE TEAM BEHIND DREAMSIDE MAROON, TO DISCUSS THE GAME'S ORIGINS, AND WHY THE TEAM CONSCIOUSLY SHIED AWAY FROM TRADITIONAL GAME DESIGN TROPES.

Tom Curtis: Tell me about the team's approach to design for DREAMSIDE MAROON. What process did you use—prototyping? Paper sketches?

Terraced: Our initial meetings started when we were trying to form the team. Both Ian Eller and Matt Anderson had game concepts they wanted to explore, but unfortunately [or perhaps fortunately] there was little to no overlap between their ideas. Additionally, neither was terribly interested in the other's concept. We eventually decided to scrap all [those early designs] and start from scratch.

From there, we held a series of brainstorming sessions where we just generated as many game ideas as possible. The concept of DREAMSIDE MAROON first took root in one of our meetings when Matt suggested, "How about we grow a vine to the Moon?" For whatever reason, this simple thought resonated with all of us. Nobody had any idea how that thought



would translate into a game, but nonetheless, it was our spark. Over the many following brainstorming sessions we continued to flesh out the idea. Would the player climb the vine like Jack and the Bean Stalk? Would this be a platforming game where the player leads the vine around? Would it be 2D or 3D? What should the world look like?

Drawing inspiration from childrens' stories like *Le Petit Prince* and *Harold and the Purple Crayon*, we decided on a surreal, dream-like world of floating islands, where the player would direct the vine's growth using lantern light. One of our professors, Ben Ellinger, helped early on by painting a mental picture of our little character "flying" through swarms of fireflies atop a hearty vine on the way up to the Moon. The atmosphere and the images that Ben conjured up convinced us all. From there, we began building the engine and the game.

TC: What tools did the team use on DREAMSIDE MAROON? Did you build your own engine?

T: It is worth noting that DigiPen is a school, and as such, all of our courses are designed to help us learn. Often times when you tell someone you are taking a game [GAM] class, the first thing that pops into their head is, "Oh, you just play games." Be assured that although it sounds like a joke, the GAM courses are both difficult and time consuming. We create games in order to facilitate learning, and as such, we are limited in the amount of middleware our games can use.

For DREAMSIDE MAROON, we used OpenGL for our graphics API, FMOD for sound, SDL as a windows wrapper, the STL for various containers, and Lua as our scripting language. Outside of those, we wrote the game in its entirety, with the engine in C++ and the higher level gameplay portions in LUA. It was the biggest project any of us had undertaken, as well as the first 3D game we had created.

TC: DREAMSIDE MAROON seems to shy away from concrete player objectives in favor of open exploration. How did this affect your implementation of objectives?

T: Interestingly, it wasn't until the end of the development cycle that we completely opened the game up and removed as many restrictions as possible from the player. Throughout the creation of the game, we kept trying to force the player to do this, that, or the other thing, with the result always being frustration. Our original idea was to direct the player along a loose path by forcing them to "dock" with an island to allow the vine to replenish its nutrients. To facilitate this, various limits were put on how the vine would grow, such as a length cap or a maximum distance it could grow from the last island it had attached to. Everything we tried felt cumbersome and unintuitive.

Finally, after months of almost succeeding, we listened to our focus testers and simply opened up the player's movement, placing elements in the world to entice the player to explore. In hindsight, this was one of the best choices we made. It didn't make much sense to give the player the freedom of flight and then drastically limit where they could go and how they could get there. The result is that the player is left to do as they please with the hope that they want to explore.

TC: The game seems to put a lot of emphasis on creating a mood and telling a story. Did you start with the story, or the mechanics, or did it naturally evolve?

T: Without a doubt, nailing the gameplay was our primary goal. Crafting the atmosphere came as a secondary objective, simply because it was so natural. On the other hand, we had little reference material to help us decide on DREAMSIDE MAROON's mechanics. After about three months of work, we had the basic game engine running, and focus testing began.

Initially, we concentrated simply on making the growth mechanics intuitive, and then gradually progressed toward general gameplay testing. The islands went from being large to small; we removed the "docking" mechanic, along with need to collect seeds; variations on firefly behavior were explored, and lanterns and poetry and dynamic instrumentation were added. It was a long process, and the design phase actually ended only a few weeks before the IGF deadline. Including a story was more of a desire than an objective; we tossed around several different notions, but with gameplay taking priority, we didn't give it extensive thought. The poetry hidden in each lantern represents the themes we would have explored in a more complete narration.

