Spelunky

Spelunky

Derek Yu

Boss Fight Books
Los Angeles, CA
bossfightbooks.com

ISBN 13: 978-1-940535-11-1
First Printing: 2016

Series Editor: Gabe Durham
Book Design by Ken Baumann
Page Design by Christopher Moyer

For Frances and Caroline

CONTENTS

TIMELINE

2002	June 21	*Eternal Daughter* released
2005	February 3	I take over TIGSource from Jordan Magnuson
2007	December 7	*Aquaria* released
2008	September 22	*Spelunky Classic* beta release
	December 21	*Spelunky Classic* first public release
2009	January 16	Jonathan Blow recommends *Spelunky* to Microsoft
	October 31	I begin working with Andy on *Spelunky* XBLA
	December 25	*Spelunky Classic* v1.1 released
2012	June 8	*Spelunky* XBLA submitted to final certification
	July 4	*Spelunky* released on XBLA
2013	August 8	*Spelunky* released on Steam
	August 27	*Spelunky* released on PS3 and PS Vita

	November 10	Bananasaurus Rex completes the solo eggplant run
2014	October 7	*Spelunky* released on PS4
	August 14	Work begins on *Spelunky* book

PART I:
SPELUNKY CLASSIC

Take this key.
It will open the entrance to the mines.
Follow me if you dare!

— Yang

Beginning of the Adventure

When I meet someone for the first time and the topic turns to *Spelunky*, I describe it as "a platform game, like *Super Mario Bros.,*" because whether or not you play video games, you've heard of Mario. Mario was the face of the art form during the 1980s and 1990s, when platformers dominated the gaming landscape. Unlike movie genres, which are distinguished by similarities in the themes or narratives, video game genres are generally defined by how the games are played. Platformers, for example, are defined by jumping onto platforms and over obstacles, just as shooters are defined by shooting things and puzzle games are defined by solving puzzles.

Playing *Spelunky* for the first time, you'll notice similarities to other platformers: You run, you jump, you hit enemies. The game starts you off in an abandoned mine, armed with a whip that you can use to dispatch snakes, spiders, bats, and the occasional caveman, as well as ropes and bombs to help you reach otherwise unavailable areas. Since it's your first time playing, you'll probably die quickly. Maybe you didn't know that the stone block would let loose an arrow when you stepped in front of it, or maybe you misjudged a long jump, sending you into a pit of sharp spikes. Or maybe you lingered too long in the level and a creepy ghost floated onscreen and vaporized you. Or one of your bombs blew

up in your face. Oh, well! In *Spelunky* you only have one life, so if you run out of hearts you'll be returned to the lobby room.

It's when you start up your second game that you'll notice *Spelunky* is different from Mario. In *Super Mario Bros.*, you'll always start a new game by taking that familiar stroll to the right, where you'll pass by the same four question blocks, the same goomba, and the same short pipe. The levels are unchanging. Each time you play *Spelunky*, however, you'll get a completely different set of sixteen levels. In one run, you may encounter a large pit filled with snakes in the third level, and in the next run you may encounter a sacrificial altar and a bomb shop instead. Occasionally, levels will be dark, forcing you to maneuver carefully by the dim light of flares and glowing scarabs. While each element of the game behaves the same (a snake, for example, will always move back and forth in the same pattern, and an arrow will always do the same amount of damage), the way those elements are arranged is randomly generated at run-time. This randomization is what makes *Spelunky* stand out among platformers, and it's why my full summary of the game is that it's "a platform game, like *Super Mario Bros.*, but with randomly-generated levels."

Sometimes I'll describe *Spelunky* as a platformer that's inspired by "roguelikes," although roguelikes are much less well-known. The genre takes its name from a top-down, turn-based fantasy game called *Rogue*, which was created in 1980 by Michael Toy and Glenn Wichman.

Whereas playing a Super Mario or Sonic the Hedgehog game in their heyday was a lesson in designing elegant levels and silky-smooth controls, roguelikes have prided themselves on their randomly-generated dungeons and mechanical complexity. The original *Rogue* used the ASCII character set for graphics (where an "@" might represent the player and a "g" could be a goblin) and this tradition was carried by roguelikes far into modern times, keeping it a niche genre. Even the popularity of Blizzard's Diablo series, which was influenced by roguelikes, didn't do much to raise their profile, since the influence was not widely known. When I started working on *Spelunky*, roguelikes were only just starting to break into the mainstream consciousness.

It probably doesn't help that what a roguelike is has always been up for debate. At the 2008 Roguelike Developers Conference, a group of roguelike developers attempted to clarify the definition of the genre by creating a list of high and low value factors called the "Berlin Interpretation." The Berlin Interpretation's high value factors define a roguelike as a turn-based, grid-based, dungeon crawl that features randomized levels and permanent death. The less important low value factors include ASCII graphics, one player character as opposed to a party system, and monsters that share rules and behaviors with players.

When I was working on *Spelunky*, I focused on just three attributes of roguelikes that to me held the essence of the genre:

1. Randomized level generation.
2. Permanent death (also known as "permadeath"), whereby the player has one life and cannot reload their game to take back mistakes.
3. A ruleset for physical interactions that is shared by the player, non-player characters (NPCs), and items.

The third attribute is my version of "monsters share rules and behaviors with players" and it extends not only to monsters but to items as well. In many games, monsters and items are coded separately and handled separately. You might be able to pick up an item and kick a monster, but you couldn't kick an item and pick up a monster. In a roguelike, you should be able to pick up and kick items *and* monsters, with results that are based on consistent rules. This suggests that the player, the monsters, and the items are treated more like variations on a single type of object than three completely different objects.

Over the years, attempts have been made to add audio and attractive graphics to roguelikes, and to improve their intimidating user interfaces. For example, *Mystery Dungeon: Shiren the Wanderer* is a Japanese roguelike for the Super Famicom and Nintendo DS that makes traditional roguelike game mechanics more accessible with animated sprites and simplified controls. The latest versions of the canon roguelikes (*ADOM*, *Angband*, *Crawl*, and *NetHack*) now support tile

graphics and mouse control as well. But it was a 2006 roguelike/simulation hybrid called *Dwarf Fortress* that brought the genre into the light, and it did so not by being more accessible, but by being even more complex. Imagine a game that simulates a fantasy world many times larger than the most advanced big budget video game, but rendered completely in characters you'd find in a word processor.

Dwarf Fortress has two modes: "Adventurer Mode," in which you control a single hero, and "Fortress Mode," which tasks you with keeping a community of dwarves alive in a mountain home of your design. This home is just a small part of a world that is procedurally-generated with remarkable detail going back hundreds of years, to the point where you can trace a minor character's lineage back many generations. I read about one world where the history logs revealed that a particular NPC's part-goblin blood was the result of one of his ancestors being kidnapped by goblins during a raid. And one of the cool things about the game is that both Adventurer Mode and Fortress Mode can be played in the same world, so a player could build a fortress in Fortress Mode and then visit that same fortress as an adventurer later (or the ruins of that same fortress, once things inevitably go awry).

Dwarf Fortress made for interesting news because the sheer complexity of what the simulation could carry out seemed nearly matched by its inaccessibility. The game, which is still in constant development, has an air of

mystery about it, like it's being built by some advanced alien race.[1] Not long after the game's first release, the term "roguelike" began to pop up in mainstream news outlets and on social media. My personal history with the genre extended back to my childhood, so it wasn't new to me, but I also got caught up in the excitement for this previously closed ecosystem being discovered by the outside world.

Puzzle Pieces

The creative mind is like a big pile of jigsaw puzzle pieces. Some pieces were made by other people—inspirational words of advice, an intriguing screenshot from a game you've never heard of, a haunting melody—and some are gained through life experiences. Some pieces are already connected, either because they came that way or because while you were walking down the street or taking a shower they somehow found each other. Sometimes a single piece is missing, and once that piece is uncovered, two other pieces from different ends of the pile can finally be connected.

It's important to accumulate many, many jigsaw pieces, since the more you have available, the more

1 Actually, it's being made entirely by two brothers, Tarn and Zach Adams.

things you can build. But eventually you have to sit down and start sifting through the pieces to put them all together. This is the "work" part of creation. It can often be frustrating, like when two pieces seem like they should fit but don't. Sometimes you know that there's a perfect piece around but you're not sure where it is. Is it even in your head? But like any challenging task with a noble purpose, the frustration also gives way to joy, elation, and ultimately satisfaction when you've finished a big part of a new puzzle.

Before *Spelunky*, my friend Alec Holowka and I put together a particularly challenging puzzle called *Aquaria*. It was my first commercial game, and after two years of hard work we released it in 2007. Alec was the programmer and musician, and I was the graphic artist and main level designer. In the initial prototype, which Alec created before we met, you lead a mermaid around a small, underwater library. By the time we released the game, that mermaid became Naija, a lonely mergirl who was exploring a sprawling fantasy world where she could sing, fight, cook, and fall in love.

Aquaria was a success. It wasn't a runaway one—these were still the early days of the modern indie gaming scene, right before console exclusives and digital distribution created a gold rush. But it was more of a success than we had dared to hope for. Not only did *Aquaria* sell well enough that we could continue making games full time, but the positive feedback we received from players and critics also validated us as artists and helped

put to rest our doubts about whether we were making something worthwhile.

Earlier that year, *Aquaria* had won the grand prize at the Independent Games Festival (IGF), a showcase for smaller-budget, independent video games the same way that Sundance is a showcase for independent films. My expectation of the IGF Awards was that we would walk into a small room and have ribbons pinned to our chests, so it came as a quite a shock that an actual red carpet lead us to the place where the awards would take place: not a small room at all, but an immense conference hall inside of San Francisco's Moscone Convention Center. And while it might not have quite reached the level of Oscars in terms of glitz and glamour, to a couple of game developers who spent most of each day in their bedrooms, it might as well have. I was awed at the lights, the big stage and the cameras aimed at it, and all of the seats waiting to be filled. Up front, right beneath the stage, were tables and chairs for nominees and special guests, many of whom I'd read about and admired for years, including Shigeru Miyamoto, the creator of *Donkey Kong*, *The Legend of Zelda*, and *Super Mario Bros.* I'll never forget the feeling of walking up to that roped-off area with Alec and my family and being let in, entering a space that was also reserved for legends.

Of the four awards *Aquaria* was nominated for, we had already lost three by the time the Grand Prize nominees were announced, so when it was revealed

that *Aquaria* had won, our table erupted in shock and delight as Alec and I jumped up and hugged each other hard.

That kind of emotional swing was not uncommon for *Aquaria*'s development. In the weeks leading up to our IGF submission, the direction of the game took a sharp turn. Towns full of characters, cutscenes, and dialogue were cut in favor of focusing on Naija herself. During that period, actress Jenna Sharpe brought Naija's previously text-only voice to life—I remember being woken from a nap to hear Alec's triumphant "Fuck yeah!" after he listened to her reading for the first time. But we were also forced to kick out a friend of Alec's whom we had hired to manage the production, due to creative differences. Those two years felt like a continuous expansion and contraction, both with our collaborators and with the game itself.

At the center of it all, the one true constant was Alec and me. My wife Frances and I still joke about how I spent more time with Alec in those two years than with anyone else. How could you not become close? At the same time, we mostly communicated through Google Talk because I lived in San Francisco and he lived in Vancouver. That physical separation, I think, made it harder to really get to know each other. When we started working on *Aquaria* full time, we had only recently met online, Alec having found me through a small game project I was working on. And while we shared plenty of happy, silly, and triumphant moments in those two

years, we also clashed over our differences in personality and work style. I've always preferred to chip away on my own part of a project in relative solitude and meet with teammates for discussion when our parts overlap, but that often made Alec, who preferred more steady communication, feel like we weren't acting as a team at all. And although I consider myself to be a hard worker, Alec's capacity for game-making seemed limitless. It's a trait that I still respect, but at the time it was stressful trying to keep up.

