DZone

THE 2018 DZONE GUIDE TO

Game Development

BUILDING IMMERSIVE WORLDS

VOLUME I

BROUGHT TO YOU IN PARTNERSHIP WITH

AnswerHub

intel Software

Dear Reader,

With this month's edition of Game Informer magazine open on my desk and a fresh set of tickets to 2018's Game Developer Conference in my pocket, I'm contributing to something we've been looking forward to releasing here at DZone for a couple of years: the first edition of our Game Development Guide!

I will never forget my first video game experiences back in the 1980s – *Space Invaders*, *Donkey Kong*, *Pac-Man*, *Super Mario Brothers*, and my favorite of all time: *The Legend of Zelda*! Growing up during the "Golden Age" of video games meant that I spent much of my childhood meeting friends at the local arcade to enjoy the latest releases. These games bring back some fond memories, and as I think through how much fun the games were, I remember dreaming of writing my own video game someday – which inspired me to become a software developer in the first place.

Game development is more than just software development; it lies at the intersection between software, hardware, creativity, and human interface design. The gaming industry has evolved to push the boundaries in each of these areas, driving innovation through expanding game ecosystems in the 90s, which then accelerated through the turn of the century. Hardware wars were waged between AMD, Intel, and NVIDIA; as well as console wars between Sony, Nintendo, and Microsoft – all driven by the game industry's insatiable desire to deliver more visually stunning and immersive games. Similarly, physics engines, textures, and new rendering technologies were added to game development engines to create more realistic experiences, along with ways to track controllers moving in 3 dimensions or create games with no controller at all (remember the Microsoft Kinect?).

In recent years, we've seen mobile gaming give way to the next revolution: virtual and augmented reality (VR/AR). Once again, new hardware and software innovations are required to support what some view as the holy grail of the video game industry: a truly immersive 3-dimensional world with realistic social interactions – aka "Oasis" as described in *Ready Player One* by Ernest Cline.

In this guide you'll find advice for getting started in game development, tips on creating a positive user experience, ways to handle holograms in Unity's game engine, and evidence to show that as a software developer, you already have sufficient knowledge and experience to build your own games. Whether you're an avid gamer, game developer, or looking to get started creating your own game, we hope you will get as excited about the fascinating world of game development as we did when we assembled this Guide.

By Jesse Davis

EVP TECHNOLOGY, DZONE

Table of Contents

DZONE IS...

PRODUCTION
CHRIS SMITH
DIR. OF PRODUCTION

ANDRE POWELL
SR. PRODUCTION COORD.

G. RYAN SPAIN
PRODUCTION COORD.

ASHLEY SLATE
DESIGN DIR.

BILLY DAVIS
PRODUCTION ASSISSTANT

MARKETING
LAUREN CURATOLA
MARKETING SPECIALIST

KRISTEN PAGÀN
MARKETING SPECIALIST

NATALIE IANNELLO
MARKETING SPECIALIST

JULIAN MORRIS
MARKETING ASSOCIATE

SALES
MATT O'BRIAN
DIR. OF SALES

CHRIS BRUMFIELD
SALES MANAGER

FERAS ABDEL
SALES MANAGER

ALEX CRAFTS
DIR. OF MAJOR ACCOUNTS

JIM HOWARD
SR. ACCOUNT EXECUTIVE

JIM DYER
ACCOUNT EXECUTIVE

ANDREW BARKER
ACCOUNT EXECUTIVE

BRIAN ANDERSON
ACCOUNT EXECUTIVE

RYAN McCOOK
ACCOUNT EXECUTIVE

TOM MARTIN
ACCOUNT MGR.

JASON BUDDAY
ACCOUNT MGR.

BUSINESS AND PRODUCT
RICK ROSS, CEO

MATT SCHMIDT, PRESIDENT

JESSE DAVIS, EVP, TECHNOLOGY

KELLET ATKINSON, MEDIA PRODUCT MANAGER

EDITORIAL
CAITLIN CANDELMO
DIR. OF CONTENT & COMMUNITY

MATT WERNER
PUBLICATIONS COORD.

SARAH DAVIS
PUBLICATIONS ASSOCIATE

MICHAEL THARRINGTON
CONTENT & COMMUNITY MGR.

KARA PHELPS
CONTENT & COMMUNITY MGR.

TOM SMITH
RESEARCH ANALYST

MIKE GATES
SR. CONTENT COORD.

JORDAN BAKER
CONTENT COORD.

ANNE MARIE GLEN
CONTENT COORD.

ANDRE LEE-MOYE
CONTENT COORD.

Executive Summary

BY MATT WERNER
PUBLICATIONS COORDINATOR, DZONE

Just as it's self-explanatory to say there is no software without developers, there are no video games without developers. Yet, DZone has not extensively covered this area of the technology industry, until now. For our first major effort in covering this space, we were most interested in what our readers thought of the topic, and how to make it easier for new game developers to jump in. For DZone's first-ever guide to game development, 587 DZone members replied to our audience survey to tell us about their interest in the topic, what tools they've used if they have developed games, and their platforms of choice.

HOW DEVS PLAY

Data: 3% of DZone users are developing games as part of their core job responsibilities and 9% are developing games in their spare time. 10% used to develop games in their spare time, and 1% used to do it professionally. Those who are interested and not interested in game development are equal at 39%. 71% of respondents play video games, primarily on Windows PC (72%), Android devices (56%), and Playstation 4 (28.5%). Surprisingly, iOS devices were only the fourth most-popular device to play games on at 24%.

Implications: DZone users are split nearly down the middle on their interest in developing games, though a majority do play them. The two most popular platforms, Windows PCs and Android devices, are those with a higher degree of customizability compared to locked systems like consoles or iPhones.

Recommendations: Mobile development still has an enormous amount of interest behind it, and it is relatively easy to reach a huge number of people thanks to the Google Play Store and Apple App Store. If you're interested in game development, especially if you're a Java or C developer, it'll be easier to not only experiment with these platforms, but also pick up some players along the way.

JUMPING INTO GAME DEV

Data: Those who have developed games primarily develop for Windows PC (62%) and Android devices (48%). Game developers are primarily using Java (65%), C/C++ (35%), JavaScript (30%), and C# (30%) to develop their projects, compared to non-game developers who primarily use Java (83%), JavaScript (67%), Python (34%), and Node.js (33%).

Implications: C/C++ and C# are likely more popular among game developers since they're often used with both Unity and the Unreal Engine, both very popular game development engines. Due to the popularity of Java among DZone's general audience, it's not surprising that it's the most popular language in this survey as well. It's also not surprising that JavaScript's popularity on DZone is evident in both game and enterprise developers.

Recommendations: While they are not the only tools, pre-existing knowledge of popular game development languages will be incredibly helpful in creating your own games. It's more important to know how to approach problems than to master a particular tool, so a lack of familiarity with languages like C/C++ is not a blocker, especially since engines like Unity or GameMaker can provide a low-code approach.

THESE ARE THE GAMES THAT DEVS BUILT

Data: Game developers have mostly built puzzle games (45%), action games (37%), or strategy games (28%). Those who are interested in game development or already do it are most interested in developing strategy games (57%), action games (51%), puzzle games (40%), and role-playing games (40%). The genres with the most interest are also the favorite genres of all respondents who play games.

Implications: Action games are one of the more popular game genres and can encompass games as diverse as *Uncharted*, *The Legends of Zelda*, and *Celeste*, so it's not surprising to see a preference for them. Puzzle games can be an easy start to game development that doesn't require intense graphics or physics capabilities.

Recommendations: There's a huge amount of overlap in the preference for action and strategy games for developers and gamers. For those who are interested in either genre, start by trying to build a clone of a popular existing game in the genre, such as the turn-based strategy of *Final Fantasy Tactics* or a "Metroidvania"-style platformer, to get a sense of how these games are built before starting your own original project.

Key Research Findings

BY **G. RYAN SPAIN**, PRODUCTION COORDINATOR, DZONE
AND **JORDAN BAKER**, CONTENT COORDINATOR, DZONE

DEMOGRAPHICS

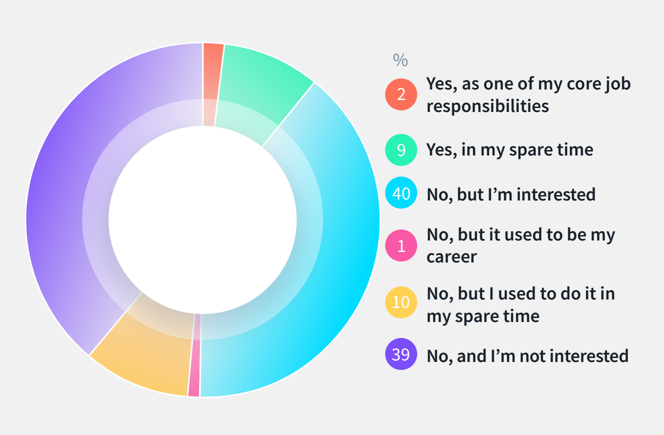
476 software professionals completed DZone's 2018 Game Dev survey. Respondent demographics are as follows:

- 40% of respondents identify as developers or engineers, 20% identify as developer team leads, and 13% identify as software architects.
- The average respondent has 12 years of experience as an IT professional. 48% of respondents have 10 years of experience or more; 30% have 20 years or more.
- 37% of respondents work at companies headquartered in Europe; 32% work in companies headquartered in North America.
- 17% of respondents work at organizations with more than 10,000 employees; 21% work at organizations between 1,000 and 10,000 employees; and 24% work at organizations between 100 and 1,000 employees.
- 79% develop web applications or services; 52% develop enterprise business apps; and 22% are modernizing legacy applications.
- 22% of respondents are currently developing or have developed video games; 40% of respondents are interested in game development.
- 84% work at companies using the Java ecosystem; 67% at companies that use client-side JavaScript; 34% at companies that use Python; and 32% at companies that use server-side JavaScript. 62% of respondents use Java as their primary language at work.

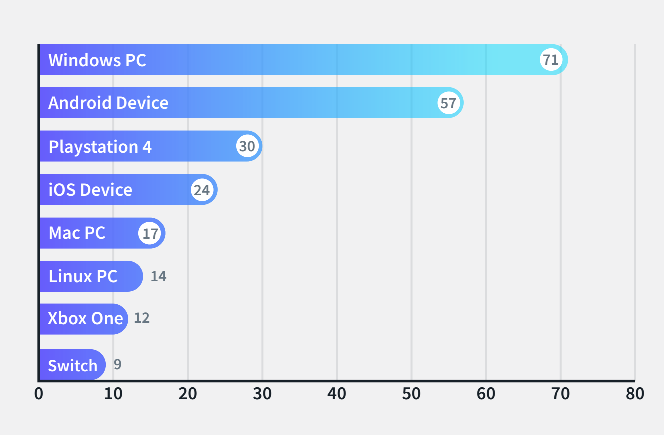
GAME DEVELOPERS

Only 2% of survey respondents said they are currently developing games as part of their core job responsibilities, with an additional 1% of respondents saying that game development used to be their career. Another 19% of respondents are developing (9%) or have developed (10%) games in their spare time. 83% of these spare time developers say they have developed games for personal enjoyment, while only 15% say they have ever tried to commercially distribute their games. Considering that 87% of respondents say they are currently developing web applications/services or enterprise business apps, the lack of respondents who consider themselves "professional game developers" is unsurprising. Furthermore, "game developer" is often a more nebulous term than the developers, engineers, and architects who make up the majority of DZone's survey respondents. The title of game developer can include "…artists and game designers as well as engineers…", and even hobbyists depending on how the description is being defined [Fuller]. And game dev can be a great way for amateur and expert coders alike to sharpen or expand their skills, so it's also unsurprising that the majority of respondents (69%) have spent time or are interested in spending time developing games, even it's just for "personal enjoyment."