TC: How much playtesting did the team do for the project? What was the overall reaction of players when they were first introduced to the concept?

T: We began a heavy regime of focus testing in January 2009, one that gradually petered out and was closed more or less by April. As was previously mentioned, these playtesting sessions were meant mainly to help us refine the controls, and later on to evolve the overall gameplay design. Through studying our testers, we came to realize our game needed—inverted control options, separate controls for camera and player movement, camera zooming, better feedback for player/world interaction, and an option to turn off the poetry. And of course, we learned to stop trying to limit the distance the player could grow.

—Tom Curtis

DREAMSIDE MAROON:
<http://sites.google.com/site/dreamsidemaroongame>

Game Art & Animation

Associate's Degree

Careers Include:

- Animator
- Modeler
- Technical Animator
- Level Designer

Program Highlights:

- Motion Capture facilities
- Utilizing: Unreal III Engine, Maya, Motion Builder, Mudbox, Body Paint, & more

Additional Emphasis:

- Story development
- Performance
- Cinematography
- Traditional art
- Color theory

Launch your career today!

Madison: 800.236.4997 Minneapolis: 877.416.2783

**MINNEAPOLIS
MEDIA INSTITUTE**
College of Media Arts



**MADISON
MEDIA INSTITUTE**
College of Media Arts

MADISON: 2702 Agriculture Drive | Madison, WI | madisonmedia.edu MINNEAPOLIS: 4100 West 76th St. | Edina, MN | minneapolismediainstitute.com



Piedmont Community College
Digital Effects & Animation
ART 4 GAMES

Learn

- * 3D Modeling
- * 3D Animation
- * 2D Graphics
- * Motion Graphics

Practice

- * 24/7 Animation Lab

Grow

- * Creativity
- * Internships
- * Group Productions



Piedmont Community College
331 College Dr., Yanceyville, NC
336.694-5707
hindmap@piedmontcc.edu
www.piedmontcc.edu



TURN YOUR PASSION FOR GAMING INTO A CAREER

Game Art

Bachelor's Degree Program
Campus & Online

Game Design

Master's Degree Program
Campus

Game Development

Bachelor's Degree Program
Campus

Game Design

Bachelor's Degree Program
Online



Campus Degrees

Master's

Entertainment Business
▶ Game Design

Bachelor's

Computer Animation
Digital Arts & Design
Entertainment Business
Film
▶ Game Art
▶ Game Development
Music Business
Recording Arts
Show Production
Web Design & Development

Associate's

Graphic Design
Recording Engineering

Online Degrees

Master's

Creative Writing
Education Media
Design & Technology
Entertainment Business
Entertainment Business:
with a Sports Management
Elective Track
Internet Marketing
Media Design

Bachelor's

Computer Animation
Entertainment Business

▶ Game Art
▶ Game Design
Graphic Design
Internet Marketing
Music Business
Music Production
Web Design & Development



FULL SAIL
UNIVERSITY

fullsail.edu

Winter Park, FL

800.226.7625 • 3300 University Boulevard

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

Game Design and Interactive Media in the heart of Boston

Our Creative Industries program strives to create, foster, and implement digital media innovation through research, education, and ground-breaking team-based interdisciplinary projects.

Collaborate and create with one of seven combined majors, or the Creative Industries minor.

For more information, go to:
www.ci.neu.edu



Northeastern
CREATIVE INDUSTRIES

Northeastern University
300 Huntington Avenue
Boston, MA 02115-5000



THE LEADING PROFESSIONAL
GAME SOURCE FOR JOB
SEEKERS AND EMPLOYERS

**ACCESS
THE WEB'S
LARGEST
GAME
INDUSTRY
RESUME
DATABASE
AND JOB
BOARD**

 **GAMASUTRA**
The Art & Business of Making Games

 **Take Control of Your Future Today! Log on to www.gamasutra.com/jobs**

**geeked
at birth**





Please geek responsibly.
You may speak the language,
but are you geeked?
Here's a chance to prove it.