I'm proud of how *Aquaria* reflects the ups and downs of our partnership. But for the same reasons that I'm proud of *Aquaria*, I also couldn't make another *Aquaria*. When the time came to start on our second game, we agreed to make something small, but each of our ideas quickly ballooned into something bigger. For me, it was too soon to start another intense, two-year project—although Alec was eager to begin working again, I realized that I needed to slow down permanently. So as difficult as it was to walk away from a successful team and a talented friend, I felt like we would both suffer if I didn't.

So in 2008, I returned to my pile of puzzle pieces, alone again. Somewhere off to the side sat *Aquaria*, recently completed. In front of me, pieces from two seemingly different types of puzzles, platform games and roguelikes, were scattered. But at the time, I didn't know what I was going to build next, just that I wanted

to build something. So I reached for the pieces that were closest to me and started trying to fit them together.

Game Maker

After Alec and I went our separate ways, I decided to look into Game Maker, which was at version 8.1 at the time. Created by Dutch computer scientist Mark Overmars, Game Maker is an all-in-one 2D game-making system that includes a variety of editors and drag-and-drop interfaces. All of the basic functionality required of a game—like input, rendering, and collision-detection—is handled by the program and hidden away from the user, so it's easy to get a game up and running quickly. You can even create low-resolution pixel art and animate it right in Game Maker via the Sprite Editor.

It would have been difficult to make a large game like *Aquaria* in Game Maker 8.1, since it lacked the power and functionality of Alec's custom-made engine.[2] There was no way, for example, that Game Maker could handle the sheer number of objects that existed in a single area of *Aquaria*, especially at the game's default resolution. On the other hand, Game Maker's simplicity made it very popular amongst hobbyist developers who

2 The successor to Game Maker, GameMaker: Studio, is much more powerful than the version I used to make *Spelunky*.

wanted to create something smaller, with graphics and music similar to video games of the 80s and 90s. I loved the feel of these hobbyist titles, bold and bite-sized, each one offering a few hours of pure entertainment. The personality of each game's creator shone brilliantly through unique and decidedly uncommercial art and music choices. Even the names of the games—*Clean Asia!*, *Punishment: the Punishing, The Jeluvian Project*—and the handles of their creators—cactus, messhof, Pondwater, respectively—were refreshingly strange.

The games reminded me of something the author Haruki Murakami says in the intro to the English edition of *Blind Willow, Sleeping Woman*: "I find writing novels a challenge, writing short stories a joy. If writing novels is like planting a forest, then writing short stories is more like planting a garden. The two processes complement each other, creating a complete landscape that I treasure." In video games, we don't have corresponding terms for novels and short stories—we simply have large games and small games. These Game Maker games were Murakami's gardens: vibrant, intimate, and full of charm. Having just planted a large forest with Alec, I was eager to find my own little patch of fertile soil.

Game Maker also reminded me of Klik & Play, the first game creation tool that I used. When it was released in 1994, I was twelve and wanted nothing more than a way to bring my game ideas to life, so a magazine ad for Klik & Play sent me running to my parents to beg them for it. When it arrived, I marveled at the box, on which

a dragon, a spaceman, a stealth bomber, a tank, and a court jester burst out of a boxy gray computer mouse. Beneath the logo was a bold claim: "The Revolutionary Instant Game Creator." I was thrilled!

It wasn't long before I came up with a label, "Blackeye Software," and released my first game onto the internet: a top-down, two-player deathmatch called *Trigger Happy* where you had to take out your opponent using machine guns, flamethrowers, and bazookas. On the title screen, a red-bereted military officer watched over a group of soldiers doing exercises while you and a friend pondered which of the three deadly arenas you wanted to do battle in. It was a simple game, but I was flush with nervous excitement when I uploaded *Trigger Happy* to AOL with my parents' 2400 baud modem. Not only was this my first game, but this was the first time I put something I had made online.

Within a few days I began receiving emails praising the game and asking for more. I was ecstatic. Although the email writers didn't know it, they had gotten a twelve-year-old hooked on releasing games, pushing him toward his dream. From there, I teamed up with my friend Jon Perry to continue making Blackeye Software games using Klik & Play, and eventually enrolled in the electrical engineering and computer science major at the University of California, Berkeley. I thought a computer science degree would give me the skills I needed to enter the game industry. Drawing, my parents and I

reasoned, was something I had always loved doing and could therefore keep learning outside of the classroom.

Until recently, programming was the crucial gatekeeper to making games. Early in the history of video games, the lead programmer was typically also the lead writer, artist, and designer. If you could program software, you could be a game creator. You may not have necessarily been able to craft a good game, but at least you could try. If you weren't a programmer, you needed to find one. The advent of game-making tools like Klik & Play and Game Maker changed the landscape of game development drastically, putting less of a premium on programming and allowing all kinds of new people to make games.

I think the people these programs benefited most are people like me, generalists for whom programming is not a primary passion but a means to an end. What I lack, perhaps, is an enjoyment of programming for the sake of programming, or "programming in the abstract sense," as John Carmack, the programmer of *Doom* and *Quake*, once described it. While the problem-solving and optimization involved in programming is sometimes a welcome change from making artwork, most of the time I would prefer code to just work and get out of my way. Were it not for video games, I would probably have very little interest in learning how to program at all. This was evidenced by the mediocre grades I got in college, save for a single

"A" in a computer graphics course where I submitted a video game as the final project.

I've tried programming a game engine from scratch before, surrounding myself with books like *Tricks of the Game-Programming Gurus* in an effort to "make games the right way," but when days of work yield as much as I could make in Game Maker in a few minutes, it's hard to stay motivated. At the beginning of each semester at Berkeley I had the same sort of naive gumption, buying pristine notebooks and attentively jotting down everything the professors said, only to succumb to ennui a week later, my notes devolving into irreverent doodles. Post-school, however, I accepted that I wasn't cut out for academia and programming theory. I no more wanted to program my own game engine than I wanted to fashion my own paintbrushes. This important realization meant I could stop wasting my time trying to be something I wasn't. Instead of being embarrassed about not being a "real programmer" using "real programming languages," I vowed to make games whichever way felt good *to me*.

The first post-*Aquaria* games I built in Game Maker were simple platformers. For indie and hobbyist developers, limiting the number of new things you have to learn is crucial to finishing your projects and since I was already learning a new game-making tool, I decided to stick to a genre I was familiar with. Given how popular the platformer genre is, I also knew that I'd be able to find plenty of user-made scripts to set up the basic input and physics required to get my game running quickly. That way, I

could focus on what I liked working on most: creating sprites, designing levels, and programming game logic.

Aside from the wealth of resources for making them, there are other, less obvious reasons why 2D platformers are a practical choice for small developers. For one thing, the side-view perspective means that the game's artist only has to draw every object in one direction, after which the game engine can flip the sprite to create the other direction. It's also a very broad genre, for which the only requirements for inclusion are that your character can run and jump, making it the perfect canvas on which developers can build their own themes and mechanics. In one of my early Game Maker platformers, you controlled a cloud and your primary action was soaking up water that you could unleash as rain, hail, or thunder. In another, you were a questing king. In another, you were a robot that could build a variety of guns out of spare parts.

But ultimately, none of these ideas felt like they broke enough new ground. When working on games, I've found that the projects I have the highest chance of finishing fall at the intersection of three categories: games I want to make, games I'm good at making, and games I wish I had made. "Games I want to make" and "games I wish I had made" sound similar, but in reality, not every game that is fun to work on is necessarily a game that I'd be proud of releasing, and vice versa. My platformer prototypes were enjoyable enough to work on, but they only added small twists on conventional gameplay, a fact that

was hurting my motivation to keep working on them. If I wasn't going to make another big game, I wanted my small game to be something special.

Roguelike

I've only played the original *Rogue* once or twice. My first real roguelike experience was with one of its spiritual sequels called *Hack*, a genuine treasure hidden within the many 5.25-inch floppy disks that my parents acquired in graduate school before I was born. The disks themselves were treasures, too—poorly labeled with no apparent rhyme or reason to their organization, they were arcane to my young mind. Looking for computer games amidst the various school documents was like exploring a dungeon or brushing away the dirt from a fossil, and I rifled through the disks many times, hoping to find something I missed.

It helped that the filenames gave little to no indication as to what the games were about. "HACK.EXE" conjures up images of hacking software rather than a roguelike.[3] There was another game, a text adventure

3 The name *Hack* actually *is* inspired by software hacking of a sort: *Hack*'s creator, Jay Fenlason, described his first release of the game as "a quick hack," meaning he threw it together without much care. It's also a reference to "hack and slash," a popular term used to describe combat-oriented fantasy games.

called *Madame Fifi's Whorehouse*, which hid behind the mysterious but rather innocent-sounding filename "FIFI.EXE." Imagine my delight at making that discovery, as well as my gradual disappointment once the titular whorehouse turned up nothing more risqué than a PORNO item that couldn't even be opened.

What the games and the disks elicited was a real feeling of mystery and excitement that is hard to come by in the era of online trailers, reviews, maps, and walkthroughs. While some of the games, like *Hack*, were genuine classics, and others, like *Madame Fifi's Whorehouse*, were little more than novelties, they all had the air of a real adventure around them, magnified by my youthful exuberance.

Of all the adventures on those disks, *Hack* was the best. The last stable release of *Hack* was in July 1985, a few months before Nintendo released *Super Mario Bros.* on the Famicom, making the games close contemporaries. And yet the two titles could not be more dissimilar, each making the other seem primitive in different ways. Visually, *Hack* seems like a relic, with its monochrome, ASCII-based graphics that look like the random garbage an old Nintendo game might spit onto the screen if the cartridge was breaking down. But playing the two games, it becomes obvious which one has the larger "possibility space," a term used in video games to describe the number of different things that can happen within the constraints of a game's rules. In *Hack*, a dizzying array of commands is available to

the player—nearly the entire alphanumeric keyboard, including lowercase and capital letters, is used for moving around and interacting with the world. The interactions are relatively complex, too. With even the simple act of eating food, *Hack* goes above and beyond the detail of most games, in which eating means walking over a hamburger or cherry and gaining points for it. In *Hack*, the player has five levels of hunger, from "fainting" all the way to "satiated," and you have to watch what you eat, particularly since a good majority of your nutrition comes from the corpses of monsters, some poisonous, that you slay along the way. On at least a few occasions, I forced my player character in *Hack* to eat food past the point of satiation and actually choked to death on the meal. Just because I could!

NetHack is the successor to *Hack* and adds new content, as well as a charming graphical tileset. Once, while playing *NetHack*, I came across a troll in a spike pit. Trolls are not friendly creatures, and although they're easy to kill, they have a nasty habit of reviving after death. In this situation, it would make sense to leave the troll where he was—stuck—but for whatever reason, I felt compelled to pull him out of the trap with the #untrap command (one of the "extended commands" that is not mapped to its own key on the keyboard). After doing so, a little heart icon appeared next to the troll's sprite to let me know that he had been tamed, grateful for the rescue. Stories like these abound in *NetHack*. There is, for example, an entire spoiler page that details all of

the ways that a cockatrice can turn you to stone. Just touching the corpse of this deadly creature petrifies you, so if you want to pick one up safely you need to be wearing gloves. Why would you want to hold one, though? To use it as a weapon, of course! But even with gloves on, a dead cockatrice is worrisome, since you could easily fall on top of it while descending the stairs or because you stumbled while wearing cursed boots. The vastness of the game's possibility space and the fact that each possibility is handled elegantly, without bugs or inappropriate messages, has led *NetHack* players to create a mantra that's now applied to other games, as well: "The DevTeam Thinks of Everything."

Years after *Spelunky*'s initial release, I was asked at a video game convention which roguelikes I had beaten, to which I replied that the only one I could remember beating was *Mystery Dungeon: Shiren the Wanderer*. The questioner seemed mildly disappointed that I'd only beaten one roguelike before, and that it was a rather easy roguelike, at that. The honest reason why I've never beaten *NetHack*, though, is simply because I don't have the patience for it. To beat *NetHack*, also known as "ascending," you have to travel 45 to 53 floors deep to acquire the Amulet of Yendor, then climb back up and past the first level, through the first four Elemental Planes and finally to the Astral Plane, where you must sacrifice the amulet to the deity of your alignment. In the "Maniac's Ascension Guide," a popular strategy guide that was first posted to the game's USENET

newsgroup in 2002, the author claims that a "fast ascension" takes around 40,000 turns. "There is no time limit," he explains. "Take as long as you need to do the job properly."