GRAPH 01. Are you currently developing a video game?



GRAPH 02. Which platforms do you own and use to play games?



GAMER PREFERENCES & GAME DEV INTEREST

Many more survey respondents (75%) say they play video games than develop them. Most gamers among respondents own and play games on Windows PCs (71%) and Android devices (57%), over the iOS mobile platform (24%); dedicated gaming platforms like the PlayStation 4 (30%) and XBox One (12%); and other personal computer platforms–Mac (18%) and Linux (15%). Respondents who develop/have developed games have done so mostly on Windows PCs (65%) and Android (50%). Game players are most interested in action games (52%), and strategy games (49%), followed by puzzle games, role-playing games, first-person shooters, and sports games, which all garnered between 32%-35% of responses. As far as developers interested in game dev go, strategy games (58%) and action games (52%) are at the top of the list regarding the types of games these respondents are interested in developing, while puzzle games (40%) and RPGs (38%) follow. Those respondents who are developing or have developed games mostly developed puzzle (44%), action (36%), or strategy (29%) games.

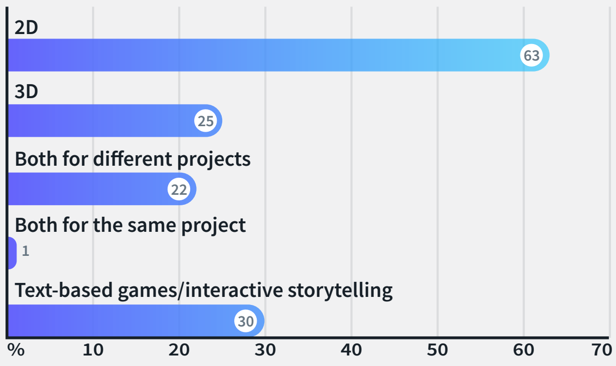
LANGUAGES AND PLATFORMS

Examining the platforms developers are making games for and the languages they use to make those games, some interesting trends emerge. Of the survey respondents who told us they are interested in developing games, the platform that garnered the most interest was Android (70%), while the majority of respondents use Java as their primary language at work (62%). This makes sense, as the Android OS itself is based on Java, and recognizes Java as its official language. Furthermore, the third most mentioned platform among interested game devs is Apple's iOS (41%), yet only one respondent claimed to actively work with Swift. An interesting correlation emerges here, as JavaScript was the third most popular language among our respondents, and the second most popular among those who develop native mobile applications. One possibility is that cross-platform JavaScript frameworks such as React Native are a more popular option among iOS developers than Swift as they allow applications to be deployed on both Android and iOS devices.

MONETIZATION

With the advent of mobile gaming, internet-connected consoles, and platforms like Steam, game monetization has changed dramatically in the past decade. As established earlier, most respondents who are developing or have developed games are not doing so for commercial purposes, and this is reflected in the responses on how games are being monetized. 48% of respondents with game dev experience say their game/games are free-to-play, and another 43% say they don't know yet or that it's too early in development to tell whether the game is free-to-play or pay-to-download. Only 9% of respondents say they are monetizing through pay-to-download games. 83% of respondents say their game/games do not offer unlimited in-game purchases, and 73% say their game/games do not offer one-time in-game purchases such as expansions or additional levels. When asked about personal approval on game monetization strategies, 45% of respondents approved of unlimited in-game purchases; 35% approved of one-time DLC purchases, and another 36% approved of DLC as long as the content was not released and charged for on the game's release.

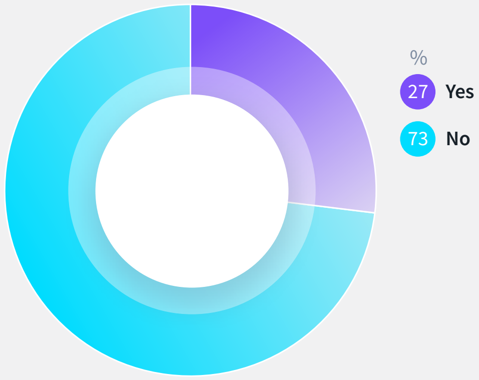
GRAPH 03. Have you developed 2D games (side-scrolling platformers, classic RPGs, etc.), 3D games, or text-based games?



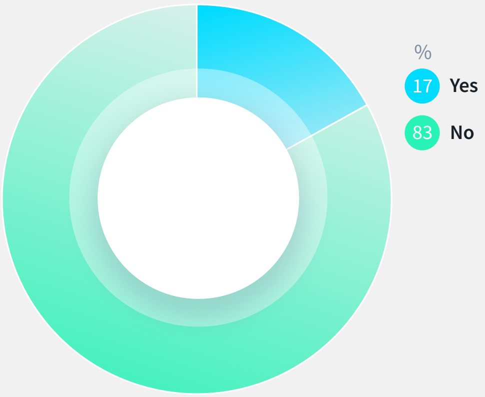
SOURCES

- **Evan Fuller** - quora.com/How-many-game-developers-existed-in-2016
- **Allan Hoffman** - monster.com/career-advice/article/career-spotlight-game-programmer
- gamedev.stackexchange.com/questions/33780/how-many-developers-are-theres

GRAPH 04. Does your game offer one-time in-app purchases, such as expansions, new levels, characters, etc.?



GRAPH 05. Does your game offer unlimited in-game purchases for content?



A Conversation with Nels Anderson

A Designer's Guide to Getting Started in Game Development

BY MATT WERNER
PUBLICATIONS COORDINATOR, DZONE

QUICK VIEW

01. Learn how lead designer of *Mark of the Ninja* and designer on *Firewatch* got his start in the video game industry.

02. What can you do to get started on making your own games or joining a studio?

03. Knowing how to solve problems is more important than knowing specific tools.

04. The player's experience is the most important thing to keep in mind at all times.

Nels Anderson has been developing games professionally for over eight years. His first job out of college was working with HotHead Games on Ron Gilbert's (of *Monkey Island* fame) game *DeathSpank*. From there he became the lead designer of stealth 2D platformer *Mark of the Ninja* in 2012 from Klei Entertainment. He's perhaps best known for his work on 2016 indie hit *Firewatch* from Campo Santo, where his experiences growing up in Wyoming helped shape the environments of the game. He's currently venturing on his own with Caledonia, working on a currently unknown debut project. With his technical background in computer science and experience in the industry, I spoke with Nels to learn about getting started in the industry, why technical experience is important for game design, and focusing on players.

Nels got his start in the industry with a Computer Science degree at the University of Colorado, and became interested in designing games, beginning with supplemental content he created for tabletop games like *Dungeons and Dragons*. He had not programmed anything until starting school. After that, he moved to Vancouver, British Columbia, for graduate school at the University of British Columbia in 2005. During that time, he took the opportunity to volunteer at the first few Penny Arcade Expos (PAX) that were held in Seattle. Since the then-small conference was just starting out, he was able to easily meet with younger Vancouver studios who made the short trip across the border while he was an undergrad, talking about development and asking for jobs. By the time he graduated, he had made a few contacts which he leveraged into a job at HotHead Games.

One thing Nels made explicitly clear is that there's very little need for specific game development education. "There are a few good programs, like DigiPen in Seattle, Carnegie Mellon, and SMU," he explains, "but I question how much they can teach you compared to what you can learn online." There are other ways to learn about game development without an industry-specific education. One piece of advice that Nels recommends is that developers "download a game engine, like Unity or GameMaker, and just make a clone of *Breakout*, or *Tetris*, or something. Just learn how these old classic games work and that'll help you understand a lot. The core of games is 'get input from player, put thing on screen, repeat.'" He also points out that "the only thing stopping you, is you," and recommends going to game jams once you have some basic skills under your belt.

We've come to a similar conclusion at DZone. Our audience survey has found that DZone users who are developing games are using most of the same languages as users who are not developing games, but are interested in doing so. Most enterprise developers who read DZone are using either Java or JavaScript (when I told him about this, Nels recommended working with "anything other than JavaScript"), while most game developers are using Java or C/C++.

When creating a new game, Nels recommends that designers start with an emotion they want the player to experience. "It has to be more complicated than 'fun,'" he explains. "Those are kind of useless descriptors compared to something like 'loneliness,' for example. How do we make the player feel loneliness through mechanics or graphics or music?" That central idea is what your game is about, it can come from an individual or the team, and acts as an anchor for the whole team to keep in mind as they work on mechanics or solve problems, though how this vision is communicated differs by team or company.

"Download a game engine, like Unity or GameMaker, and just make a clone of Breakout, or Tetris, or something. Just learn how these old classic games work and that'll help you understand a lot."

With the central concept of a game figured out, Nels prefers to approach development systemically. To use *FireWatch* as an example, the story and narrative are definitely the core of the game experience, but the way it was actually built was based on systems, figuring out what needs to happen in the game — such as movement, interacting with objects, etc. You need to build an idea, refine it, rebuild it, and refine it again until it's ready. Game development is an incredibly iterative process. When you're a designer, having a technical background can help make these cycles easier.

"I have a very nuts-and-bolts engineering background," says Nels. "When I went to school it was the kind of program where you go to the bookstore, buy a book on C, and just go." However, having that background does not mean it's strictly necessary, or that you even need to be an amazing developer to design games. Nels readily admits that nowadays he's "a very mediocre programmer. But I can create a prototype and pass it to a 'real developer' and tell them what I'm thinking and what I want to happen in the game." However, if you're a non-technical designer, it can add days or weeks to your release schedule if you need someone else to implement an idea to iterate on, rather than creating it yourself. When you're iterating on these ideas, Nels warns that "you can't be married to anything. You're going to make a lot of wrong assumptions, and the only barometer for your success is the person behind the controller."

The experience the person behind the controller has is perhaps the biggest difference between game and enterprise development, other than the overall purpose of the software being created for entertainment vs. business function. As previously mentioned, games need to evoke an emotion from the player, whether it's loneliness, excitement, or dread. While this doesn't seem much different in theory than enterprise software's goal to meet a business need, enterprise software doesn't need to encourage their users to immerse themselves in the experience. Games, on the other hand, need not only strong mechanics, but also strong visuals and UI that don't take them out of the experience. While enterprise software can solve an issue without necessarily being a great experience for the user, that experience is essential to creating successful video games.

"You can't be married to anything. You're going to make a lot of wrong assumptions, and the only barometer for your success is the person behind the controller."