Learn

GAME ART & ANIMATION
GAME DESIGN
GAME PRODUCTION AND MANAGEMENT
GAME PROGRAMMING
SERIOUS GAME & SIMULATION

ADVANCING COMPUTER SCIENCE > ARTIFICIAL LIFE >
PROGRAMMING > DIGITAL MEDIA > DIGITAL VIDEO >
ENTERPRISE SOFTWARE DEVELOPMENT > NETWORK
ENGINEERING > NETWORK SECURITY > OPEN SOURCE
TECHNOLOGIES > ROBOTICS & EMBEDDED SYSTEMS >
TECHNOLOGY FORENSICS > VIRTUAL MODELING &
DESIGN > WEB & SOCIAL MEDIA TECHNOLOGIES

www.uat.edu > 877.UAT.GEEK

Baby tattoo Computer (game)

ADVERTISER INDEX

COMPANY NAME	PAGE	COMPANY NAME	PAGE
Academy of Art University	43	Perforce Software.....	14
Bungie.....	11	Piedmont Community College	45
Epic Games	29	Rad Game Tools.....	C4
Full Sail Real World Education	46	Scaleform Corporation	C2
Havok.....	C3	Seapine Software	6
Madison Media Institute.....	45	Simpson Strong-Tie	41
Montreal International Game Summit.....	30	TwoFour54	3
Northeastern University.....	46	University of Advancing Technology.....	47

Game Developer (ISSN 1073-922X) is published monthly by United Business Media LLC, 600 Harrison St., 6th Fl., San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address. Canadian Registered for GST as United Business Media LLC, GST No. R13288078, Customer No. 2116057, Agreement No. 40011901. **SUBSCRIPTION RATES:** Subscription rate for the U.S. is \$49.95 for twelve issues. Countries outside the U.S. must be prepaid in U.S. funds drawn on a U.S. bank or via credit card. Canada/Mexico: \$69.95; all other countries: \$99.95 (issues shipped via air delivery). Periodical postage paid at San Francisco, CA and additional mailing offices. **POSTMASTER:** Send address changes to Game Developer, P.O. Box 1274, Skokie, IL 60076-8274. **CUSTOMER SERVICE:** For subscription orders and changes of address, call toll-free in the U.S. (800) 250-2429 or fax (847) 647-5972. All other countries call (1) (847) 647-5928 or fax (1) (847) 647-5972. Send payments to *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 *Game Developer* on any correspondence. All content, copyright *Game Developer* magazine/United Business Media LLC, unless otherwise indicated. Don't steal any of it.



CONSENSUS REACHED

CHRONICLES OF AN OVERZEALOUS PRODUCER

MEETING REQUEST: SPACE BATTLER - HELMET KICK-OFF

AGENDA: The purpose of this meeting will be to decide how to proceed with the design of the helmet section of the body armor for SPACE BATTLER.

The open questions that will receive a drive toward clarity in this meeting are the following:

1. What kind of helmet is the most optimal to forward the goals for SPACE BATTLER's character, as set out in the SPACE BATTLER Character Bible document (see attached)?
2. What color should the helmet be? (If multiple colors, what color scheme should be employed to ensure that the multiple colors do not clash with each other?)
3. What is the overall purpose of the helmet?
4. Will there be other characters that need helmets? If so, will they share a similar helmet design, or do they need independent helmet designs? Can geometry between helmet types be shared in order to speed up construction time without sacrificing quality? If yes, what percentage of geometry will experience re-use?
5. After art, design, engineering, production, audio, and QA sign-off on a direction for the helmet, what are the next steps and checkpoint dates?

NOTES: SPACE BATTLER - HELMET KICK-OFF

A meeting was held to determine the direction of the helmet design for SPACE BATTLER. The meeting was attended by art, design, engineering, production, audio, and QA. The results of the meeting are as follows.

- It was agreed that the helmet for SPACE BATTLER should be of the "science fiction" type. Since the setting of the game is science fiction, it follows that SPACE BATTLER's helmet should fit in with a science fiction universe.
- According to the writing department, which participated in the meeting via conference call, the purpose of SPACE BATTLER's helmet is to protect his head from possible injury.
- A concern was raised by engineering that an overly complex helmet may reduce the game's frame rate.
- It was agreed upon by all parties that the helmet should "look cool."