Ascending not only requires extensive knowledge about the game, but also a great deal of tedium. High-level play involves tactics like training a pet to steal items for you or "pudding farming," an act where the player splits a monster called a black pudding in half over and over again to collect its corpses. The *NetHack* wiki describes the act as its own punishment because of how dull it is. Another high-level technique is called "polypiling" and involves creating large piles of items to zap with a wand of polymorph, transforming the items into something different (but of the same general category). Usually the transformed items are just useless junk, but there is a small chance that you will get something that is good enough to keep for your "ascension kit," a collection of equipment that is more or less required to ascend—a problem in and of itself, since the existence of such a kit reduces the number of interesting choices the player can make if they want to win the game.

The person who asked me which roguelikes I had beaten had rightly assumed that *Spelunky* was inspired by roguelikes because I'm fascinated with them, but wrongly assumed that I loved everything about them. In truth, there's a lot about the games that I'd rather read about in wikis and spoilers than experience firsthand, in

the same way that someone studying medieval history might not actually want to ride into battle in a full suit of armor. The parts I did love were the randomly-generated levels and the tension created each time you encountered a new monster or item. If you died, then you weren't just losing your character, you were losing that entire world that was generated for you, since in most roguelikes you can't reload a save file to undo your mistakes. This permadeath factor gives death more meaning, which, in turn, gives more meaning to what you did while you and your character were alive.

Unimpressed with my platformer prototypes, I turned to roguelikes. But after *Aquaria*'s complex development I was still attracted to simplicity and minimalism, so I decided my roguelike would be very contained. The player would only be allowed to carry nine items, regardless of their weight, and the actions they could perform on the items would be limited to "use" and "throw." Another constraint I placed on the game was that each floor of the game's dungeon had to take place on a single grid of 15x15 tiles. Compare that to *NetHack*'s 24x80 tile levels, for example. To generate a dungeon in such a small space, I had to forgo the twisting passages that lead you from one room to the other. Instead, the nine rooms would be crammed together like tiny cubicles in an office space.

This seemed reasonable—even though the larger dungeons of *NetHack* allow for more variety, not much of the time spent traversing the long hallways between

rooms is interesting. There's a reason why *NetHack* includes a special key that lets you "fast move" from one end of a hallway to another, stopping in the middle only if something crosses your path. Whereas the hallways are generally just hallways, a room in *NetHack* can be many things. At the most basic, they're small chess-like boards that contain monsters, treasures, and traps, but they can also be shops, monster zoos, or throne rooms. Sometimes they're pitch black and you have to bump your way through.

It wasn't long before I stopped working on my roguelike. By the time I had made a couple of generic dungeons populated with enemies, traps, and shops, I was running into the same problem I ran into with my previous prototypes—distilling roguelikes into something simpler was a fun exercise, but it didn't add anything new to the genre. So once again, my mind began to drift away, from this little puzzle I had in front of me to the rest of my puzzle pieces. When I started, there was only a large, disorganized pile strewn in front of me, but now there were some smaller puzzles partially completed.

Platformers and roguelikes, platformers and roguelikes.

I asked myself, what is it that I like about platformers? I'm a more action-oriented game player and I like how easy they are to pick up and play. I like the tension behind each jump.

What didn't I like about platformers? I didn't like the repetitiveness of playing the same levels over and

over again, and the reliance on memorizing level layouts to succeed.

What did I like about roguelikes? I liked the variety that the randomly-generated levels offer and how meaningful death is in them.

What didn't I like about roguelikes? I didn't like the repetitiveness of certain tasks that were required to succeed, like polypiling, and the reliance on memorizing commands.

Platformers and roguelikes, platformers and roguelikes…

Then it clicked!

EXPLORER.GMK

The earliest build I kept of *Spelunky* is from June 29, 2008 and has the tantalizingly nondescript filename of "EXPLORER.GMK." Opening it up now in Game Maker, I notice that the protagonist, Spelunky Guy, looks slightly different from his current form—the brim of his trademark "fedora with a miner's lamp attached to it" is a pixel shorter on either side, making it look more like typical caver's helmet, and the two pixels of dark brown hair that delineates his ear from the rest of his head is missing, along with a pixel of shading on one of his hands. On a small sprite that is only 16x16 pixels wide, these small changes are jarring. I also see remnants of the older platformer prototypes still in the project,

just like relics of ancient civilizations. There's a slime, goblin, brigand, and gargoyle from one prototype and a missile sprite, robotic spider enemy, and empty room (named "rWestTower2") from another.

Otherwise, this build of *Spelunky*, though it only features a single level with bats, spikes, and arrow traps, looks and feels remarkably close to the final release, a reminder of how quickly the core design came together. Nearly all of the key elements that define *Spelunky* are present in EXPLORER.GMK. First and foremost are the randomized levels, the big hook of the game. Secondly, even though your basic whip attack is not available, you can pull out a bomb and blow up the walls, creating holes that can send Spelunky Guy falling off the bottom of the level and into the infinite space below.

Destroying the environment is a fairly novel thing to do in video games. In most games, aside from specific set pieces that are meant to be destroyed (for example, a wall with an obvious crack in it), the backgrounds, even windows and wooden fences, are completely impervious. The reason is not only that invincible scenery is an easy way to corral a player along a path, but also that it's a time-consuming and exacting process to make everything destructible. Destroying a piece of the world requires it to break convincingly, and it also requires that there is something behind it for the player to interact with. That's one of the reasons why destructible terrain was more common in older titles than in modern,

large-scale ones—as the complexity and realism of game worlds increased, so did the difficulty in making them break apart.

If you happened to live in Japan in 1983 and own a Sharp X1 computer, you might have played *Kagirinaki Tatakai*, an obscure, side-scrolling action game where you control a man equipped with a jetpack, lasers, rockets, and grenades. The most interesting aspect of the game was that almost all of its levels were fully destructible. And while I never played *Kagirinaki Tatakai* myself, an article about the game nonetheless helped inspire the inclusion of destructible terrain in *Spelunky*.

In the article, written for the website Hardcore Gaming 101, John Szczepaniak says this:

> Adding to the realism is the fact that you can destroy the environment in real-time. Any part of it. All of it. No, seriously. See that floor? If you don't feel like navigating a maze of enemies, just bomb your way through the floor and avoid them. This adds a HUGE tactical layer to the game, since you can bomb holes in the floor and let enemies pop through one at a time to take them out, or avoid them altogether. But since your grenades are finite, and essential to passing later sections, you need to use them carefully.

That sounds pretty cool, doesn't it? Destructible terrain not only gives the player more agency, making the game more immersive, but it also makes the level generation easier, since the developer can safely generate areas that are walled off. The level generation system I ended up using for *Spelunky* depends on your inability to access the entire level without destroying the walls.

Another key element that's present in this early build is *Spelunky*'s "theme," which, in game design terms, refers to the setting, story, and imagery that we wrap around the abstract rules and systems. The name *Spelunky* is a reference to caving, also known as "spelunking." Originally, it wasn't intended to be the game's final name—I just needed a more interesting filename than "explorer" for a preview screenshot I posted online. It was another indie developer, Phil Fish, who made me consider it again when he replied, "*Spelunky*, huh? I like it." Looking back, it's uncertain whether he meant that he liked the name or the screenshot, but I took it as the former.

Thematically, *Spelunky* comes from a long lineage of stories about rugged heroes hacking their way through thick jungles, exploring ancient ruins, and ultimately defiling the resting places of long-dead native peoples for treasure and glory. The most well-known of these adventurers is Indiana Jones, a character created by George Lucas and Steven Spielberg for a series of four major movies. The first three movies, which came out in the 80s, left a big impression on me, particularly

Indiana Jones and the Temple of Doom, with its terrifying traps and gruesome scenes of human sacrifice and chilled monkey brains.

Indiana Jones had a huge impact on video gaming, too. In the same way that *Alien* influenced science fiction video games like *Contra* and *Metroid*, Indiana Jones has inspired digital treasure hunters like Pitfall Harry, Rick Dangerous, Lara Croft, and Nathan Drake. This isn't surprising when you consider the popularity of Indiana Jones, the influence of movies on video games in general, and the fact that exploring dangerous ruins for fame and fortune has always been a popular theme for video games.

But "Indy" himself was a collection of homages, taking inspiration from James Bond, as well as the movie serials and pulp novels of Lucas and Spielberg's youth. His look—the fedora, brown jacket, khakis, and shoulder bag—are said to be directly influenced by Harry Steele, played by Charlton Heston in the 1954 adventure serial *Secret of the Incas*, which was an admitted influence on the first Indiana Jones film, *Raiders of the Lost Ark*. And his trusty bullwhip? From Zorro, who carried one on his belt and, in the serial *Zorro Rides Again*, used it to whip a gun out of someone's hand and swing out of a window. And speaking of *Raiders of the Lost Ark*, that movie was also inspired by, of all things, the adventuring of the cartoon characters Donald Duck and Scrooge McDuck, of whom Lucas was a huge fan growing up. The iconic boulder trap that Indy barely

escapes at the beginning of *Lost Ark* is very similar to the boulder trap featured in the 1954 Uncle Scrooge comic *The Seven Cities of Cibola*, penned by Carl Barks. The similarity goes down to the way the trap is triggered—by pulling an idol off of its pedestal.

None of this is to say that Lucas and Spielberg are unoriginal, just that they are part of a long tradition of adventure tales of which *Spelunky* is also a member. And what they gave back in Indiana Jones is much more than what they borrowed. That's what makes it Indiana Jones and not a clone of Harry Steele or a Zorro rip-off. Even the borrowing, if it's done honestly and with genuine respect and love, is a good thing—a way for new audiences to connect with old serials and comics they probably wouldn't have heard of otherwise. In a way, producing, experiencing, and interacting with art is like participating in a long-running conversation that artists have been having and will keep having in the future.

Themes also act as a sort of lubricant to understanding the rules of a game, bringing abstract concepts to life by tying them to concrete scenarios that humans already understand. Perhaps the best example of this is not a video game, but a board game: chess. Without a doubt, the rules of chess are beautiful on their own, and it's the rules that make chess such an influential and long-lasting game. But consider if the pawns were not pawns, the knights not knights, the kings and queens not kings and queens, etc., but just abstract shapes of varying sizes. If the military theme of the game was

removed entirely, chess would still be an elegant game, but not nearly as compelling. It's much more exciting to have a knight on a horse leap over a row of pawns to make an attempt on a king's life than to move a shape from one part of a grid to another. The personification of the chess pieces also makes the rules easier to learn—thinking of the knight piece as a knight on horseback helps me associate it with its leaping movement.

With *Spelunky*, the caving theme seemed appropriate for a number of reasons. For one thing, I had already decided that the player would be heading downwards toward the exits, since it's easier to let the player fall than to generate a path they can climb up. Caves, dungeons, and other natural or ruined environments also lend themselves well to video games because they're easy to mold into levels of almost any shape without looking out of place. This is even more valuable when you're generating levels algorithmically and can't anticipate what they'll look like. While it's certainly possible to set a game in, say, a pristine office building, the rigidity and repetitiveness of the layout gets boring quickly and places restrictions on how a character can move. Finally, I wanted to give the player special equipment so they could create their own paths, and spelunking tools like rope, bombs, and pickaxes were a natural fit for that.

So long as your theme is consistent with your rules, it doesn't matter whether it makes perfect sense in real life. On the surface, a story about an Italian-American plumber saving a Mushroom Kingdom from turtles

seems surreal and disconnected from reality. But after just a few minutes of playing *Super Mario Bros.*, we start accepting that eating mushrooms makes us grow larger, flowers let us throw fireballs, and stomping on koopas will let us kick their empty shells across the floor. Given how quickly human beings adapt to new situations, I could have easily set *Spelunky* in a world where you were blowing up cheesecake floors with exploding penguins, but since there was already so much for the player to figure out and so much for me to design, it made sense to start from a setting that both the audience and I recognized. That was the same reasoning I used when I started making platformers with Game Maker: Use the familiar as solid footing to learn something new.