"Ultimately, you're making entertainment," says Nels, "there aren't very clear requirements to figuring out whether something is successful." In other engineering projects, something either works or doesn't. With something creative, "what 'successful' means is a lot more intangible until you've actually built the thing." If you're interested in developing games, even as a hobby, there are more avenues to getting started than ever, and if you have a technical background it'll be even easier.

MATT WERNER is a publications coordinator at DZone. He's been with the company for almost six years, starting in sales before working with the DevOps and Agile zones. He currently manages the content focus, research, production, and release of DZone's series of research guides and Refcardz. He has a BA in Business from Furman University, and in his spare time he's often playing tabletop games with friends or playing bass guitar.

Top 3 Invisible UX Design Decisions That "Level Up" Player Happiness

BY NELSON RODRIGUEZ
GLOBAL DIRECTOR, AKAMAI

QUICK VIEW

01. There are several aspects that go into good games — beyond the design of the game itself — that help keep players happy.

02. Game developers need to optimize game download speed to capture immediate player attention.

03. Fairly balancing game delivery to players at different distances from game servers makes a big impact on player experience.

04. Crippling DDoS attacks against game architecture that knock players offline during prime gaming hours can disrupt player happiness if game companies don't prepare adequately.

A game's user experience starts before a player levers a thumbstick or taps their screen. Game developers have gotten the offline, in-game experience down to a science, but increasing dependence on the internet for exceptional gaming experiences has created new opportunities for developers to win...or for "Game Over."

In the old days, we had unboxings where players could handle the physical components of a game — CDs, cartridges, game guides, maybe even an action figure for special editions — that amped up their excitement.

In the digital era, that's mostly been replaced by a loading bar (Amiibo aside). Even when players get into the game itself, they often require an internet connection for richer gaming experiences such as multiplayer interactions and special events. Internet-connected games can be either more enjoyable or frustrating in equal measure, depending on how well they are managed.

Many obstacles can hinder (or outright stop) players from the first minute they discover a new game on their favorite site, from finding a trusted platform or distributor, to clicking a button for download and installation. Then it's time to actually play: the login, the day one patch, regular game updates, micro-transactions, matchmaking, and other game-related issues could arise to complicate your players' experience. In short, there are many moments and situations that have nothing to do with the game or gameplay itself that can get in the way of someone buying and playing your game.

The goal of any game development team should be to reduce all the points where they can lose players before they get hooked on the game. For this article, let's take a look at some of the infrastructure decisions game developers can make to level up player happiness and ensure that the game they worked so hard to build runs smoothly.

OPTIMIZING GAME DOWNLOAD SPEED TO CAPTURE IMMEDIATE PLAYER ATTENTION

The worst part of downloads is the hair-tearing anticipation as players wonder where their game is coming from and why it takes six hours to get it. Add FOMO (fear of missing out) to the mix on release day, and the odds are stacked against game providers.

Players want to join their friends in being among the first to play. While developers no longer have to worry about stores running out of stock, they do have to worry about server overload. If the initial download rush were to crash their data centers, the technical team would have some explaining to do.

The key to success on launch day is flexibility. It's hard to predict where players will be when they download the game, or when exactly they will push the button. Game companies need to be ready by developing a global-ready infrastructure that puts the game files close to the end users. The fewer hops between the game server and the player, the better the download speed and the lesser chance for failed downloads.

Making the investment in the financial, technological, and human resources required to build that kind of global infrastructure is difficult for most companies, especially if they aren't offering large-scale MMOs that require constant player connectivity. Instead, they should they look for a cloud provider with strong server mirroring capabilities that copies key files in locations around the world. That way the files will be closer to their destinations and no single datacenter will feel the full strain of the download requests.

They should also examine the dependencies between their servers and users such as ISPs and DNS providers to make sure they are all prepared for the influx of traffic. Building strong relationships with them can help ensure all players get equal service when they download the game.

FAIRLY BALANCING GAME DELIVERY TO PLAYERS AT DIFFERENT DISTANCES FROM GAME SERVERS

E-sports have made consistent and equal gameplay a necessity. Beyond the actual game itself is a vast and complex network of systems that cooperate to deliver your customers' gameplay experience. A few microseconds' difference in lag during a fast-paced FPS can make the difference between a headshot and a miss. Players expect reliable connectivity, even when they are only playing for fun.

Matchmaking systems need to take into account not only the players' skill level, but also the players' locations relative to the servers they are using. While last-mile connections — broadband, satellite, 4G — are the player's responsibility, game companies should work with an appropriate partner to ensure speedy packet delivery through the heart of the internet. While it is easy to dismiss the stuff in the middle that they don't control or maintain as someone else's responsibility, the end result of player happiness still belongs to the game developer.

Use the relationships you built to optimize the initial game download in order to understand how the game streaming dependencies differ. When something goes wrong on a network outside the datacenter, knowing who to call to fix it can make the difference between loyal gamers and lost customers. Game developers should watch those connections carefully so they can start responding to any connectivity issues before their players notice.

SECURING GAME ARCHITECTURE FROM CRIPPLING DDOS ATTACKS THAT KNOCK PLAYERS OFFLINE DURING PRIME GAMING HOURS

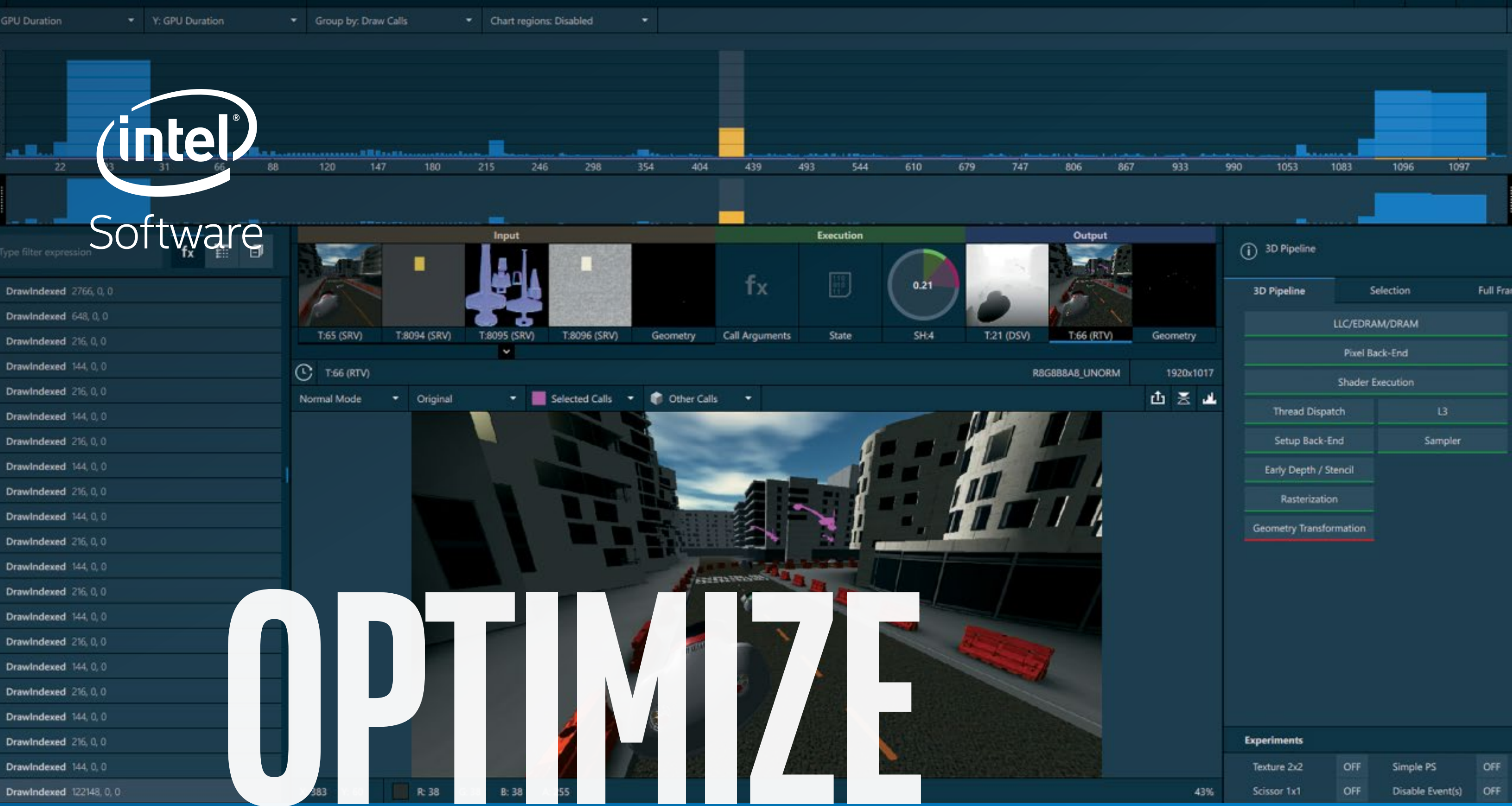
The gaming industry is a constant target of DDoS attacks, with some companies deflecting as many as 2,000 separate attacks in a year. Whatever the reasons for an attack—prestige, ransom, PII exfiltration, or something else — every gaming company needs to shore up their defenses against hackers.

The vast majority of DDoS attacks are volumetric in nature, meaning they employ networks of hijacked devices to send thousands of artificial requests per second to the server. Since the server cannot discern between a human and compromised device, it must spend the same amount of processing power on each request until it reaching the point of ignoring everything and disconnecting legitimate users. It would be impractical for a company to deploy enough computing power to handle a DDoS attack on its own. More cost-effective investments could be made in capabilities that are designed to mitigate these types of attacks and help prevent the servers from becoming overwhelmed.

Whatever service they use, it is important to remember that the time and effort spent to establish these defenses will be the same if they ever need to rebuild them again. It can be tempting to go through a quiet period with no threats from DDoS attacks and decide to stop paying extra for a service adding no visible value. However, a DDoS attack can come from anywhere, at any time, making the ongoing investment worth it.

Building the trust and enthusiasm of loyal fans in the gaming community is the most rewarding experience of game development. But there are a lot of hurdles in the way of that relationship. Players may set out on that journey excited and on your side, but when download speeds slow to a crawl or they can't connect to the server, their first thought will not be to blame the obscure network provider they never interact with — it will be to blame the game. Positive gaming experiences are the sole responsibility of the game developer. There is a lot of work that goes into building that experience before the player even starts to play, but it is essential to the success of the game.

NELSON RODRIGUEZ has helped launch dozens of games across every platform, including blockbusters *Halo 3*, *Tekken 5*, and *Assassin's Creed Brotherhood*, and indie titles *Tweet Defense* and *A Kingdom for Keflings*. After two years on the Xbox marketing team developing social and community strategy, Nelson spent six years creating award-winning digital marketing campaigns for clients such as Ubisoft, Microsoft, Sony, and Hasbro. Nelson currently works as a global director in Akamai Technologies' games industry division.



intel Software

OPTIMIZE TO OUTPERFORM

Improve your game's frame rate and performance with the free Intel® Graphics Performance Analyzers (Intel® GPA).

This set of analyzers gives you everything you need to enhance the gamer experience without having to recompile source code.