WARNING: The color of the helmet is still not decided. Different helmet colors need to be explored before the final helmet color can be locked. However, the art director stated that whatever color the helmet ends up being, it should need "no more than two or three" different colors on it.

A checkpoint date was set for tomorrow.

MEETING REQUEST: SPACE BATTLER - HELMET CHECK-IN

AGENDA: Assess the current progress of SPACE BATTLER's helmet.

1. Is the helmet science fiction, as was decided upon in the kick-off meeting (see notes)? Do we need to adjust course?
2. Helmet color discussion continued.
3. Next steps and any other business.

NOTES: SPACE BATTLER - HELMET CHECK-IN

Progress on the helmet design was reviewed by art, design, engineering, production, audio, and QA. Possible helmet designs were narrowed down to three main options:

1. A helmet with rivets on the side and top that accentuate the feeling of strength possessed by SPACE BATTLER.
2. A helmet with an organic shape meant to evoke the alien technology that Nalana, Princess of Andromeda, gives to SPACE BATTLER. **CONCERN:** Players may not understand that this helmet was designed with alien (non-human) technology. Consider ways to explain SPACE BATTLER's possession of an alien helmet in a follow-up meeting.
3. A helmet made out of garbage that SPACE BATTLER had put together himself from junkyard parts. This would showcase resourcefulness and highlight the severe lack of resources the Earth has after the devastating alien invasion. It could also make him look more heroic, as he is fighting to save the world but with poor equipment.

Another meeting was scheduled to settle on one of the options.

Animation entered the meeting to ask why QA was included in a discussion about helmet design but not an animator.

MEETING REQUEST: SPACE BATTLER - HELMET FOLLOW-UP

AGENDA: Decide on the final helmet design approach and achieve sign-off from all parties.

This is the meeting where the helmet design will be locked. If you have any additional concerns about the helmet design, now is the time to bring them up.

NOTES: SPACE BATTLER - HELMET FOLLOW-UP

The SPACE BATTLER helmet design was reviewed by art, design, engineering, production, audio, and animation with final sign-off as the goal.

1. Animation brought up a concern that none of the options make it possible for SPACE BATTLER to tilt his head.
2. Engineering expressed some reservations about the alien helmet design, saying it might require complex shaders. There was also concern about the junkyard design, as it may require too many polygons. Finally, the riveted design may be too memory-intensive due to its high-resolution normal maps.
3. The color discussion was settled and locked. The helmet will be gray.

MEETING REQUEST: SPACE BATTLER - HELMET PROCESS POSTMORTEM

AGENDA: Discuss the design process of the SPACE BATTLER helmet.

1. What worked well about the SPACE BATTLER Helmet Design Process (SBHDP)?
2. What didn't go as smoothly as hoped?
3. What process improvements can we carry forward into the design process for SPACE BATTLER's kneepads?

NOTES: SPACE BATTLER - HELMET PROCESS POSTMORTEM

Meeting was postponed indefinitely.

Due to time constraints, participants decided to begin the SPACE BATTLER kneepad design immediately and discuss process at a later date. ☹

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

Reliable Pathfinding Technology. Havok AI.



havok[™]
www.havok.com
*Turning Creative Aspirations
into Technical Realities[™]*



THERE'S NOTHING LIKE BEING A



RAD

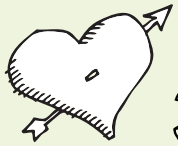
GAME DEVELOPER



It'll feel like you have

SUPER POWERS

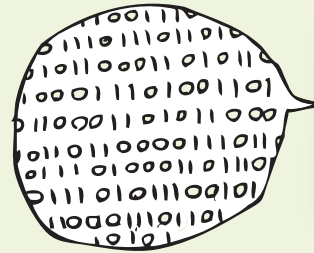
when you use Bink, our amazing, super *FAST* video and audio codec - all in a simple, clean API.



You'll fall in LOVE with Granny, our run-time dynamic

3D ANIMATION SYSTEM

with AMAZING content exporters.



And with like Miles, our multichannel

sound system - you get the world's *FASTEST*

MP3 and Ogg DECODERS

+ way cool **new** high-level audio tools.



.....

So it's obvious, that there's nothing like using RAD tools

Well, nothing except having a killer

MUSTACHE



www.radgametools.com
(425) 893-4300