Spelunky was both my easiest development and also most popular project to date, and I believe it's because the key elements—the theme, the random level generation, and the destructible terrain—work together without friction. Nothing was compromised to make something else fit and each part only boosted the signal of the other parts.

I'm reminded of Shigeru Miyamoto's definition of a good idea: "A good idea is something that does not solve just one single problem, but rather can solve multiple problems at once." Once those good ideas were in place, it felt like it was only a matter of plugging in the obvious holes and adding more and more content. New ideas either made sense, fitting into the existing schema with only a little bit of finagling, or they were quickly

rejected. In other words, *Spelunky* began to design itself from an early stage.

Randomized Levels

When I set out to create *Spelunky*'s level generation system, I used my roguelike prototype's level generation system as a basis. In that game, each level was built out of a simple grid of rooms. This simplifies the generation tremendously, because from the outset you already know where each room is—you don't need to worry about whether a room is going to overlap another room or how to connect them. Each level in *Spelunky* is composed of a 4x4 grid of rooms that are each made up of 10x8 tiles. Furthermore, *Spelunky* has four different areas: the Mines, the Jungle, the Ice Caves, and the Temple. Each area is broken up into four levels, for a total of sixteen levels in the game.

Moving forward, I had a few rules that I wanted to make sure my level generation followed:

1. Levels start at an entrance door at the top of the level and end at an exit door at the bottom.
2. There must be a way to get from the entrance to the exit without using bombs, rope, or any special items. (Anything not on this path can be walled off or otherwise inaccessible.)

3. The level generator must do its best not to create any places where the player can get stuck and have to use items to get out. (If such a place is created after level generation through terrain destruction, that's fine.)
4. Try to make the edges of the rooms as hard to notice as possible. (But don't worry too much about it.)

The game's first priority is to make sure that there is a path from the entrance to the exit that is traversable without the use of bombs, rope, or other special equipment. This makes winning less luck-dependent, reduces the chances of getting stuck, and lets players save bombs and rope for reasons other than getting past badly-generated levels. A script called "scrLevelGen" begins by picking one of the four rooms in the top row of the grid and marks it as an "entrance." From there, the algorithm generates a path from the entrance to the exit by randomly moving left and right on the grid and dropping down either randomly or when it hits the edge. Rooms that the algorithm never walks over are assigned a number "0," and rooms it walks over are assigned a number greater than "0." This lets the game know whether a room is on the path and whether it's a room with openings on the left and right sides ("1"), a room that has an additional opening on the bottom ("2"), or a room that has openings to the left, right, and top ("3"). In my head, I think of these as normal rooms, drops, and landing rooms,

respectively. In the case where a "2" is on top of another "2," the bottom room will also have an opening on the top. On the bottom row the algorithm stops and marks the room it stops on as the "exit."

```
[0][X][1][2]
[0][0][2][3]
[2][1][3][0]
[3][1][X][0]
```

A sample level after the scrLevelGen has walked through it, creating a path.

Rooms that are not on the path (the zeroes) are designated as "side rooms." Their defining characteristic is that the player may not necessarily be able to reach them without destroying the walls—they can be completely sealed off. Again, one of the nice things about destructible terrain is that it makes the random level generation easier, since we only need to worry about whether there is a clear path from the entrance to the exit. If the opening on the edge of the path happens to connect with an opening on the edge of a side room, that's great, but if not, that's okay. To the player, openings that end up meeting with walls will simply look like they were walls to begin with.

After the path is set, each room will run a script called "scrRoomGen" that builds the room out of tiles. First, it randomly selects a room template out of a set of

six to twelve templates, depending on the type of room. Templates are based on themes such as "basic room," "low ceiling," "ladders," "upper platforms," or "treasure below" (these names are taken from comments in the source code), and are defined in the script by a long string of characters. Here's a template from the Mines, for example:

```
0000000011006000OL040000000P110000000L1100
00000L11000000001100000000111112222111
```

That string doesn't make much sense unless you add carriage returns after every tenth character, turning it into this:

```
0000000011
0060000L04
0000000P11
0000000L11
0000000L11
0000000011
0000000011
1112222111
```

Now, at least, it's in the shape of a room. In this form, the room almost looks like a roguelike level, where each character represents something in the game. The 0's are empty space, the 1's are basic walls, the L's are ladders, the P is a ladder with a wooden platform cutting across it, and the 4 is a stone block that can be pushed horizontally. The 2's have a 50% chance to be

a wall and a 50% chance to be empty space. Once you know what each character represents, it's easy to see that this is a room where a ladder on the right side leads to a small passageway plugged up by a stone block. Because of the 2's, there's a good chance that there will be an opening downward, but it could also be flat ground.

Later on in the script these characters get converted into graphical tiles.

The "6" floating in mid-air doesn't represent a single tile, but a 5x3 group of tiles that I call a "chunk," randomly selected out of a possible ten. One chunk might look like this:

```
01110
02220
00000
```

That's a platform three tiles wide (the 1's), with some stray tiles possibly sticking out at the bottom (the 2's). Once the program replaces the "6" in the room with this chunk, it looks like this:

```
0000000011
0001110L04
0002220P11
0000000L11
0000000L11
0000000011
0000000011
1112222111
```

The chunks come in two main varieties: chunks that are meant to hang in the air and chunks that are meant to sit on the floor. There are some special varieties, too, like in the Jungle, where there are chunks for groups of hanging vines. The purpose of these smaller groupings of tiles is to provide further randomization within a room, allowing me to create what feels like a lot of variety with a fairly small number of room templates.

Once scrRoomGen has done its thing, it's time to generate monsters, traps, items, and treasures, in a new script called "scrEntityGen." This script checks each floor tile and decides whether to generate an entity on or around the tile. It will, for example, check that there's an empty space above the tile and then roll the proverbial dice to possibly generate a monster there. Item crates, treasure chests, gems, and gold like to spawn in alcoves where there is an empty space surrounded by walls in three out of the four directions. In this fashion, the level is populated with all of the deadly obstacles and tempting rewards that make the game fun.

I've been asked how I come up with the percentages for how frequently monsters, traps, and items are generated in *Spelunky*. The truth, which may be a relief or maddening depending on how you look at it, is that it's not an exact science—a lot of trial and error is required. Usually, I find it best to just start with some ballpark figure, like 1/10, 1/20, 1/50, or 1/100. From there, values are just tweaked up and down until they feel right.

Take this block of "if" statements from the source code, for example:

```
if (rand(1,60) == 1)
instance_create(x, y-16, oManTrap);

else if (rand(1,60) == 1)
instance_create(x, y-16, oCaveman);

else if (rand(1,120) == 1)
instance_create(x, y-16, oFireFrog);

else if (rand(1,30) == 1)
instance_create(x, y-16, oFrog);
```

This code is run when the game finds a floor tile in the Jungle that is not already occupied by anything and isn't in a shop or some other monster-free space. Basically, what this says is that, on top of a suitable floor tile in that area, there's a 1/60, or around 1.67%, chance to spawn a mantrap (man-eating plant), otherwise there's a 1/60 chance to spawn a caveman, otherwise there's a 1/120 chance to spawn a fire frog. And finally, if none of those spawn, then there's a 1/30 chance to spawn a regular frog.

The gist of it is that frogs are plentiful, mantraps and cavemen appear at an "average," "normal" amount, and fire frogs are quite rare (about twice as rare as cavemen and four times as rare as regular frogs). That gist is more

important than the numbers themselves. Would 1/65 be a better percentage to spawn cavemen at than 1/60? It's probably not worth taking the time to find out, even if finding out was possible. The only thing I needed to understand was that, after playing the game many times, 1/60 seemed to spawn a reasonable number of a monster per level.

The randomized level algorithm in *Spelunky* is the perfect example of the kind of programming exercise I enjoy. In this case, it was my distinct lack of passion for programming that led me to the idea, since I didn't want to do anything more complicated. This system doesn't create the most natural-looking caves ever, and players will quickly begin to recognize certain repeating landmarks and perhaps even sense that the levels are generated on a grid. But with enough templates and random mutations, there's still plenty of variability. More importantly, it creates fun and engaging levels that the player can't easily get stuck in, something much more valuable than realism when it comes to making an immersive experience.

TIGSource

Over the summer of 2008, the source code for *Spelunky* accompanied me wherever I went, tucked away in the Windows XP partition of a MacBook Pro. I worked on the game in complete secrecy during this time—after

sharing *Aquaria* with Alec, publishers, fans, and press, it felt nice to have something that was entirely my own.

By September, however, pressure started to build up as I realized I might be creating something pretty cool. Or was I? That's the problem with working in secret—without someone else there, you feel like you could be going crazy, doing bad work that only seems good in a vacuum. I also had an uneasy feeling that if I didn't release *Spelunky* soon, someone else might put out a platformer/roguelike hybrid before I could. With platformers as a perennial favorite of indie game developers and roguelikes gaining in popularity, it seemed like only a matter of time.

Since *Spelunky* was still quite buggy and lacking in features that I wanted to add, I was hesitant to make an official release, but I knew a relatively private place where I could show the game to a small group of indie game developers and enthusiasts: The Independent Gaming Source, also known as TIGSource.

My relationship with TIGSource began before *Aquaria* and long before *Spelunky*, in 2005. I was at a bit of a low point—Berkeley had spat me out a year earlier with a computer science degree, but not much desire nor energy to code. I had been unfocused my senior year, distracted by a break-up with my long-term girlfriend, and ended up failing a self-directed programming course that I had planned to spend making a game. Before that, my long-time game development partner Jon Perry had lost interest in working on video

games, choosing to focus on his film degree and other hobbies instead. I was tired and lonely.

Around this time, Jordan Magnuson, the founder of TIGSource, interviewed me about the freeware games I had made with Jon, and during the interview I explained the pessimism I had toward my future as a professional game developer:

Jordan:
I remember once that you had this idea of making a living making games… is that idea still around, or are you planning on doing something else to bring in your bread and butter?

Me:
Ah, hmmm… well, somewhere around college, I think I got a bit jaded about the games industry. I feel like it's a hard place for creative people. Most people want to be Sid Meier, or Warren Spector, or Shigeru Miyamoto, or something. But I started wondering: If I tried to get in the games industry, how many Spongebob Squarepants games would I have to make before I could get to do what I wanted? And making a living off of indie games is hard.

I guess it's like any industry: you have to find the right people to work with. To be honest, college also burned me out. Between trying to do art, make games, study for classes, keep up

> a serious relationship, go out and have fun and meet new friends… If you want to be a game programmer, you have to really love programming. After college, I wasn't sure that I really did, I guess. And you can't just jump into the games industry as a designer.

After my interview with Jordan, TIGSource became a regular stop in my daily surfing. Previously, the only game development community I was familiar with was the Klik & Play community, and after reading TIGSource, I realized that the Klik community was just a small puddle in the larger tide pool of indie games on the edge of the vast ocean that was gaming. In many ways, indie gaming was familiar: People were making games in very small teams, having multiple roles in the development, and releasing them outside the traditional publisher system. But after Jon and I released our final Klik game, a fantasy platformer called *Eternal Daughter*, I felt like we had hit a ceiling in terms of what could be accomplished with those tools. On a technical level, we had pushed Multimedia Fusion (MMF), the successor to Klik & Play, to its breaking point—*Eternal Daughter*'s final source file was a tangled mess of workarounds that Jon developed to get around MMF's many bugs and lack of features. The Klik community itself seemed to stagnate around that time, perhaps because it was a community based around such a limited set of software. When the limits

of the tools were reached, the limits of the community were also reached. Long-standing members either moved to making games in mainstream game studios or they left gaming altogether to pursue other interests. By contrast, in the wider world of indie gaming people used all kinds of tools, and without that limitation they could turn their hobbies into careers.