And we're constantly making it better—check out the most recent updates:

- New Windows* Mixed Reality profiling support for the utmost in immersive experiences.
- New improved user interface for optimizing DirectX* 11 games.
- New profiling feature, Hotspot Analysis Mode, that easily identifies the most troublesome bottlenecks in a frame.

Free Download

#IntelGPA, #GameDev, #IndieDev @IntelDevTools

Intel® Graphics Performance Analyzers Help Improve Your Game's Performance

From one developer to another, I understand what it's like to always be on the lookout for ways to improve game performance. That's why I'm excited to share the latest release of Intel® Graphics Performance Analyzers (Intel® GPA)—a set of analyzers that specifically help you improve the performance of games and other graphics-intensive applications using graphics APIs like Microsoft DirectX*, Apple Metal*, OpenGL*, and OpenGL ES*. Whatever platform you're using, you can easily debug, analyze, and optimize your games.

Check out these latest features:

- **Improved frame profiling for DirectX 11**: We've added the ability to profile DirectX 11 applications using the DirectX 12 and OpenGL Frame Analyzer UI. This UI provides many new features to the DirectX 11 tool suite, such as resource history, Hotspot Analysis Mode, bar-chart grouping features, and much more.
- **Hotspot Analysis Mode**: This mode essentially groups events by bottlenecks or states, making it easier to identify the most impactful slowdown within the frame.
- **Windows* Mixed Reality profiling support**: Profile Mixed Reality applications on Intel GPA with the new Universal Windows Platform and Mixed Reality Portal support.

Quickly identify problem areas using the Intel GPA tool suite:

- Understand how your system is utilized with real-time performance metrics in **System Analyzer**
- Analyze the performance of a single frame down to the draw call level with **Graphics Frame Analyzer**
- Holistically analyze multiple frames within Metal applications using **Graphics Multiframe Analyzer**
- Identify CPU/GPU contention, queue usage, and synchronization issues with **Graphics Trace Analyzer**

If you have any feedback regarding the new UI, we'd love to hear it on the **Intel GPA forums**. And be sure to download Intel GPA for free on our **website**.

WRITTEN BY SETH SCHNEIDER
PRODUCT OWNER, INTEL® GRAPHICS PERFORMANCE ANALYZERS INTEL CORPORATION

PARTNER SPOTLIGHT

Game Development Tools

intel Software

Analyze, Optimize, and Debug Games. Improve your game's performance by quickly identifying problem areas with Intel® Graphics Performance Analyzers.

CATEGORY	OPEN SOURCE?	RELEASE SCHEDULE
Game Development Tools	No	4x per year

CASE STUDY

Use the full potential of your favorite platform while improving a game's frame rate and performance with this free download. Featuring a convenient panel overlay, you can quickly identify problem areas and experiment with improvements without having to recompile source code.

SYSTEM ANALYZER

Isolate common bottlenecks that affect your game's performance in real time.

GRAPHICS FRAME ANALYZER

Analyze performance on a single frame down to the draw call level.

GRAPHICS MULTIFRAME ANALYZER

Holistically analyze multiple frames within Metal* applications.

GRAPHICS TRACE ANALYZER

Identify where you can evenly distribute workloads across the CPU and GPU.

STRENGTHS

- Free
- Quickly identify problem areas
- Experiment with improvements without having to recompile source code
- Increase performance on Intel® hardware

WEBSITE software.intel.com

TWITTER @IntelSoftware

DOWNLOADS software.intel.com/en-us/gpa/free-download

Dragging and Placing Holograms With Unity

BY JOOST VAN SCHAIK
MIXED REALITY, MOBILE, AND AZURE DEVELOPER

QUICK VIEW

01. Learn how to use Unity and the HoloLens to move objects and place them on surfaces for augmented reality apps or games.

02. Use a simple version of a collision detector to maintain immersion by not letting objects pass through each other.

03. Detailed instructions on setting up a basic project to move objects by gaze and air tap.

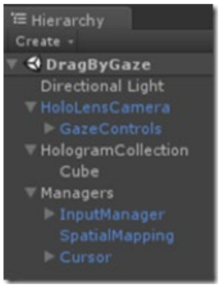
I've been trying to create the effect of what you get in the – in the meantime good old – HoloLens Holograms app: When you pull a hologram out of a menu, it "sticks to your gaze" and follows it. You can air tap, and then it stays hanging in the air where you left it, but you can also put it on a floor, on a table, or next to a wall. You can't push it through a surface. That is, most of the time. **So, like this**.

In the video, you can see that it follows the gaze cursor floating through the air until it hits a wall to the left and then stops, then goes down until it hits the bed, and then stops, then up again until I finally place it on the floor.

A NEW YEAR, A NEW TOOLKIT

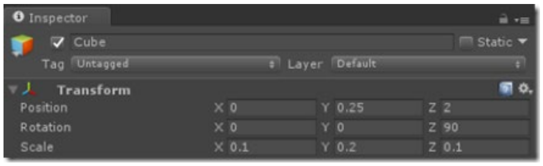
As happens often in the bleeding edge of technology, things tend to change pretty fast. This is also the case in HoloLens country. I have taken the plunge to Unity 5.5 and the new **HoloToolkit**, which has a few big changes. Things have gotten way simpler since the previous iteration. Also, I would like to point out that for this tutorial, I used **the latest patch release**.

SETTING UP THE INITIAL PROJECT



```
▼ DragByGaze
    Directional Light
  ▼ HoloLensCamera
    ▶ GazeControls
  ▼ HologramCollection
      Cube
  ▼ Managers
    ▶ InputManager
      SpatialMapping
    ▶ Cursor
```

This is best illustrated by a picture. If you have set up the project, we only need this. Both Managers and HologramCollection are simply empty game objects meant to group stuff together. They don't have any other specific function here. Drag and drop the four blue prefabs in the indicated places, then set some properties for the cube (see left).



☑ Cube ☐ Static
Tag Untagged Layer Default

▼ Transform

	X	Y	Z
Position	0	0.25	2
Rotation	0	0	90
Scale	0.1	0.2	0.1

The Cube is the thing that will be moved. Now it's time for "some" code.

THE MAIN ACTORS

There are two scripts that play the leading role, with a few supporting roles.

- `MoveByGaze`
- `IntialPlaceByTap`

The first one makes an object move, the second one actually ends it. Apropos, the actual moving is done by our old friend **iTween**, whose usefulness and application was already described in **part 5 of the AMS HoloATC series**. So, you will need to include this in the project to prevent all kinds of nasty errors. Anyway, let's get to the star of the show, MoveByGaze.

MOVING WITH GAZE

It starts like this:

```
using UnityEngine;
using HoloToolkit.Unity.InputModule;
using HoloToolkit.Unity.SpatialMapping;
namespace LocalJoost.HoloToolkitExtensions
{
    public class MoveByGaze : MonoBehaviour
    {
        public float MaxDistance = 2f;
        public bool IsActive = true;
        public float DistanceTrigger = 0.2f;
        public BaseRayStabilizer Stabilizer = null;
        public BaseSpatialMappingCollisionDetector
            CollisonDetector;
        private float _startTime;
        private float _delay = 0.5f;
        private bool _isJustEnabled;
        private Vector3 _lastMoveToLocation;
        private bool _isBusy;
        private SpatialMappingManager MappingManager
        {
```

code continued on next page

```
            get { return SpatialMappingManager.Instance; }
        }
        void OnEnable()
        {
            _isJustEnabled = true;
        }
        void Start()
        {
            _startTime = Time.time + _delay;
            _isJustEnabled = true;
            if (CollisonDetector == null)
            {
                CollisonDetector =
                  gameObject.AddComponent<Default
                  MappingCollisionDetector>();
            }
        }
    }
}
```

Up above are the settings:

- `MaxDistance` is the maximum distance from your head the behavior will try to place the object on a surface. Further than that, and it will just float in the air.

- `IsActive` determines whether the behavior is active (duh).

- `DistanceTrigger` is the distance your gaze has to be from the object you are moving before it actually starts to move. It kind of trails your gaze. This prevents the object from moving in a very nervous way.

- `Stabilizer` is the stabilizer made, used, and maintained by the `InputManager`. You will have to drag the `InputManager` from your scene on this field to use the stabilizer. It’s not mandatory, but it's *highly* recommended.

- `CollisionDetector` is a class we will see later – it basically makes sure the object that you are dragging is not pushed through any surfaces. You will need to add a collision detector to the game object that you are dragging along – or maybe another game object that is part of the game object that you are dragging. That collision detector then needs to be dragged on this field with `MoveByGaze`. This is not mandatory. If you don’t add one, the object you attach the `MoveByGaze` to will just simply follow your gaze and move right through any object. That’s the work of the `DefaultMappingCollisionDetector`, which is essentially a **null pattern** implementation.

Anyway, all the work is done in the Update method:

```
void Update()
{
    if (!IsActive || _isBusy || _startTime > Time.time)
        return;
    _isBusy = true;
```

code continued on next column

```
    var newPos = GetPostionInLookingDirection();
    if ((newPos - _lastMoveToLocation).magnitude >
    DistanceTrigger || _isJustEnabled)
    {
        _isJustEnabled = false;
        var maxDelta = CollisonDetector.GetMaxDelta(newPos
        - transform.position);
        if (maxDelta != Vector3.zero)
        {
            newPos = transform.position + maxDelta;
            iTween.MoveTo(gameObject,
                iTween.Hash("position", newPos, "time",
                2.0f * maxDelta.magnitude,
                    "easetype", iTween.EaseType.
                    easeInOutSine, "islocal", false,
                    "oncomplete", "MovingDone",
                    "oncompletetarget", gameObject));
            _lastMoveToLocation = newPos;
        }
        else
        {
            _isBusy = false;
        }
    }
    else
    {
        _isBusy = false;
    }
}
private void MovingDone()
{
    _isBusy = false;
}
```

We're only doing anything if the behavior is active, not busy, and it's within the first half second. And the first thing is – telling the world we are busy indeed. This method, like all Updates, is called "60 times a second" and we want to keep things a bit controlled here. Race conditions are annoying.

Then we get a position in the direction the user is looking, and if that exceeds the distance trigger – or this is the first time we are getting here – we start off by finding how far ahead along this gaze we can place the actual object by using `CollisionDetector`. If that’s possible – that is, if the `CollisionDetector` does not find any obstacles — we can actually move the object using `iTween`. It's important to note that whenever the move is *not* possible, `_isBusy` immediately gets set to false. Also, note the fact that the smaller the distance, the faster the move. This is to make sure the final tweaks of setting the object in the right place don’t take a long time. Otherwise, `_isBusy` is only reset after a successful move.

Then the final pieces of this behavior:

```
private Vector3 GetPostionInLookingDirection()
{
    RaycastHit hitInfo;
    var headReady = Stabilizer != null
```

code continued on next page

```
        ? Stabilizer.StableRay
        : new Ray(Camera.main.transform.position, Camera.
        main.transform.forward);
    if (MappingManager != null &&
        Physics.Raycast(headReady, out hitInfo,
        MaxDistance, MappingManager.LayerMask))
    {
        return hitInfo.point;
    }
    return CalculatePositionDeadAhead(MaxDistance);
}
private Vector3 CalculatePositionDeadAhead(float distance)
{
    return Stabilizer != null
        ? Stabilizer.StableRay.origin +
             Stabilizer.StableRay.direction.normalized *
             distance
        : Camera.main.transform.position +
            Camera.main.transform.forward.normalized *
            distance;
}
```

`GetPostionInLookingDirection` first tries to determine the direction you're looking in. It tries to use the Stabilizer's `StableRay` for that. The Stabilizer is a component of the `InputManager` that stabilizes your view – and the cursor uses it as well. This prevents the cursor from wobbling too much when you don't keep your head perfectly still (which most people don't – this includes me). The stabilizer takes an average movement of 60 samples, and that makes for a much less nervous-looking experience. If you don't have a stabilizer defined, it just takes your actual looking direction – the camera's position and your looking direction.

Then it tries to see if the resulting ray hits a wall or a floor – but no further than `MaxDistance` away. If it sees a hit, it returns this point, if it does not, if gives a point in the air `MaxDistance` away along an invisible ray coming out of your eyes. That's what `CalculatePositionDeadAhead` does – but also trying to use the Stabilizer first to find the direction.

DETECT COLLISIONS

Okay, so what is this famous collision detector that prevents stuff from being pushed through walls and floors, using the spatial perception that makes the HoloLens such a unique device? It's actually very simple, although it took me a while to actually get it this simple.

```
using UnityEngine;
namespace LocalJoost.HoloToolkitExtensions
{
    public class SpatialMappingCollisionDetector :
    BaseSpatialMappingCollisionDetector
    {
        public float MinDistance = 0.0f;
        private Rigidbody _rigidbody;
        void Start()
        {
            _rigidbody = GetComponent<Rigidbody>() ??
```

code continued on next column

```
            gameObject.AddComponent<Rigidbody>();
            _rigidbody.isKinematic = true;
            _rigidbody.useGravity = false;
        }
        public override bool CheckIfCanMoveBy(Vector3
        delta)
        {
            RaycastHit hitInfo;
            // Sweeptest wisdom from
            //http://answers.unity3d.com/questions/499013/
            cubecasting.html
            return !_rigidbody.SweepTest(delta, out
            hitInfo, delta.magnitude);
        }
        public override Vector3 GetMaxDelta(Vector3 delta)
        {
            RaycastHit hitInfo;
            if(!_rigidbody.SweepTest(delta, out hitInfo,
            delta.magnitude))
            {
               return KeepDistance(delta, hitInfo.point); ;
            }
            delta *= (hitInfo.distance / delta.magnitude);
            for (var i = 0; i <= 9; i += 3)
            {
                var dTest = delta / (i + 1);
                if (!_rigidbody.SweepTest(dTest, out
                hitInfo, dTest.magnitude))
                {
                    return KeepDistance(dTest, hitInfo.
                    point);
                }
            }
            return Vector3.zero;
        }
        private  Vector3 KeepDistance(Vector3 delta,
        Vector3 hitPoint)
        {
            var distanceVector = hitPoint - transform.
            position;
            return delta - (distanceVector.normalized *
            MinDistance);
        }
    }
}
```

This behavior first tries to find a `RigidBody` and, failing that, adds it. We will need this to check the presence of anything "in the way." But – this is important – we will set `isKinematic` to true and `useGravity` to false, or else our object will come under the control of the Unity physics engine and drop on the floor. In this case, we want to control the movement of the object.

So, this class has two public methods (its abstract base class demands that). One, `CheckIfCanMoveBy` (that we don't use now), just says if you can move your object in the intended direction over the intended distance without hitting anything. The other essentially does the same, but if it finds something in the way, it also tries to find a distance over which you can move in the desired direction. For this, we use the `SweepTest` method of `RigidBody`. Essentially, you give it a vector, a distance along that vector, and it has an out variable that gives you info about a hit, should any occur. If a hit does occur, it tries at again at 1/4th, 1/7th, and 1/10th of that initially found distance. Failing everything, it returns a zero vector. By using this rough approach, an object moves quickly in a few steps until it can't move any longer.

And then it also moves the object back over a distance you can set from the editor. This keeps the object just a little above the floor or from the wall. That's what KeepDistance is for.

The whole point of having a base class `BaseSpatialMappingCollisionDetector`, by the way, is a) enabling null pattern implementation which is implemented by `DefaultMappingCollisionDetector` and b) making different collision detectors based upon different needs. It's a bit of architectural consideration within the sometimes-bewildering universe of Unity development.

MAKING IT STOP: `InitialPlaceByTap`

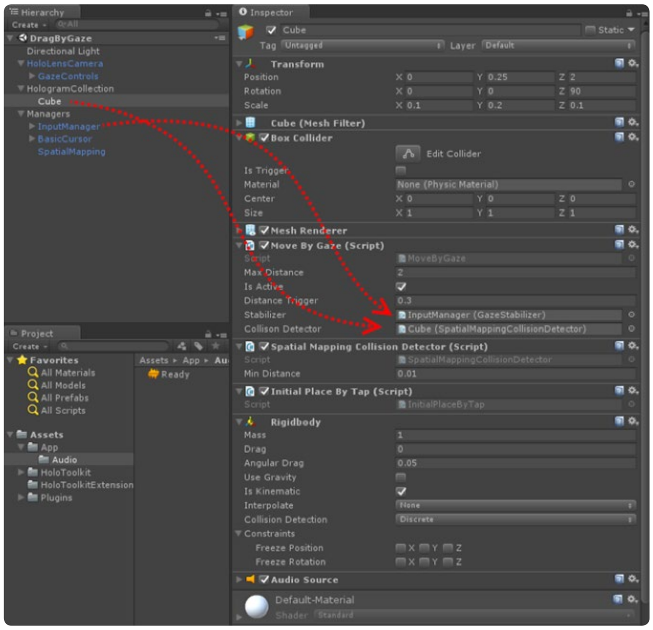
Making the `MoveByGaze` stop is very simple – set the `IsActive` field to false. Now we only need something to actually make that happen. With the HoloToolkit, this is actually very, very simple:

```
using UnityEngine;
using HoloToolkit.Unity.InputModule;
namespace LocalJoost.HoloToolkitExtensions
{
    public class InitialPlaceByTap : MonoBehaviour,
    IInputClickHandler
    {
        protected AudioSource Sound;
        protected MoveByGaze GazeMover;
        void Start()
        {
            Sound = GetComponent<AudioSource>();
            GazeMover = GetComponent<MoveByGaze>();
            InputManager.Instance.
            PushFallbackInputHandler(gameObject);
        }
        public void OnInputClicked(InputEventData
        eventData)
        {
            if (!GazeMover.IsActive)
            {
                return;
            }
            if (Sound != null)
            {
                Sound.Play();
            }
            GazeMover.IsActive = false;
        }
    }
}
```

By implementing `IInputClickHandler`, the `InputManager` will send an event to this object when you air tap it and it is selected by gaze. But by pushing it as the fallback handler, it will also get this event when it's not selected. The event processing is pretty simple – if the `GazeMover` in this object is active, it's de-activated. Also, if there's an `AudioSource` detected, its sound is played. I very much recommend this kind of audio feedback.

WIRING IT ALL TOGETHER

On your cube, drag the `MoveByGaze`, `SpatialMappingCollisionDetector`, and `InitialPlaceByTap scripts`. Then, drag the cube again on the `CollisionDetector` field of `MoveByGaze`, and the `InputManager` on the Stabilizer field. Unity will select the right component.



☑ Cube □ Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0.25 Z 2
Rotation X 0 Y 0 Z 90
Scale X 0.1 Y 0.2 Z 0.1
Cube (Mesh Filter)
☑ Box Collider
Is Trigger □
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
☑ Mesh Renderer
☑ Move By Gaze (Script)
Script MoveByGaze
Max Distance 2
Is Active ☑
Distance Trigger 0.3
Stabilizer InputManager (GazeStabilizer)
Collison Detector Cube (SpatialMappingCollisionDetector)
☑ Spatial Mapping Collision Detector (Script)
Script SpatialMappingCollisionDetector
Min Distance 0.01
☑ Initial Place By Tap (Script)
Script InitialPlaceByTap
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity □
Is Kinematic ☑
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position □ X □ Y □ Z
Freeze Rotation □ X □ Y □ Z
☑ Audio Source
Default-Material
Shader Standard

So, in this case, I could also have used `GetComponent<SpatialMappingCollisionDetector>` instead of a field where you need to drag something on. But this way is more flexible – in-app, I did not want to use the whole object's collider, but only that of a child object. Note that I have set the `MinDistance` for the `SpatialMappingCollisionDetector` for 1 cm – it will keep an extra centimeter of distance from the wall or the floor.

CONCLUDING REMARKS

So, this is how you can more or less replicate part of the behavior of the Holograms App by moving objects around with your gaze and placing them on surfaces using air tap. The unique capabilities of the HoloLens allow us to place objects next to or on top of physical objects, and the HoloToolkit makes using those capabilities pretty easy.

The full code, as per my MVP "trademark," can be found here.

JOOST VAN SCHAIK has over 25 years in the IT business. Managed to avoid job roles as team manager, project leader, architect – he still does real work and code. Coming from a Geographical Information Systems, he loves strong graphic data visualizations. He has had various love affairs with web development, UWP, XAML in general, Xamarin Forms, and currently Mixed Reality development. He has also have been a Microsoft MVP for 7 years.

Interview With Penny De Byl

Creating Convincing AI

BY MATT WERNER
PUBLICATIONS COORDINATOR, DZONE

QUICK VIEW

01. Why AI is important for believable worlds but can also create issues with immersion.

02. Emotion drives a lot of human behavior, so AI should act in the same way.

03. The key is condensing situations and variables into emotional experiences for AI characters.

04. Do your math homework, what you learned will show up in AI and video games.

05. Player experience is important, if you can create an illusion with three lines of code instead of a huge system, do it.

Dr. Penny De Byl is a Professor of Games and Media at Australia's Bond University, whose books include *Holistic Game Development* and *Programming Believable Characters for Computer Games*. She also teaches several online courses through Udemy and **holistic3d.com**. I spoke with Penny about her research in AI, the player experience, and how character AI might evolve.

Could you give us a quick review of your resume?

I was an academic for about twenty-five years until last year, I completed a PhD focused on artificial intelligence in game characters, and an honor's degree before that in computer graphics. Since then I've been teaching programming in games and computer science stuff. Last year I decided to branch out on my own, teaching online courses in game development, and it's going really well.

When games became popular around the late 90s, when they became quite visual, running on more devices, and getting traction, academics were starting to take them more seriously as a research domain. I found that teaching programming with these visuals was really appealing to people who didn't "get" programming and didn't want to deal with the math without seeing results, and games give you immediate results on the screen, which was really motivating for them.

I teach a lot of beginners, but we do touch on AI in getting characters to move around and act and behave in the environment. Not a lot of the hardcore stuff. I just finished writing a course (**you can find it here**) that looks at neural networks and machine learning in Unity. I'm trying to make it as visual as possible, not to "dumb it down," but to make it more accessible. We want to get rid of all those big scary equations in research papers and say: "this is what it means, this is what the code looks like, and it's not that scary after all."