Unfortunately, shortly after our interview, Jordan was forced to abandon TIGSource due to personal responsibilities. There were other indie gaming blogs around, like Bytten and Game Tunnel, but they sorely lacked Jordan's honest and sometimes acerbic style, which rankled developers with harsh critiques and confronted the community's drama head-on. It was a raw and passionate voice that reminded me of some of the popular websites that sprouted up in the heyday of the Klik community, making me nostalgic for that period of my life and excited about the future of gaming. I was already considering asking Jordan for an editor position on the site, but since he was going to abandon it, I emailed him asking if I could take it over instead.

According to Jordan, there was no one he'd rather have running the site. I immediately went to work on it, giving the site's bare-bones Blogspot template a visual makeover and keeping the flow of news updates going. It couldn't have happened at a better time for me, since I was lacking a worthwhile project to pour my heart into and feeling restless because of it.

I even worked on TIGSource regularly during the development of *Aquaria*, and through the site Alec and I became friends with other indies. This group felt like a family, and we worked together to define an "indie spirit" that existed outside of both the big-budget interactive movies and the low-budget casual clones. Between working on *Aquaria*, writing about games for TIGSource, and arguing about what "indie games" were with the various people that passed through the site, I was immersed in the indie game community whole hog.

By the time I was ready to show off *Spelunky*, thousands of active members had transformed TIGSource into a vibrant community for indie game developers and enthusiasts, with a forum where they could share their games, set up jam sessions, and participate in game-making competitions. Within the TIGSource Forums, or "TIGForums," I created a hidden subforum for moderators and regular members with high post counts. It was here that I decided to share *Spelunky* for the first time, with a small subset of the indie gaming community. To avoid raising expectations too high, I called it "Spelunky 0.8 BETA."

The feedback from the gallery was swift and plentiful. There were still a lot of bugs to work out, but a palpable feeling of excitement built up as people delved into the game. On the first page of the thread, one player exclaimed: "EPIC MOMENT: When a spider jumped at my face but was killed by an arrow

from a trap." The excitement was shared on both sides—after months of working silently, it was a great relief that the game was well-received, and scrolling through the thread was like reading the banter of actual explorers eager to share their exploits with one another. I loved it.

Spurred by the enthusiasm of the beta testers, I raced to fix bugs and add more content, which, in turn, increased their enthusiasm. This positive feedback loop was exactly what I was hoping for, and I rode it hard for the next three months, preparing to unleash the game on the rest of the world.

The Shopkeeper

Among the most important things I added in this frantic period were shops where the player could acquire items. Shops are in nearly every kind of game, giving players a way to spend in-game currency and customize their characters. In some games, they're also used as a "speed bump" that forces you to grind out the money to buy equipment required to progress. In *Spelunky*, shops won't slow you down, but they are one of the biggest causes of death for players old and new.

The shopkeeper has become the most notorious character in *Spelunky*, equally feared and beloved by his patrons. His shop, which appears 100% of the time in level 2 and has a decreasing chance of appearing in later

levels (2/3 chance in level 3, 2/4 in level 4, 2/5 in level 5, and so on), is the most consistent source of life-saving items as you spelunk your way downward. Given the utility of the items, they carry hefty price tags that increase with each level: A sack of extra bombs will cost upwards of $2,500, and more exotic tools like the jetpack can reach into the tens of thousands of dollars. Players will rarely have enough money to purchase every item in a shop, and sometimes nothing will be affordable. Money is also used as the player's score, so a shopping spree will damage your standing in the leaderboards.

The difficulty of buying all the items you want and the consequence to your score creates a strong temptation to steal from the shop. In most games, you only have two options when it comes to shopping: buy something or leave. The complexities of dealing with stealing are beyond the scope of those games—not only does it require adding mechanics to allow the player to leave with an item unpaid, but the ethical consequences of the player's theft have to be dealt with, as well. Letting the player commit a crime may not fit a narrative where the player is supposed to be a hero.[4]

4 *The Legend of Zelda: Link's Awakening* for the Game Boy handles thievery in an interesting way: The game will change your name to "THIEF" and the next time you return to the shop, the shopkeeper will kill you. There are no other consequences.

But if some action in a roguelike seems like it could be possible, it *should* be possible. This is a genre where in all of the canon titles you can suffer penalties from overeating—of course you can steal from shops! In *NetHack*, there are many ways to go about this. The most direct method, of course, is to leave the shop with the unpaid items, but the shopkeepers will move to block the door as soon as you pick something up. Given that they sell all kinds of potentially useful items and equipment, they're also quite tough—*NetHack* itself has an idea of how difficult each monster is, and the shopkeeper's in-game difficulty value is 15 out of a possible 57, placing him between a vampire lord and a pit fiend in terms of scariness. Attacking one early on in the game is suicidal, but there are other, nonviolent ways to rob them, like teleporting them away, teleporting yourself away, digging through the shop's wall, or training your pet to bring the items outside for you. Less orthodox shoplifting strategies include polymorphing the shopkeeper into a weaker monster and then killing that, or having a gelatinous cube eat the items and then killing the cube after it exits the shop.

Even if one could easily afford to buy every item in the game, it's simply human nature to see if you can get away with bad behavior, something I can attest to as the father of a toddler. Testing boundaries in a safe environment is how babies grow into confident and independent adults, and in a similar fashion, well-designed

video games can help people grow into confident and independent players.

One of the differences that I've noticed between inexperienced and experienced players is that new players are much more timid when playing a game for the first time. The idea of failing makes them anxious. On the other hand, experienced players will seek out the boundaries of a new game as quickly as possible, even if it involves dying over and over again. They understand that, not only is it a more fun way to play, but the knowledge they gain from it will help them succeed in the long run.

Ideally, I wanted the world of *Spelunky* to be fraught with high-risk/high-reward scenarios, because those are the most tempting ones for players who are trying to find the boundaries of the game. Among those scenarios, the shop offers some of the highest risks and rewards, since robbing one gives you access to a wealth of free items but also plasters your face on wanted posters in every shop afterward. Not only that, but a hostile shopkeeper will spawn at the exit of each successive level. *Spelunky*'s shopkeeper is much stronger than the other monsters in the game, too. Not only is he impervious to being whipped, but he also runs quickly and can easily kill you from a distance with his shotgun or, if he catches you, by throwing you into the wall.

But as difficult as he can be, the shopkeeper has a fairly simple set of behaviors. While he's alive, there are four main "states" that he can be in:

- IDLE is a resting state when he's standing still and watching over his shop.
- FOLLOW is a state he enters when the player picks up an item in his shop, causing him to walk up to the player and stay close by. Whether he's following to offer assistance or out of suspicion is left to the player's imagination. I added this state because in initial tests it was too easy to steal items when he stayed at his counter.
- ATTACK is the shopkeeper's angry state, when he's actively hunting down the player. And in the world of *Spelunky* he is the judge, jury, and executioner. If he's not already carrying a shotgun when he enters this state, he will pull one out. He can only pull out one shotgun, though, so if you manage to wrest it from his grasp, he'll become unarmed (but still dangerous).
- PATROL is a state that replaces IDLE when the player is wanted as a criminal. In this state, the shopkeeper will walk back and forth holding the shotgun until the player draws near, in which case he will enter his ATTACK state.

In the ATTACK state, the shopkeeper is fairly adept at finding the player, although the underlying AI is nothing special. Pathfinding is an area with a lot of research behind it and there are a variety of algorithms

that people use to get a non-player character (NPC) from one place to another. For the shopkeeper, however, my simple solutions worked just fine. His ATTACK AI can be summed up like this:

- Every now and then, turn toward the player.
- Every now and then, fire my shotgun at the player if they are close enough and I have the shotgun.
- If the player is below me, and not too far away from me horizontally, do nothing.
- Otherwise, if there's an obstacle directly in front of me, jump.
- Otherwise, if the player is above me, jump.

Basically, when the player is at the same height or above the shopkeeper, or far enough away from him to the left or right, he likes to jump… a lot. The strength of his jump is random, too. I wasn't sure how this would work when I first coded it, but it turns out that jumping around a lot at random heights is an effective method of getting around obstacles and finding little gaps in the wall to get through. The repeated jumping also makes his movements seem manic and unpredictable, increasing his danger factor and going hand-in-hand with his angry and violent persona. It's a great example of how thematic imagery can mask simple mechanics.

Feedback Loop

On December 21, 2008, I felt like the game was finally ready for a wider audience. I posted it publicly to the Feedback section of TIGForums: "Submitted for the approval of The Midnight Society, I call this story… SPELUNKY! This is my latest game release and my first Game Maker game."

The public release blew up and the thread quickly became the top-viewed thread on the TIGSource forums (today it's the fifth most-viewed, with nearly 1.3 million views). The reaction to the game was overall very positive, but also surprisingly contentious. Initially, people were most likely drawn to the thread because of my role in the site and in *Aquaria*, but the fact that it kept heating up months later was thanks to the intensity of people's opinions about the game, good and bad. Almost immediately, there were three vocal camps: people who loved the game, people who wanted to love the game, and people who already thought the game was overrated. These groups squabbled furiously with one another, creating drama that, combined with some positive write-ups in blogs elsewhere, ensured a steady flow of visitors to the thread.

The players who wanted to love *Spelunky* were fascinated by the core idea—a platformer with randomized levels—but were put off by some combination of the game's high difficulty level, the complex controls, the restrictive nature of the game's item management, or

the invincible and unrelenting ghost monster. These hurdles worked together to create frustrating situations where players would wrestle with the controls to do relatively simple things and end up getting killed either because they pressed the wrong button or because the fumbling delayed them enough that the ghost appeared and hunted them down. All the bugs in the project didn't help matters, either.

That initial blast of feedback can be overwhelming. It feels like a private conversation you've been having with some close friends has suddenly been opened up to the entire world. If you've done a good job, you may get showered with praise, but no matter how amazing your game is, you'll also deal with people who are unsatisfied with your work. And the internet being what it is, you can't count on them to be tactful, even-handed, or rational. You can't even count on them to enjoy video games at all.

Unfortunately, it's easy to float quickly across positive comments only to sink into the negative ones. In a thread with all positive comments and one negative comment, it's usually the negative one that stands out the most, like a rusty barb along a wire. This is particularly true if you are a small developer without the benefit of a large team to cushion the impact. It makes you feel vulnerable and defensive.

One *Spelunky* critic that still stands out to me from that early deluge of feedback was adept at hurling his frustrations back at me. "Derek, I hate your fuckin'

game," he said. "Because it's the first roguelike/platformer implementation I've ever seen, which is totally awesome. But from this game it seems you suck at difficulty." While he and I agreed that a player's death should always be attributed to their own mistakes, we disagreed on what actually constituted a player's mistake. In his mind, a death that resulted from his own self-described lack of "natural skills" at video games—including good dexterity and reflexes—was the game's fault and not his. "Feeling cheated and insulted by a game is not fun," he exclaimed. "Unless a person is brainwashed or mentally handicapped." When I die in a game, it's frustrating but it makes me want to keep trying and improve. To him, it was insulting.

It's not pleasant being told that you suck at anything, whether it's true or not. These kinds of pejoratives make the legitimate criticism that follows a lot harder to swallow, and at the time I wished nothing more than for this person to take his lack of natural skill away and leave me alone. There were a couple of reasons why I didn't ignore him, though. The first reason is related to an adage that's attributed to Elie Wiesel: "The opposite of love is not hate, it's indifference." It was clear that what was driving his anger was a whole lot of passion, and I would rather read lengthy diatribes about how to fix my game than simply gaze into an empty thread. The second reason is that, above all else, I want to make my games as good as they possibly can be. Given how much my games mean to

me, it seems silly to let a few disparaging remarks get in the way of doing my best work.

This isn't to say that everyone who gives you feedback is right or that all feedback is equally useful. How could it be, when so much of the feedback is conflicting? But to determine what's right and what's not, it's important to listen. A single piece of feedback might seem worthless on its own, but when placed next to other pieces of feedback, it starts to have some meaning. In the case of that one angry player, although I didn't agree with his opinions about skill levels or his insistence that I include difficulty options for players like him, I felt like there could be some credence to the idea that the game was too difficult, since other, more level-headed players shared the same sentiment. What was the real reason, though?

While I mulled over this problem, I set about fixing bugs and implementing changes suggested by other players that seemed obviously good, like reducing the complexity of the controls and the number of steps that were required to perform certain actions. Interestingly enough, I noticed from the angry player's posts that while he was getting more and more angry, he was also making progress, getting to the later levels and eventually beating the game. This convinced me that the difficulty was not itself the problem, and that I was right to not include an easy mode in the game, since it would have become an unnecessary crutch for players like him. The real culprits were all of the aggregate

smaller annoyances that made interacting with the game more difficult. *Spelunky* did get easier with each update, but in a way that improved the core experience rather than watering it down. So in some sense, that angry player was right: The game was too hard. But not for the reasons that he or I assumed. If we were talking about the game of tennis instead of *Spelunky*, it's as though this player asked me to remove the net because he was having trouble hitting the ball over it, neither of us realizing that the real problem was that I gave him a racket with broken strings.

Another way to think about interpreting a player's feedback is as a doctor diagnosing a patient. A patient brings his or her symptoms to the doctor ("My stomach hurts!") and may even offer a possible solution ("I think it could be the flu!"). In that scenario, it'd be a bad idea for the doctor to either dismiss the validity of the patient's symptoms or to blindly accept that they have what they say they have. Like patients, players are often most in tune with how they feel about the problem rather than what is causing it. Doctors and game developers, with their experience and knowledge of their field, need to ask questions, perform tests, and eliminate possibilities in order to zero in on the correct solution.

Game developers tend to get feedback from other game developers, too, but that feedback can also be misleading. Another game developer might be able to articulate problems that your average player can't, but their understanding of the development process may

cause them to miss the forest for the trees, obsessing about minutiae that most players won't notice. This becomes even worse if the developer is a friend or someone from your social circle, since it adds a layer of sympathy that the honesty must cut through. I've often felt the temptation to suggest to another indie developer that they fix this or that small problem when something more fundamental was the real problem. It can be hard to tell someone to start over when you know how hard they've worked.

But regardless of their relationship to me, each person who played *Spelunky* brought in their own expectations of what *Spelunky* was and what it should be. Whether I was defending my design choices or implementing others' suggestions, their feedback tested my vision for *Spelunky*—allowing me to rethink decisions in some places, hardening my resolve in others. The truth is that developing a vision never really ends. When we think of a "vision," we often think of something that feels crystalline and hard, when in fact it's something much wetter and squishier. Over time, that wet and squishy vision can be nurtured by creator and audience into something more solid, but it takes time. When I made *Spelunky*, I was totally in the moment. Like many of the doodles in my school notebooks, it started off as something small and flippant in the margins and quickly took over the entire page. As such, I didn't have a lot of insight into the ramifications of what I was doing, only whether or not something "felt right" to do.

A lot of my decision-making was instinctual, built up over many years of thinking about games. It was only after the audience entered the picture that I started to understand the reasoning behind my own designs.

PART II:

SPELUNKY UNLOCKED

Or are there even greater delights to be discovered?
Well, there's only one way to find out!

— Yang

Indifference

I PLAYED GAMES EVERYWHERE as a kid—on my parents' PC and their Atari 2600, at the arcades, in the car with my Game Boy, and at friends' houses where I was introduced to Chinese pirate multicarts[5] and exotic game systems like the Neo Geo and TurboGrafx-16. But for me, that era still belongs to Nintendo. My uncle was the first in my family to get a Nintendo Entertainment System (NES), and I spent entire visits playing *Super Mario Bros.* and *Duck Hunt*. When I wasn't playing, I'd read my new issue of *Nintendo Power* compulsively until the next month's issue. No one in the 80s built worlds as magical and well-crafted as Nintendo did. And although many talented men and women deserve credit for that, the one who stands out among them all is the developer who I was most excited to see in the crowd at IGF 2007: Shigeru Miyamoto.

Miyamoto once said that his childhood exploration of the Kyoto countryside was the inspiration for creating *The Legend of Zelda*, a top-down action-adventure game set in the fictional land of Hyrule. Recalling the time he discovered a lake while hiking, he explained, "It

5 These are illegal cartridges that have multiple games on them, sometimes hundreds.

was quite a surprise for me to stumble upon it. When I traveled around the country without a map, trying to find my way, stumbling on amazing things as I went, I realized how it felt to go on an adventure like this." It's the perfect way to describe my experience with *The Legend of Zelda* as a child, when my dad and I spent many hours meticulously exploring and mapping Hyrule. As I moved from screen to screen, slaying monsters and prodding the environment for hidden secrets, he would mark them down on our map with colored pencils.

It felt like we were Lewis and Clark trekking across the American West. I'll never forget the first time I entered a dungeon and watched the bright greens, browns, and yellows of the overworld give way to ominous blues and reds—the sound of Link's footsteps on stairs heralding the eerie dungeon music that still echoes in every Nintendo kid's ears. It seems strange now, but in *The Legend of Zelda* no one tells you where the first dungeon is located. It's possible to wander into the farthest reaches of Hyrule before locating it, and when you find the entrance—a gaping black "mouth" beckoning you into a giant tree—you may not necessarily know what you have found.

In a 2003 interview with *SuperPlay* magazine, Miyamoto recalled the day the game was released: "I remember that we were very nervous since *The Legend of Zelda* was our first game that forced the players to think what they should do next." This bold and risky design, based on the joy of discovery, had a huge

impact on me as a game designer. In *Spelunky*, as in all of my games, I wanted to capture the same emotions I had on that first adventure.

Unfortunately, that feeling about Hyrule waned with each successive game. Even as the worlds grew more beautiful and vibrant, a feeling of disappointment clouded my initial wondrous experience. Part of it is that I grew up. Zelda is 30 years old now, and in that time I've played 30 years' worth of games and released some of my own. But while it's harder to surprise me now, it also doesn't appear that the series is as interested in trying. If the original Zelda game was made only for children, I might chalk it up to my age, but revisiting it as a "Classic Series" Game Boy Advance reissue, I was amazed at how strange and wild it still felt compared to the later games, and to modern games in general. It was like returning to the wilderness after a long hiatus, trying to get back in touch with senses that had been steadily dulled.

In Tevis Thompson's brilliant 2012 essay "Saving Zelda," Thompson likened modern installments of the game to theme parks, saying, "*Skyward Sword*, with its segregated, recycled areas and puzzly overworld dungeons, is not an outlier; it is the culmination of years of reducing the world to a series of bottlenecks, to a kiddie theme park (this is not an exaggeration: Lanayru Desert has a roller-coaster)." Gone is the wild frontier that I explored with my dad and the Kyoto countryside that inspired the series, replaced

by something that feels too linear, too elegant, too smooth, too… designed? Quests have been turned into fun house games with obvious goals and rewards. "Secrets" are outlined with bright, flashing signposts. A theme park is exactly what it feels like.

Is a theme park necessarily a bad thing, though? I also have great memories of going to Disneyland, Magic Mountain, and other amusement parks. But leaving the park after a full day of riding rides and eating cotton candy, I'm not eager to go back the next day or even the next week or month. The thrills are garish and over-the-top, but also obvious and safe. Compare a theme park to that Kyoto countryside—Miyamoto purportedly came across a cave during his explorations and hesitated for days before eventually going inside. Why did that cave feel so dangerous to him, even though there was likely nothing inside? Why did my wife and I feel the same trepidation as adults in Hawaii, when we ducked into a little path carved into a bamboo field off the side of the road?

Thompson continues:

> Hyrule must become more indifferent to the player. It must aspire to ignore Link. Zelda has so far resisted the urge to lavish choice on the player and respond to his every whim, but it follows a similar spirit of indulgence in its loving details, its carefully crafted adventure that reeks of quality and just-for-you-ness. But a world is not for you.

> A world needs a substance, an independence, a sense that it doesn't just disappear when you turn around (even if it kinda does). It needs architecture, not level design with themed wallpaper, and environments with their own ecosystems (which were doing just fine before you showed up). Every location can't be plagued with false crises only you can solve, grist for the storymill.

It's easy to mistake Thompson's assertion that "Hyrule must become more indifferent to the player" for an assertion that game developers shouldn't care about the player or shouldn't guide the player toward their ultimate vision. What it means is that the guides must be a natural part of the world, and the world, like Miyamoto's cave, must simply exist. If a world is independent and self-sufficient, so are its inhabitants. If every part of a world exists only for the player, both the world and the hero will feel artificial.

Nintendo wasn't the only developer to lose sight of that cave in Kyoto. All game creators must control the player's experience to a degree, and it's easy to take it too far—this is particularly true of large studios with bigger budgets that they have to recoup from audiences that include many casual players. Designers often mistake intentionality for good game design: We think that a cave must have a treasure chest in it, and if there's a treasure chest it must be guarded by a monster, and if there's treasure in the cave, then the player must find it,

and if the player must find it, then there has to be a map that leads the player to the cave. That feels like good design because we took the time to plan it out and in the end the player did what we expected. But it doesn't guarantee that the player will feel like they're on a true adventure, making genuine discoveries.

Creating *Spelunky* was the perfect project to help me think about what a true adventure meant to me. Working by myself on a small freeware game made it easier to focus on my personal vision instead of what other people wanted. Using Game Maker allowed me to focus on game design rather than technology. And then there was the randomization of the levels, which made it impossible to fully control the player's experience. All I could do was create the building blocks of the world and set them in motion—what came out could be as surprising and indifferent to me as it was to the players.

Arcade

Like Miyamoto, I also spent quite a bit of time outdoors as a child, although it was the virtual caves like the ones in *The Legend of Zelda* that attracted me the most. Outside of the house, however, there was another place I visited that evoked the flashing lights and garishness of a theme park, but nonetheless felt a little bit dangerous and a little bit wild: the arcade. Arcades used to be everywhere. In my suburban hometown of Pasadena,

California, there was the famous Pak Mann Arcade, which had the best selection of games but never shook its reputation for being a seedy hangout for "Asian gangsters." And then there was Tilt, tucked away neatly in a corner of the mall, where I would weave my way to the back to beat up thugs in *The Punisher*. I also spent more than a few quarters on the *Mortal Kombat* cabinet in a nearby ice skating rink, trying to rip out people's spines as Sub-Zero.

Arcade games were even more violent, zany, and sexualized than the games I played at home, and arcades themselves were dimly lit and smelled like sweat and smoke. The brevity of my experiences with these games only added to their mystique—more often than not, a visit to the arcade was just a side trip, somewhere to spend a few quarters before moving on. When my elementary school friends and I became obsessed with *Street Fighter II*, we spent more time theorizing about the game than actually playing it.

The home and arcade gaming experiences have always been defined, and separated, by how they're played. In PC and console gaming, single-player campaigns are sometimes expected to last hundreds of hours long, but the optimal length of an average arcade game is around 30 minutes to an hour—short enough to play through in a single session, standing on your feet. And while someone who has spent $60 on a big budget home game might be willing to sit through long tutorials or cutscenes, those features would be suicidal

in the arcade environment, where the player's time is limited and they're buying games a quarter at a time. A dull game might tease the first quarter out of a new player, but it will quickly lose the rest to any number of great-looking games a few feet away. Looking back at the arcades of the 80s and 90s, it's easy to see the games that stood the test of time were lean, sinuous animals that offered a lot of challenge and excitement in a short amount of time.

By the time I took over TIGSource in 2005, I had become increasingly frustrated by mainstream home gaming for a multitude of reasons: the "theme parking," the long tutorials, the hand-holding, and the constant interruptions of play for hints and cutscenes. Time and time again I'd be drawn to a new big-budget game by the promise of advanced graphics and technology, only to be hamstrung by game design that felt like a step backwards. As a result, I took a renewed interest in both the games of my childhood and indie games, which seemed less susceptible to modern design traps. I realized that I was most attracted to the dense and challenging gaming experiences that these games offered. Arcade games in particular held a special significance because the arcade experience itself seemed indifferent to you—you never owned any of the games at an arcade, you could only buy some time on the machines. And how much time you bought depended purely on your skill and knowledge.