What are some issues that come up in AI and game characters?

Well there are two issues with emotions and AI, there's modeling how the user is feeling at the time and there's also making the agent itself behave emotionally. You've got recognition of emotions and you've got generation of emotions. I did do a research paper on modeling emotions about two years ago, and that was a whole journal looking at the psychology around games and game characters. I guess it's a matter of figuring out what to do with emotions.

There are a lot of AIs out there commercially that aren't quite ready, so they get a bad rap when they behave unpredictably or incorrectly, which is a lot of the issue with AIs in public. Even with the new Unity ML agents, they're in this developmental and experimental phase, but I noticed in the footnotes in the documentation they make it clear that this is an experimental system and they can't guarantee what your AI are going to do if you release them. Then you think of the old days where you have characters that are bumping into walls constantly, and they just look completely stupid. It's getting there, and there are areas where you can apply it and get away with it, and there are areas where if you push the boundaries too much they can come off looking like they haven't been programmed at all.

What spurred your interest in researching AI?

I happened to be at a university where there were a lot of academics studying computer graphics and AI, so that's where I got involved and interested on it. Then, when I went to do my PhD, I was looking at modeling or simulating different organizations but on the people level. Kind of like *The Sims*, but in a serious scenario where you're looking at how people are moving around an environment and behaving during the day and simulating that.

What drives human behavior? The answer is "emotion." We were looking at how people make decisions and found that emotion comes into it a lot. When people make decisions, they tend to make them quickly, and they're based on a lot of background information which has been stored as emotional reactions to things, whereas AIs have to make a decision based on an awful lot of parameters. We want to condense those parameters into an emotional reaction that they might've had. So, if we created a situation, the AI made a decision based on variables, and the outcome was "good," we'd say the agent was "happy" with that experience, so the next time a similar situation comes up, it knows it was "happy" and it doesn't need to calculate it again.

Emotion in AI also appealed to me from a female point of view, not to make it about the gender debate in computer science, but I found AI to be really dry when I was working with it. It was all focused around these equations and code, and I just thought it was really boring. As you may know, a lot of girls play *The Sims*, and it's been great at getting them into gaming. I guess that being able to visualize what's happening with little people walking around was really appealing to me. It helped me look at AI from an angle other than the dry, abstract stuff underneath it.

You mentioned earlier seeing issues with AI in game characters like walking into walls, so in your experience what's made AI feel natural to people playing games?

The classic problem is nonreactive AI. It's not even stupidity, because stupidity is a level of intelligence. It's just programmed behavior like a machine instead of what you'd expect a human to do. You keep coming back to the Turing Test. Can an AI completely fool you into thinking it's another person in a multiplayer environment?

> When people make decisions, they tend to make them quickly, and they're based on a lot of background information which has been stored as emotional reactions to things, whereas AIs have to make a decision based on an awful lot of parameters.

If the player becomes immersed in a world, and an AI does something it shouldn't do, it throws people out of that immersive state. Even programmed stupidity is a good thing, because people do stupid things in life and MMOs and games. You include spelling and grammar mistakes when they chat, because that's what real people do. Programming bots to run around an environment doesn't really require AI algorithms, just pathway scripting and putting in random tweaks to make them do different stuff. It's way more controllable than using adaptive algorithms, and a programmer has more control over what an AI is going to do. What machine learning can do in games is observe, live, how other people are playing and adapt to that on the fly. But, the challenge is monitoring how that adaptation happens.

In neural networks and genetics algorithms there's always a chance that a mutation happens in order to explore the whole problem space, and sometimes that mutation can create really weird behaviors that you don't want the player to see, but that also restricts how a character learns and evolves. Having said that, if you come back to games like *Spore*, the player is interacting with a genetics-based game that operates in a similar way and it's quite exciting to see mutations happen.

Is there any sort of "uncanny valley" for AI behavior where characters can seem too real and make people uncomfortable? We're definitely starting to get that with computer graphics, particularly in some movies.

That's an interesting question, and it would be a good research area to see if there is. I guess the AI could become really creepy. The thing is, you'd have to know it was an AI, and then know it was acting inappropriately or that it was doing something you didn't feel it was capable of. If you knew a character was a machine or AI and it was having an emotional — or overly emotional — reaction to something it would become quite creepy. Even those humanoid-type robots that you see coming out, they're kind of creepy because they can tell you how you feel. In a game you'd have to be aware it was AI-controlled. With the uncanny valley in graphics you can still tell it's not real but if you're in an MMO for example you may not know if it's real or not. Now I'm getting creeped out thinking about it!

> A lot of developers build prototypes with cubes and then toss them away if it doesn't work. If you spend all your time creating beautiful art assets, then you'll waste a lot of time.

There are characters that are scripted to have emotional responses to you that can make you grow attached to them, so if they die or something you become sad. It would be weird for this thing to adapt to you and for you to become aware of it. That's interesting, I haven't thought about that before.

When you've worked with students, how important has programming AI for NPCs been for the overall experience of the game?

I'd say quite a lot because they want to make intelligent characters, or believable characters, and they feel to do that there need to be NPCs that act believably. Most students start off wanting to make this massively multiplayer first-person shooter that they have to do in 13 weeks, and by the end they only have one character or one AI. They want an AI or group of AIs to be a good opponent for players in the game. All games my students have ever created haven't been puzzles, but instead need an NPC.

Part of their idea of a game is that it needs to be competitive, there needs to be a worthy opponent. Most of the AI techniques we start with are the "patrol/chase/shoot/run away" stuff. That's where I start teaching because they can see how to create a behavior, which mathematically and programmatically is simple but still compelling.

What advice would you give to developers who are looking to create their first game with respect to AI or just generally?

Prototyping is very important. If you've got a whole bunch of cubes running around on a map that's all you need. It's about getting timing, behavior, virtual spatial aspects, and gameplay right. A student will want everything to look spectacular right away, but if the gameplay isn't there, then it's a waste of time. A lot of developers build prototypes with cubes and then toss them away if it doesn't work. If you spend all your time creating beautiful art assets, then you'll waste a lot of time.

With respect to AI, you should start with a simple algorithm and then build on it later if you need to. Often, so much in the background of computer games is faked. It's about what you can make the player believe what is happening, not what you've actually programmed. If you can make your character look intelligent by walking down some stairs with three lines of code, then do that. It's not about having these fancy algorithms if it has the same effect as far as the player's concerned. Having said that, the point of having clever AI is so the player can't tell whether it's just been scripted or there's something else going on there. Overall, it's about player experience, not how clever you are as a programmer.

The really important thing when getting into AI is to know your math. So many people see the math and get intimidated, and so much of this is driven by mathematics. If you get into games, you'll see what you learned in school every single day and you'll be thankful that you learned it.

Penny De Byl's online courses and books can be found at **holistic3d.com**.

MATT WERNER is a publications coordinator at DZone. He's been with the company for almost six years, starting in sales before working with the DevOps and Agile zones. He currently manages the content focus, research, production, and release of DZone's series of research guides and Refcardz. He has a BA in Business from Furman University, and in his spare time he's often playing tabletop games with friends or playing bass guitar.

DIVING DEEPER

INTO GAME DEVELOPMENT

#GAMEDEV TWITTER ACCOUNTS

@br

@drgamermom

@SeamusBlackley

@CeliaHodent

@MckKirk

@darrencearnaigh

@romero

@Joel_Couture

@tha_rami

@Laralyn

GAME DEV ZONES

Web Dev Zone

dzone.com/webdev

Web professionals make up one of the largest sections of IT audiences; we are collecting content that helps Web professionals navigate in a world of quickly changing language protocols, trending frameworks, and new standards for UX.

Mobile Zone

dzone.com/mobile

The Mobile Zone features the most current content for mobile developers. Here you'll find expert opinions on the latest mobile platforms, including Android, iOS, and Windows Phone. You can find in-depth code tutorials, editorials spotlighting the latest development trends, and insight on upcoming OS releases.

AI Zone

dzone.com/ai

The Artificial Intelligence (AI) Zone features all aspects of AI pertaining to Machine Learning, Natural Language Processing, and Cognitive Computing. The AI Zone goes beyond the buzz and provides practical applications of chatbots, deep learning, knowledge engineering, and neural networks.

GAME DEV REFCARDZ

Core C++

Provides an overview of key aspects of C++, and is aimed at existing C++ programmers and object-oriented developers looking to transition to C++.

Java Performance Optimization

Getting Java apps to run is one thing. But getting them to run fast is another. Performance is a tricky beast in any object-oriented environment, but the complexity of the JVM adds a whole new level of performance-tweaking trickiness — and opportunity.

Getting Started With Kotlin

Kotlin has become one of the most popular JVM languages in the past few months, partly because it experienced a lot of attention in the Android community after Google made Kotlin an official language for Android development.

GAME DEV PODCASTS

Building the Game

In this weekly podcast, listen as two game designers discuss game mechanics, getting games published for the first time, and more.

The Game Developers Radio

This podcast about game development features episodes on autotelic emergent lessons, VR, minimalist games, and more.

Game Dev Loadout

This weekly podcast features interviews with game industry professionals to help you enhance your gaming career.

GAME DEV RESOURCES

Extra Credits

This frequently updated YouTube channel takes a deeper look into games, how they're made, what they mean, and how they can be made better.

Game Programming Patterns

This free book covers various common programming patterns that are all related to game development.

Three Hundred Mechanics

The creator of this website went on a quest to design and write 300 game concepts. This is documentation of his ideas and design process.

GAMING THE SYSTEM

IN OUR FIRST EVER GAME DEVELOPMENT SURVEY OF DZONE'S AUDIENCE, WE LEARNED THAT 72% OF OUR READERS PLAY VIDEO GAMES IN THEIR SPARE TIME. WE ALSO DISCOVERED THAT 12% OF OUR READERS ARE CURRENTLY DEVELOPING VIDEO GAMES, 11% USED TO DO IT, AND 39% ARE INTERESTED IN PURSUING IT AS EITHER A HOBBY OR CAREER. STARTING TO DEVELOP A GAME MAY SEEM DAUNTING AT FIRST, BUT WE WANT TO LET YOU KNOW THAT IF YOU'RE PASSIONATE ABOUT GAMES, YOU'RE PROBABLY CUT OUT FOR IT! TAKE A LOOK AT WHAT FELLOW DZONE MEMBERS ARE USING TO CREATE THEIR GAMES, AND HOW THEY OVERLAP WITH WHAT PEOPLE ARE INTERESTED IN — OR ARE ALREADY — USING. MAYBE THIS WILL GIVE YOU THE CONFIDENCE TO LEVEL UP AND TRY YOUR HAND AT BRINGING YOUR BIG IDEA TO LIFE.

IDEAS

OF DZONE USERS WHO HAVE DEVELOPED GAMES, MOST HAVE BUILT **PUZZLE GAMES** (45%) OR **ACTION GAMES** (38%), THE LATTER OF WHICH CAN INCLUDE 2D/3D PLATFORMERS OR ADVENTURE GAMES.