My interest in arcade games also made its way into *Spelunky*. Just like I set out to mix and match my favorite

parts of roguelikes with my favorite parts of platformers, I also set out to recreate that arcade experience in an environment where worrying about quarters was a thing of the past. Just because people were spending more time playing games at home, it didn't mean that their time should be any less respected.

Beating *Spelunky* from beginning to end takes, on average, about 30 minutes to an hour, that perfect arcade length, and there's pressure to move quickly. Lingering too long in a level will induce a scary message ("A chill runs up your spine!"), followed by a steep drop in the music's pitch. The source of this nightmarish transformation is revealed soon enough as the ghost floats onscreen from somewhere outside the level, its mouth gaping open, and begins to slowly give chase. The warning that precedes its entrance is clear enough: instant death if it touches you!

Timers are a popular carry-over from arcade games that depended on the patronage of easily distracted crowds and were designed to prevent any one player from hogging the machine. Anyone who's spent time in arcades during the 80s and 90s is probably familiar with the sight of a frantic arrow or cartoon finger pointing to one side of the screen, emblazoned with the word "GO!"—the games say it so that the people waiting to play don't have to. When timers in video games run down, they typically kill the player immediately like in *Super Mario Bros.* (i.e. a "hard timer") or make life more difficult for them (i.e. a "soft timer"). In Taito's *Bubble*

Bobble, the command "Hurry Up!" is displayed on the screen in suitably bubbly letters before the frightening "Baron von Blubba" appears onscreen and begins pursuing the players.

Does a timer make sense for someone playing *Spelunky* in the comfort of their own home, though? Why not let them take their time if they want to? I think it depends on what type of experience you want to give your audience. While the arcade business model may have shaped the design of the games, in many ways it shaped them well—maximizing a player's time was a trait of successful arcade titles.

Ultimately, I never intended *Spelunky* players to collect every piece of treasure, get every item, or explore every room each time they play. Instead, I wanted to force them to make difficult decisions and experience both the satisfaction of choosing correctly and the regret of choosing poorly. Collecting treasure and acquiring items without limitation is joyless, but add some time pressure and it becomes an exciting dilemma that makes skillful play more meaningful. Good players can make better decisions within the time constraints and are rewarded with better results. In *Spelunky*, they can even use the ghost to their advantage: Leading the apparition over large emeralds, sapphires, and rubies will cause them to change into more valuable diamonds.

But the arcade model wasn't perfect, by any means. Aside from not being able to support longer-form gaming experiences, arcade games suffer from a

poorly-designed continue system that allows players to recover from a game over by putting in more quarters. That continue system, combined with their high difficulty, gave arcade games a reputation for being "quarter-munchers," a stigma that contributed to their decline over the years. It's not dissimilar to the reputation that modern "free-to-play" games have for sucking you in with the promise of a free game to get you to buy in-app purchases later on. Not surprisingly, paying to win is viewed negatively in an art form where the audience prides itself on playing to win. In both arcade gaming and free-to-play gaming, the abundance of cheap, exploitative titles has only strengthened their reputation as being greedy and manipulative.

What people who think of arcade games as quarter-munchers don't realize is that good arcade games can be beaten on a single credit (known as a "1 credit clear," or "1cc," for short). What gives them their longevity isn't the amount of time a single playthrough takes, but how long it takes you to 1cc them. Seasoned players will use continues sparingly to practice challenging levels, but will normally wait out the continue screen after a game over and start from the beginning. Playing this way, you'll find that after some practice you can clear more and more of the game on a single quarter. Consequently, your trips to the arcade will also begin to change character—instead of burning through your quarters in a few minutes and wandering the arcade, you might find yourself being able to sit in front of a

machine for half an hour or more, attracting your own spectators in the process.

That this is how the games are meant to be played is hinted at in their high score systems. Until online leaderboards became ubiquitous, we had come to think of high scores in console games largely as vestigial structures, but in arcade games, scores have always been important, offering a sense of personal progress and also acting as a way to compete with other players who you may never meet in person. In most arcade games, your score is reset to zero when you put in another quarter to continue. Only by obtaining a 1cc will you have a chance of reaching the top of the scoreboard.

"Credit-feeding," or making progress in an arcade game by continuing at every opportunity, will drain you of your quarters exponentially faster as you go on. I made this mistake once at Pak Mann Arcade with *Metal Slug 3*, a cup full of quarters, and Jon Perry (the co-creator of *Eternal Daughter*). The Metal Slug games by SNK are frequently hailed as having the best-looking pixel art ever made. They're also quite tough, and while the first two games are relatively short in length, *Metal Slug 3* has a final mission that, unbeknownst to me and Jon at the time, is nearly as long as a second game. Credit-feeding that stage, you can easily find yourself running out of lives in a few seconds, and adding another inexperienced player only makes the quarters run out faster. After each death we prayed that the final boss was getting close, only to be sucked into another

room full of its deadly minions. By the time we were watching the credits scroll—with our cup empty and our scores near zero—we wondered why we bothered.

Why did arcade game developers allow unlimited continuing if it's not the best way to play the games? I suppose my empty cup of quarters is one answer. Without continues, casual players would stop playing and move on to a new machine after a few deaths, since their primary goal isn't to 1cc a game, but to see as much as possible during their brief visit to the arcade. In other words, it's a way to fleece inexperienced patrons. Regulars, on the other hand, will spend the same number of quarters across a much longer period of time. Either way, the publishers and arcade owners earn their money.

In developing *Spelunky*, I wanted to avoid the hand-holding of modern Zelda games and the opaque continue system of arcade games, but I also wanted to give players a way to practice and offer a tangible sense of progression beyond the leaderboards. For that reason, I introduced the Tunnel Man, a happy-go-lucky, shovel-totin' entrepreneur who greets you in the transitional corridors between areas. If you give Tunnel Man enough money, he'll build permanent shortcuts from the game's lobby room to the later areas of the game. On each visit, players can donate some of their money toward unlocking a shortcut—$100,000 for the Jungle shortcut, $200,000 for the Ice Caves shortcut, and $300,000 for the Temple shortcut. Although everything

else that you gain in a run is lost when you win or die, your donations are saved, and once the shortcut to an area is dug, you can start a run there, bypassing earlier levels. In this way, players can chip away at something even when they're nowhere near completing the game.

The question is, once you've introduced shortcuts, how do you get players to stop using them of their own accord? I wanted players to realize that *Spelunky* was designed to be played from beginning to end in one sitting, like an arcade game, but I didn't want to say it explicitly in a text message, since that would rob them of the satisfaction of realizing it themselves.

My solution was in two parts. The first was already built into the basic design. How do you convince a kid to take training wheels off her bike? Simple: You don't need to, because bikes can be ridden faster and with more control once the training wheels are off. Once you understand *Spelunky* well enough, you come to realize that it is a huge boon to start each run from the beginning, because there are more opportunities to prepare yourself for the difficult later sections. Bombs and rope, for example, are easily obtained in crates or from the shop in the second level of the Mines, and these items are crucial to overcoming obstacles later on. Likewise, each level you skip is a missed chance of getting kissed by a damsel or currying favor with Kali by sacrificing monsters on her altars. It might also mean missing a chance to grab a shotgun from the hands of a stunned shopkeeper, along with all of his wares. A new

player may barely make it to the Jungle with anything, and that's when shortcuts are the most useful. But with more experience, they'll find it a setback to start there with only the basic equipment.

The second part of my solution was added deliberately. Recalling Miyamoto's saying that "a good idea solves many problems," I developed something that not only obliterates the long-term usefulness of shortcuts, but also serves as *Spelunky*'s biggest secret.

I call it "The Chain."

From Beginning to End

When designing a game, there's always a tug-of-war between wanting players to be proud of their accomplishments and not wanting them to get stuck. The easier a task is, the more people who can carry it out, but the less meaningful it will be. Examining the Zelda series through that lens, it seems that with each successive game, the designers focused on making the challenges and secrets more accessible above all else. In *The Legend of Zelda: A Link to the Past*, the series' Super Nintendo debut, "hidden" caves became marked by door-shaped cracks in the wall. In *The Legend of Zelda: Ocarina of Time*, Nintendo introduced the first of many sidekicks, a fairy named Navi who teaches the player the controls, notifies them when something of interest is nearby, and offers clues and hints for puzzles. These additions seem

so counter to the design of the original *Zelda*, which "forced the players to think what they should do next," that they almost feel like spin-offs rather than sequels.

If my experience adding secrets to games has taught me anything, it's that I always underestimate how good video game players are at finding them. Instead of worrying about how many players might be able to discover the Chain and complete it, I focused on making it feel like part of the world of *Spelunky*. That way, no matter who found it or how long it took them, it would be truly hidden, and not "hidden" with a wink and nudge.

I call it the Chain because it's not just a single secret, but a chain of secrets that culminates with the player entering a special level called the City of Gold. There are four major items in the Chain, corresponding to the four areas of the game, and each item is used to obtain the one in the next area. The first item is the Udjat Eye, locked inside a unique treasure chest with a golden lock—this chest and the gold key that unlocks it are generated in one of the last three levels of the Mines. The chest is always generated somewhere on the path from the entrance to the exit, but the key can be hidden in any alcove on the map.

In *Spelunky*, items come in different categories:

- Bombs and ropes are so fundamental that they're assigned their own buttons. You start with four of them each and can accumulate more as you keep playing.

- Items that you pick up and carry in your hands include rocks, shotguns, and corpses. Unlike bombs and rope, you can only ever carry one item at a time, which makes deciding what to carry an important strategic element.
- The jetpack and cape are worn on your back after they're picked up and are activated by pressing jump while in mid-air.
- The remaining items are equipped and cannot be removed afterward. They appear in the HUD (heads-up display, also known as a status bar), below the bomb and rope counters. These items—like the climbing gloves or spike boots—give the player special abilities that don't require any further actions on their part.

The Udjat Eye falls into that last category of items—once you equip it, it appears in the HUD. Visually, it's a golden slab, and on it is carved the Eye of Horus, a symbol that the ancient Egyptians regarded as protective. If you take the Udjat Eye to the Jungle, the second area of the game, it can help you find the entrance to a special level called the Black Market, hidden behind a square of dirt. The closer you get to the entrance, the faster the Eye will blink in your HUD (although it will only start once you're about half a room away).

Each of the four major items is also designed to have a purpose other than to complete the next part of the Chain. All too often, games have a problem of "too many

locks and keys," where the player is continuously barred by something that only has one solution, essentially functioning as a lock. Aside from literal locked chests or locked doors, one example of a lock-and-key scenario might be a character who owns a ship, but won't set sail with you until you rescue their cat from the nearby dungeon. There's nothing wrong with locks and keys when more interesting scenarios are mixed in or when the player has alternative paths they can pursue, but when locks and keys are overused in a linear game they can make the player's goals feel artificial.

Just revealing the Black Market entrance is useful enough, but the Udjat Eye also allows the player to see treasure and items that are hidden in the walls. This is especially important because it makes the Eye useful even if you miss the entrance. It's also possible to stumble upon the entrance without the Eye, and while I could have required the player to have the Eye before they could enter the Black Market, it would have made the discovery feel too forced.

The Black Market was created to expand upon *Spelunky*'s shopping experience, which added so much wonderful complexity to the game. I loved the various interactions the player could have with the shopkeeper, and how being a wanted criminal created a narrative thread that ran across levels, heightening the experience of being a treasure hunter. Unlike a normal level, the Black Market is mostly static in its layout and contains seven shops and seven shopkeepers. Two shops always

appear in the same place—the Dice House, where you can roll dice for a chance to win money and prizes, and a small stall that only sells one item, the Ankh. The other five shops are randomly generated from the remaining shop types.

The Ankh only ever appears in the Black Market, in that stall, and it costs $50,000, which is a large sum of money to have by that part of the game. In comparison, the next most expensive item is a jetpack, which costs between $26,000 and $30,000, depending on what level the Black Market is generated in. To be able to buy the Ankh, you'll have to be fairly thorough about collecting treasure and refrain from spending it on other shop items. Alternatively, you can try stealing the Ankh, but you'll be doing so at the risk of angering seven shopkeepers in a confined space.