HOWEVER, THE HEARTS OF OUR MEMBERS BELONG TO **STRATEGY GAMES**, WITH 57% OF SURVEY RESPONDENTS EXPRESSING INTEREST. THEY WERE ALSO INTERESTED IN MAKING **ACTION GAMES** (51%), PUZZLE (40%), AND **ROLE-PLAYING GAMES** (40%).

PLATFORMS

THOSE DEVELOPING GAMES ARE PICKING PLATFORMS WITH A LOW BARRIER TO ENTRY, PARTICULARLY **WINDOWS** (63%), **ANDROID DEVICES** (49%), AND **LINUX** (26%), RATHER THAN CONSOLES OR IOS DEVICES WHICH REQUIRE A DEVELOPER PROGRAM MEMBERSHIP AND CERTIFICATION.

DZONE USERS INTERESTED IN GAME DEVELOPMENT ARE MOST INTERESTED IN DEVELOPING FOR **ANDROID DEVICES** (68%), FOLLOWED BY **WINDOWS PCs** (62%).

TOOLS

DZONE READERS WHO DEVELOP GAMES ARE MOST LIKELY USING **JAVA** (65%), **C/C++** (36%), **JAVASCRIPT** (30%), AND **C#** (30%). BY CONTRAST, THE MOST POPULAR LANGUAGES ACROSS THE BOARD ARE **JAVA** (84%), **JAVASCRIPT** (67%), AND **PYTHON** (34%).

78.5% OF GAME DEVELOPERS ARE USING GAME ENGINES, MOST LIKELY **UNITY** (47%) OR **UNREAL** (24%). GAME ENGINES CAN PROVIDE SEVERAL ASSETS TO INCLUDE IN YOUR GAME, INCLUDING PHYSICS, 3D MODELS, AND LOW-CODE INTERFACES.

Executive Insights on the State of Game Dev

BY TOM SMITH
RESEARCH ANALYST AT **DZONE**

QUICK VIEW

01. The keys to successful game development are stability, scalability, and the opportunity for social interaction to increase time played and game stickiness.

02. AR/VR are the most significant changes in, and the future of, game development though its early in their development lifecycles.

03. The most common hurdles affecting game development are 1) "game glut;" 2) building out the games; and, 3) latency.

DZONE.COM/GUIDES

To gather insights on the current and future state of Game Development, we talked to ten executives involved in game development in some form or another. Here's who we spoke to:

- **Sid Sharma**, Lead Developer Evangelist, Agora.io
- **Joseph Lieberman**, Director of Marketing, Antlion Audio
- **Otakar Nieder**, Senior Director, BISim
- **Perry Krug**, Principal Architect, Couchbase
- **Patric Palm**, CEO and Co-founder, Favro
- **Doug Pearson**, CTO, FlowPlay
- **David Lord**, CEO, JumpStart Games, Inc.
- **Brian Monnin**, Co-founder and CEO, Play Impossible
- **George Buckenham**, Lead Programmer, Sensible Object
- **Grant Shonkwiler**, Commander and Shonk, Shonkventures

KEY FINDINGS

1. The three most frequently mentioned keys to developing a successful game are: **1) stability; 2) scalability; and, 3) social interaction/community**. Performance with the ability to get players on board quickly with a fast, responsive game. High availability with no downtime due to maintenance or outages. Stability and scalability so people can play/train for several hours at a time and so the developer can keep up with changing technologies and the game industry.

Social features and communities can create gaming experience with more interactivity with friends and gaming partners, including meaningful opportunities for face-to-face play and periodic events or tournaments. This increases the amount of time played and stickiness of the game – we saw this with the original Xbox. We want to get players to become emotionally connected to the game, and creating more engagement with people and the community helps accomplish this.

2. AR/VR are the most significant changes in game development, even though it's still early in their development lifecycles. More powerful tools enable developers to create more immersive experiences with AR and VR. Unreal Engine VR and Unity EditorVR are great examples. VR has broken through in the art community and now has its own category at Cannes. Apple is including AR capabilities with the new iPhone, making it easier for game developers to hack away at cool apps and games.

Mobile was the "game changer" before that, since app stores determine the success of failure of a game. We've moved from thousands of channels to promote a game to just two or three.

We're also seeing more interactivity among gamers in poker, first-person shooter games, and role-playing games. You must be able to see facial expressions when playing poker. Up to now there were device limitations. Now we have more powerful devices with little to no latency. The technology and evolving user experience is ever-changing and improving.

3. The most frequently mentioned game development platform is the **Unity game engine**, and the most common language is C++. Unity allows for cross-platform distribution on PC, mobile, and consoles. Microsoft Azure Cloud is a popular choice for a backend gaming platform because of its global footprint, reliable scale, technology diversity, development synergy, and deep commitment to partnership. Haxe is being adopted by some developers that were working with Flash.

4. Our respondents' games are **as diverse as the industry**, spanning genres like murder mystery, survival horror, first-person action adventure, and multi-player racing. Social gaming has been huge in the Asian market for some time. We're just beginning to see the rise in social interaction and community games in the U.S. A combination of digital and physical play creates unique challenges for the players and ensures no two games are ever the same.

5. The most common hurdles affecting game development are **1) "game glut;" 2) building out the game; 3) latency; and, 4) engagement**.

The independent game development bubble is gone, so smaller developers have a lot of visibility issues to contend with. While the overall games market continues to grow, it is being outpaced by the production of new games at an ever-increasing rate.

"Game glut" has driven down average prices, decreased the time you have to shine, and made it more difficult to be noticed. It takes something really special to jump to the top and be a hit, and your game MUST be a hit to win. As the game landscape gets more crowded, it's more difficult for the end user to discover your game.

It's always challenging to get your idea through production with workflow, testing, fixing what doesn't work, and planning modifications. You only get one launch, so a lot of testing needs to take place to validate your idea and your game prior to launch. It can be easier for developers to use out-of-the-box solutions so you don't have to worry about latency and performance.

Like all products and services, games need to make an emotional connection with the players. Smart organizations are collecting and analyzing data that will provide insights into how to provide a more personalized and relevant game experience.

6. The future of game development is **AR/VR**. It's huge and it's here to stay since it offers a more immersive experience that engulfs the player. AR/VR has created huge hits like Pokemon Go, which has paved the way for a Harry Potter-based AR experience to be released in 2018, and more AR devices are becoming available.

Keep your eyes open for the next big distribution opportunity. It's been years since major twists like Steam and the Apple App Store, so we are due for disruption over the next three to five years. This new distribution will reach new audiences and demographics that haven't played games before. VR should be ready for prime time by then.

7. A couple of concerns with the state of game development today is the fear of **smaller developers getting pushed out and the ethical choice of "slot machine" (or "gacha") economics**. How should games be monetized, and how do we get more paying users for non-ad-supported experiences?

When it comes to the general gaming community, players are constantly looking for the next best thing. While being able to provide content in a timely manner is important, it's also critical to create cutting-edge features for the game that engage players. We need to be able to produce what players, and non-players, want so we can grow the market.

8. As usual, respondents had several different suggestions for skills developers need to be successful for developing games. The only suggestions mentioned more than once were **knowing the Unity platform, the different levels of programming and design, and other cross-platform solutions**. That, and a lot of patience and persistence. Successful game development is a lot like becoming an overnight rock star who was discovered after 20 years of playing small venues for little to no money.

As always, great user experiences (UX), fun games, and execution are the key drivers to success. Developers need to be able to deliver that experience, and that means being a key part of the game development team, be game players, and understand current trends. Understand why a trend became a trend and why its appealing. Get under the hood and think about the game from a player's point of view or just as entertainment.

9. Some final thoughts from our contributors were: 1) gaming is a **highly competitive industry** with a lot of different roles and opportunities – none of which are easy to be successful in; 2) **know the business model for monetizing** your game upfront, since this will affect the design progression; and, 3) think about the **business aspects of your game** – especially how to gain an audience that is willing to pay and play.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

Solutions Directory

This directory of game development engines, along with graphics, audio, and physics tools, provides comprehensive, factual comparisons of data gathered from third-party sources and the tool creators' organizations. Solutions in the directory are selected based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

GAME ENGINES

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
8Cell	BuildBox	Low-code mobile game engine	Available by request	buildbox.com/buildbox
Allegro Library	Allegro Library	Abstration library for game engines	Open source	liballeg.org
Amazon	Lumberyard Engine	Cross-platform game engine	Free solution	aws.amazon.com/lumberyard
ammo.js	ammo.js	JavaScript physics engine	Open source	github.com/kripken/ammo.js
Apple	SpriteKit	2D mobile game engine	Free solution	developer.apple.com/spritekit
Apple	SpriteKit	3D game and physics engine	Free solution	developer.apple.com/scenekit
Box2D	Box2D	2D Physics engine	Open source	github.com/erincatto/Box2D
Bullet	Bullet	Physics engine	Open source	bulletphysics.org/wordpress
Cocos2d-x	Cocos2d-x	C++ game development framework	Open source	cocos2d-x.org/cocos2dx
Cocos2d-x	Cocos Creator	Game engine, development tools, and workflow	Free solution	cocos2d-x.org/products
Corona Labs	Corona SDK	2D Mobile development framework	Free solution	coronalabs.com
CraftyJS	CraftyJS	JavaScript web game framework	Open source	craftyjs.com
Crytek	CryEngine	Cross-platform game engine	Free solution	cryengine.com

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Duality	Duality	2D game engine	Open source	duality.adamslair.net
ENIGMA	ENIGMA	Game development environment	Open source	enigma-dev.org
Epic Games	Unreal Engine 4	Cross-platform game engine	Free tier available	unrealengine.com/en-US/blog
Gamebase	Gamebryo	Cross-platform game engine	Available by request	gamebryo.com
GameSalad	GameSalad	Low-code game engine	Free tier available	gamesalad.com
GarageGames	Torque 3D	Windows and browser-based 3D game engine	Open source	torque3d.org
GarageGames	Torque 2D	2D game engine	Open source	github.com/GarageGames/Torque2D
GDevelop	GDevelop	Low-code game engine	Open source	compilgames.net
Godot	Godot	2D and 3D game engine	Open source	godotengine.org
Google	Daydream SDK	Mobile-based VR SDK	Free solution	vr.google.com/daydream/developers
Haxe	Haxe	Cross-platform toolkit and programming language	Open source	haxe.org
HaxeFlixel	HaxeFlixel	2D game engine for use with Haxe and OpenFL	Open source	haxeflixel.com
Intel	Intel C++ Compiler	Game optimization and porting	Free solution	software.intel.com/en-us/c-compilers
jMonkeyEngine	jMonkeyEngine	Java-based game engine	Open source	jmonkeyengine.org/
Kiwi.js	Kiwi.js	HTML5 game engine	Open source	kiwijs.org
libGDX	libGDX	Mobile, web, and Java game development framework	Open source	libgdx.badlogicgames.com
Loom	Loom SDK	SDK, workflow, scripting platform	Open source	loomsdk.com
Löve	Löve	Lua-based 2D game engine	Open source	love2d.org
Matter.js	Matter.js	2D JavaScript physics engine	Open source	brm.io/matter-js

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
melonJS	melonJS	HTML5 game engine	Open source	melonjs.org
Microsoft	Havok Physics	Physics engine	Available by request	havok.com/physics
Microsoft	Havok Destruction	Descructible asset creation	Available by request	havok.com/destruction
Microsoft	Havok AI	Game AI programming	Available by request	havok.com/ai
MonoGame	MonoGame	Cross platform, open source version of Microsoft XNA Framework	Open source	monogame.net
Netlify	babylon.js	JavaScript 3D game engine	Open source	babylonjs.com
Nintendo	Nintendo Switch SDK	Developer program, SDK	Free solution	developer.nintendo.com
Nintendo	Nintendo 3DS SDK	Developer program, SDK	Free solution	developer.nintendo.com
Oimo.js	Oimo.js	JavaScript physics engine	Open source	github.com/lo-th/Oimo.js
Open Dynamics Engine	Open Dynamics Engine	Physics engine	Open source	ode.org
OpenFL	OpenFL	Haxe-based, cross platform development platform	Open source	openfl.org
Panda2	Panda2	HTML5 game development platform	Open source	panda2.io
Phaser	Phaser	HTML5 browser and desktop game development platform	Open source	phaser.io
PlayCanvas	PlayCanvas	HTML5 and WebGL game engine	Open source	playcanvas.com
Scirra	Construct 3	2D HTML5 game development platform	Free tier available	scirra.com/construct2
ShiVa Engine	ShiVa Engine	3D engine and application development suite	Free tier available	shiva-engine.com
Simple DirectMedia Layer (SDL)	SDL	Development library for low-level access to hardware features	Open source	libsdl.org
Sony	Playstation Developer Program	Developer program, SDK	Free solution	playstation.com/en-us/develop

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Sparklin Labs	Superpowers	HTML5 development environment	Open source	superpowers-html5.com
Stencyl	Stencyl	Visual scripting platform	Free tier available	stencyl.com
Unity	Unity3D	Cross-platform game engine	Free tier available	unity3d.com
Valve Software	Source Engine	Cross-platform game engine	Free solution	developer.valvesoftware.com/wiki/Main_Page
voxeliq	voxeliq	Voxel-based game engine	Open source	github.com/bonesoul/voxeliq
Well Caffeinated	PhysicsJS	JavaScript physics engine	Open source	wellcaffeinated.net/PhysicsJS
Yoyo Games	GameMaker Studio	2D game development platform	Free to test	yoyogames.com/gamemaker

GRAPHICS

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
3D-Coat	3D-Coat	3D modeling	30 days	3dcoat.com
Adobe	Fuse	3D character modeler	Available by request	adobe.com/products/fuse.html
Amabilis Software	3D Crafter	3D modeling and animation	Free solution	amabilis.com/
Autodesk	3ds Max	3D animation, simulation, modeling, and rendering	30 days	autodesk.com/products/3ds-max/overview
Autodesk	Mudbox	Digital sculpting tool	30 days	autodesk.com/products/mudbox/overview
AwesomeBump	AwesomeBump	Texture generator	Open source	github.com/kmkolasinski/AwesomeBump
Blender	Blender	3D modeling and rendering	Open source	blender.org
Hme	Hme	Height map editor	Open source	sourceforge.net
Imagination Community	PowerVR Graphics SDK	Cross-platform framework for building rendering engines	Open source	community.imgtec.com/developers/powervr/graphics-sdk
Inkscape	Inkscape	Vector graphics design	Open source	inkscape.org/en

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Intel	FaceTracking	Real-time face tracking and animation	Open source	github.com/GameTechDev/FaceTracking
Intel	Graphics Performance Analyzer	Graphics performance monitoring	Free solution	software.intel.com/en-us/gpa
Intel	Intel Math Kernel Library	Numerical processing and physics engine	Free solution	software.intel.com/en-us/intel-mkl
Khronos Group	Vulkan API	Graphics API for GPUs	Open source	khronos.org/vulkan
MakeHuman	MakeHuman	3D character generator	Open source	makehuman.org
Maxon	Cinema 4D	Polygonal modeling tool	Available by request	maxon.net/en-us/products/cinema-4d/overview
NeoTextureEdit	NeoTextureEdit	Texture editor	Open source	neotextureedit.sourceforge.net
NewTek	LightWave	3D animation and design	30 days	lightwave3d.com
ngPlant	ngPlant	3D plant modeling software	Open source	ngplant.org
Overlap2D	Overlap2D	Game level and UI editor	Open source	overlap2d.com
Pixologic	ZBrush	Digital sculpting tool	N/A	pixologic.com/features
SideFX	Houdini	3D modeling, rendering, game dev workflow	Free tier available	sidefx.com
SpeedTree	SpeedTree Games	Vegetation modeling	Available by request	speedtree.com/video-game-development.php
Synfig	Synfig Studio	2D animation tool	Open source	synfig.org/cms
three.js	three.js	3D modeling with JavaScript	Open source	threejs.org
Tiled	Tiled	Map editor	Open source	mapeditor.org

AUDIO

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Ableton	Ableton Live	Digital audio workstation	30 days	ableton.com

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Ardour	Ardour	Digital audio workstation	Open source	ardour.org
Audacity	Audacity	Digital audio workstation	Open source	audacityteam.org
Audiokinetic	Wwise	Sound effects engine	Free tier available	audiokinetic.com/products/wwise
Audiokinetic	Wwise Soundseed	Sound generator for games	Available by request	audiokinetic.com/products/plug-ins/soundseed
Avid	ProTools	Digital audio workstation	30 days	avid.com/pro-tools
Beast	Beast	Music editor and synthesizer	Open source	beast.testbit.org
BeepBox.co	BeepBox	Web-based chiptune composition	Free solution	beepbox.co
Facebook	Facebook 360 Spatial Workstation	Spatial audio design for VR	Free solution	facebook360.fb.com/spatial-workstation
Firelight Technologies	FMOD	Sound effects engine	Sound effects engine	fmod.com
GenAudio	AstoundSound	Spatial audio engine	Available by request	astoundholdings.com
howler.js	howler.js	JavaScript audio library	Open source	howlerjs.com
LMMS	LMMS	Digital audio workstation	Open source	lmms.io
MilkyTracker	MilkyTracker	Audio engine	Open source	milkytracker.titandemo.org
Musagi	Musagi	Music editor and synthesizer	Open source	drpetter.se/project_musagi.html
OpenAL	OpenAL	3D game audio API	Open source	openal.org
RAD Game Tools	Miles Sound System	2D and 3D audio creation engine	Available by request	radgametools.com/miles.htm
Reaper	Reaper	Digital audio workstation	60 days	reaper.fm
rFXgen	rFXgen	Sound effects engine	Open source	raysan5.itch.io/rfxgen
sfxr	sfxr	Game audio library	Open source	drpetter.se/project_sfxr.html

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
SoLoud	SoLoud	Game audio engine	Open source	sol.gfxile.net/soloud
somethinelse	Papa Engine	Sound effects engine	Available by request	somethinelse.com/projects/gaming-for-gamers-audio-games
Steinberg	Cubase	Digital audio workstation	N/A	steinberg.net/en/products/cubase/start.html
MULTIPLAYER TOOLS				
AWS	AWS Mobile Services	mBaaS	Free tier available	aws.amazon.com/mobile/
brainCloud	brainCloud	BaaS	Free tier available	getbraincloud.com
CloudBoost	CloudBoost	mBaaS	Free tier available	cloudboost.io
Exit Games	Photon	Networking engine and multiplayer platform	Free solution	photonengine.com/en-US/Photon
GameSparks	GameSparks	mBaaS	Demo available by request	gamesparks.com
Google	Firebase	mBaaS	Free solution	firebase.google.com
Heroic Labs	Nakama	Real-time games server	Open source	heroiclabs.com
lance.gg	Lance	Node-based game server	Open source	lance.gg
NextPeer	NextPeer	Multiplayer social SDK	Open source	nextpeer.com
PlayerIO	PlayerIO	mBaaS	Available by request	playerio.com
Progress Software	Kinvey	mBaaS	Free tier available	kinvey.com/application-development-platform
ShepHertz	AppWarp	Real-time and turn-based multiplayer platform	Available by request	appwarp.shephertz.com
SmartFoxServer	SmartFoxServer	Multiplayer game development SDK	N/A	smartfoxserver.com
Socket.io	Socket.io	Bidirectional event-based communication	Open source	socket.io
Ubisoft	Massgate	mBaaS	Open source	github.com/ubisoftinc/massgate

GLOSSARY

3D MODEL

A representation or simulation of an object in a three-dimensional space created by software. These models are created with several small triangles called polygons.

3D MODELERS

Software used to create 3D models that can be manipulated in video games or animation.

APPLICATION PROGRAMMING INTERFACE (API)

A specification for how various applications can interact with a set of software components. Applications can include APIs for external use by other software.

ARTIFICIAL INTELLIGENCE

A machine's ability to make decisions and perform tasks that simulate human intelligence and behavior.

AUGMENTED REALITY (AR)

A game or view in which real-world locations also have digital information associated with them.

CERT

A certification received from a console manufacturer after a game is tested and approved for sale on their platform.

CONSOLE

Any piece of proprietary hardware whose primary function is to play games.

DESIGN DOCUMENT

A reference document created by game designers, developers, and artists to use as a guiding vision for how a game will be created, including visual style and gameplay elements.

DISTRIBUTION CHANNELS

Any channel through which a game can be sold or downloaded, such as Steam or a mobile app store.

EMULATOR

An application that duplicates the functionality of hardware or operating systems for testing purposes.

GAME DESIGN

The art of creating mechanics, creating enjoyable interactions among players, and ways to reward players when making games.

GAME DEVELOPMENT

The process of programming a video game in order for it to faithfully execute game design mechanics and the design document.

GAME ENGINE

A software development platform or framework specifically designed to create video games. They may provide a number tools including 3D modeling software, physics engines, sounds, a code editor, AI, networking, and animation.

GRAPHICAL USER INTERFACE (GUI)

The visually-based tools a player uses to interact with a game, including menus, icons, and status trackers. This does not include controls.

MOBILE BACKEND AS A SERVICE (MBAAS)

A service that connects mobile applications to cloud databases while also providing user management, push notifications, and social integrations.

PC

Shorthand for any desktop or laptop device that uses a Windows operating system.

PHYSICS ENGINE

A software platform to add physics systems or laws to a video game or animation to make objects move or collide in a way similar to real life.

PLAYER INPUT

Any action from a user that can interact with a game or environment.

POLYGON

A small triangle that, when combined with other polygons, can be used to create a 3D object.

POLYGON COUNT

The number of polygons in a 3D model that can be rendered per frame.

RELEASE CANDIDATE

The first build of a game that could conceivably be distributed, though this is not often the case due to the need for bug fixes.

TEXTURE MAPPING

The process of defining detail and color on a 3D model.

VIRTUAL REALITY (VR)

A platform where a user can interact with a totally digital and immersive 3D space.

WEBSOCKET

A communications protocol that uses a TCP connection to allow communication between two computers.

AnswerHub
Ask
#GDC18