The ankh is another symbol from ancient Egypt, a hieroglyph that represents eternal life. In *Spelunky*, it gives the player an extra chance after death by reviving them at the entrance to the level before shattering. This alone makes it one of the most valuable items, as there is no number of hearts that will save the player from falling onto spikes, being crushed, disappearing into a bottomless pit, being eaten by a mantrap, sinking into lava, or otherwise getting instantly killed.

That resurrection ability is also required to access the next part of the Chain, at the giant Moai statue that appears in one of the three final levels in the Ice Caves. When you reach the Moai, inspired by the famous stone

heads on Easter Island, you might notice that the ankh symbol is carved into its forehead, a hint that it's related to the Ankh item. The secret is that dying in that stage with the Ankh will shatter the item and trigger the usual regeneration sequence, except instead of reappearing at the level entrance, you'll appear inside of the Moai. Hidden there is a second exit, as well as the Hedjet, a pharaoh's crown that's shaped a bit like an upside down vase. In *Spelunky*, the Hedjet serves dual purposes—it prevents levels from being generated dark and, more importantly, it acts as a kind of identification card to let the player enter the City of Gold.

Because of how useful the Ankh is in terms of survival, purposefully killing oneself to trigger it is a big sacrifice. It's also easy to die before reaching the Moai, especially if you've chosen to steal the item. Therefore, carrying the Ankh to the Moai and giving it up to receive the Hedjet shows an incredible amount of skill and luck, as well as dedication to the task of completing the Chain.

Even after that sacrifice, the player still needs to defeat Anubis and obtain the Sceptre, a volatile weapon that, when paired with the Hedjet, acts as a key to unlock the door to the game's ultimate reward: The City of Gold. Just the name of the place invokes treasure, and the concept is inspired by real legends like El Dorado, a mythical golden empire from South America. In *Spelunky*, the walls of the City of Gold can be destroyed into gold nuggets that can raise your score to incredible

heights, upwards of $750,000 if you manage to enter it with a mattock or a lot of bombs. But just getting there is a tremendous achievement. By the time you reach the City of Gold, you've not only mastered *Spelunky*'s basic mechanics, but you've also unraveled a mystery whose existence is never explicitly mentioned in the game.

The Chain exemplifies the *Spelunky* experience, which is about tension and the feeling of discovery. That you can beat the entire game without realizing it exists elevates it above a "lock and key" obstacle. Since each section of the Chain can be discovered independently and serves its own purpose, the player can slowly piece the secret together just through normal play.

It also gives players a more challenging way to play the game without resorting to difficulty settings. The intention of difficulty settings is to allow people to play at their individual skill level, but the settings are inherently bad design—when confronted with upfront options like "EASY," "NORMAL," "HARD," and "INSANE," the player is forced to make an important decision that will affect their entire experience without any of the relevant information. Is this game's NORMAL mode as hard as other games' HARD modes? Which setting is the game primarily designed around? These are questions that players can't possibly answer until they've started playing, causing unnecessary stress about whether the right choice has been made. As if to highlight this problem, *Wolfenstein 3D* famously shames players for choosing the easiest difficulty by calling it "Can I play,

Daddy?" and displaying the hero's face on the selection screen with a baby bonnet and pacifier. Unfortunately, the solution that modern game developers have devised to fix this problem—asking the player to switch to an easier difficulty after they've died—only makes it worse. This not only fails to alleviate the original problem, which happens at the start of the game, but it also mixes in the condescension of constant hand-holding.

There is another problem with difficulty settings, which was made apparent during my discussion with the angry *Spelunky* player: Offer players an easier difficulty setting up front, and some of them will take it even though they could handle more challenge. My solution to this was to weave difficulty seamlessly into the game itself. If you're a new player, your goal is to deliver supplies to the Tunnel Man so he can build shortcuts to the Jungle, Caves, and Temples. After that, your goal is to defeat Olmec and beat the game. From there, you're ready to tackle the Chain. But whether you're mining out the City of Gold or simply trying escape the Mines, you're playing the same *Spelunky*. This way, everyone can focus on playing instead of wringing their hands over an uninformed decision. Once you're ready, your knowledge and understanding of the game will make the harder challenges apparent, and you can go after them without switching into a different mode.

The Chain's final purpose is to provide permanent features to a randomized landscape. Although randomization gives *Spelunky* and roguelikes their longevity

and makes each playthrough feel unique, it's what stays the same from run to run that makes the world feel real. But without the randomization, playing the game enough times to figure out the Chain would be much more repetitive. The permanence of the Chain and the randomization of the world work in tandem to give you a greater appreciation of the other.

The Joy of Play

I may not enjoy the new Zelda games as much as the older ones, but I still credit the series for giving me one of my best and earliest lessons in game design. Nintendo was so influential to me that even when I criticize their games, it's often by standards that they themselves have set. My worry is that as players we've grown too comfortable with being comfortable. We revel in being consumers of products, rather than contributors to a rapidly-evolving art form. Part of it is how we buy and play games today: without leaving our homes. There's very little need to drive to a game store, since everything from indie games to big budget blockbusters is available for digital download. It's a wonderful convenience that I'd hate to give up, but it does lend itself to an indulgent kind of gaming. We've gone from asking, "How does this game play?" to asking, "Does this game play the way I want it to play?"

We can't have everything that we want all at once, though. We can't know what to expect and also be surprised. We can't be free from frustration and also be challenged. We can't go unchallenged and also feel satisfied with our accomplishments. Mystery, surprise, tension, challenge, and a real sense of accomplishment always come at the cost of feeling uncomfortable. Given the opportunity, many of us would rather take the easier road, but that's usually the less rewarding one.

The best games come out of a mutual respect between the creator and the player. The player does not demand a certain experience from the creator because they trust in the creator's expertise and because they want to be surprised. A personal creative vision cannot bloom without the freedom afforded by that trust. At the same time, creators must trust in the curiosity and abilities of their players. Continuously interrupting play to steer players with direct text messages and other obvious hints not only infantilizes them, but it also reveals the creator's insecurity in their ability to design games.

Part of what made Nintendo such a revolutionary game maker in the 80s was how elegantly they taught players how to play. In "Learning Through Level Design with Mario," Jeremy Parish writes, "In the days before tutorials, these games didn't teach by holding players' hands; rather, they dropped gamers into a live situation and gently guided them through organic level design." Parish then breaks down how the first level of *Super Mario Bros.* reveals each of the game's fundamental

concepts one-by-one, from jumping to entering pipes—all without a single line of text or break in play. Similarly, in *Spelunky*, things like the ghost and the City of Gold aren't in the game just to add thematic flavor, they also exist to guide players toward my vision of how it's meant to be played, in a way that still allows them to gain a sense of genuine accomplishment and cleverness.

The "joy of discovery" is one of the fundamental joys of play itself. Not just the joy of discovering secrets within the game, but also the joy of uncovering the creator's vision. It's that "Aha!" moment where it all makes sense, and behind the world the player can feel the touch of another creative mind. In order for it to be truly joyful, however, it must remain hidden from plain view—not carved as commandments into stone tablets but revealed, piece by piece, through the player's exploration of the game's rules.

The joy of discovery extends to other activities as well. On occasion, my wife and I treat ourselves to a high-end meal where, instead of picking what you want to eat from a menu, you're offered a set menu, or the choice between a few set menus that offer multiple courses at a *prix fixe,* or set price. For these meals, you're putting your trust in the chef's expertise and creativity. In fact, these set menus are also called *table d'hôte* menus, or literally "the host's table" menus, suggesting that you are a guest at their table.

A few times, we've ordered *omakase* at a nice sushi restaurant. In Japanese, omakase translates to "trust the

chef" and ordering in this style means that the chef will design your meal, selecting and preparing dishes that complement one another. The highest quality fish is reserved for omakase, and you can often sit where the chef is preparing the food. It's here that you can get to know the chef and their food personally—aside from casual banter, you're also treated to closer examinations of the ingredients you're eating and how they're prepared.

It's a strangely intimidating experience to eat this way, but when the chef is good it can also be intimate, uplifting, and exciting. By letting them choose the food, you're giving them the freedom to execute their vision with their full creativity and passion, which is then transferred to you as the diner. Once the food is served, you'll want to take your time and explore, savoring how each ingredient tastes with the others. In that context, the idea that you're sitting at "the host's table" makes sense—when done right, you feel a similar connection to the chef and their devotion to their cooking.

On the other hand, consider that for over 40 years the slogan of the fast food chain Burger King was "Have it Your Way."

PART III:
SPELUNKY XBLA

Despite the eeriness of these events,
I remain focused on the task at hand.
Eagerly, I press on, pursuing my fame and fortune.

— Yang

An Opportunity

A month after *Spelunky*'s release on the TIGSource forums, I received an email from another game developer named Jonathan Blow asking about a graphical glitch he was experiencing playing the game on his desktop computer. We hadn't spoken much before this, but I definitely knew who he was: In August of 2008, Jon had released his puzzle platformer *Braid* on Xbox Live Arcade (XBLA), a service where players could download games to play on Microsoft's Xbox 360 console. Like *Aquaria*, *Braid* was mostly a two-person development, with Jon doing everything but the music, which he licensed, and the artwork, which was painted by illustrator and comic book artist David Hellman. *Braid* had an even longer development, however—Jon conceived of it in 2004 and started work in 2005, putting hundreds of thousands of his own dollars into the project. Fortunately, *Braid* was a huge critical and commercial success on XBLA, and Jon, who had already garnered some fame for his talks on game design and development, became a well-respected figure in the industry and an inspiration for other indie developers who were trying to make it.

While I was never able to fix the glitch (sorry, Jon!), the email thread ended up changing my life. All it took was two sentences from Jon: "Spelunky is a really fun game—you should pitch a beefed-up version of

it to XBLA/PSN-type guys, if you have any interest in publishing that way. I would be happy to tell my contact at Microsoft that it's better than 99% of the stuff they publish."

By this point, I had started entertaining the idea of going commercial with *Spelunky*, but as a handheld console game, not a home console game. *Spelunky* seemed like the perfect game to take on the go, since the length was short and the pixel graphics would be at home on smaller screens. But it was hard to pass up an offer like this from Jon, given his success with *Braid* on XBLA. And he wasn't the first indie to strike gold that year—a humorous beat 'em up called *Castle Crashers* that came out a few months earlier was making headlines for selling more than a million copies. On the Wii, a two-man team called 2D Boy released a unique puzzle game called *World of Goo* that was also doing phenomenally. In 2008, the little indie tide pool was starting to make some seriously big waves on consoles.

I still wouldn't have taken Jon's offer if I didn't want to work on the project. No matter how much potential there was on XBLA, I knew that if my heart wasn't in it then the project would either flop or not get finished at all. But my hang-ups from *Aquaria*'s challenging development were mostly gone by 2009. My garden had flourished and I was ready to plant another forest. Even better if the forest grew right out of the garden! Plus, there was something compelling about sitting in front of the television and playing *Spelunky* in high

definition, possibly with friends in a new multiplayer mode (a feature that some fans were already requesting in the freeware game).

Once I let Jon know I was interested, things moved quickly, and pretty soon I was showing the game to Kevin Hathaway, Jon's producer at Microsoft. He liked it: Within a week of emailing Jon, *Spelunky* was being investigated as a potential XBLA title.

The Pitch

Microsoft didn't ask me to do very much while they considered *Spelunky* for XBLA. Jon's vote of confidence was worth a lot and there was a wealth of praise online, from forum threads discussing it to videos of people playing. And the original game, which I later rebranded "Spelunky Classic," was fully playable from beginning to end, so Kevin and his team could play it themselves and visualize how it might be converted into an XBLA game.

One thing Kevin did ask me for was a short design document to outline my plans. To me, *Spelunky* on XBLA represented an opportunity to reach a larger, more mainstream audience and also give fans of the original something to look forward to. To that end, I thought about how to fill out the game's possibility space. Since this was a remake and not a sequel, the original game where you travel through the Mines, the Jungle, the Ice

Caves, and the Temple had to be present, but I imagined that, aside from a visual makeover, there would be plenty of room for some new special areas, items, and monsters.

There was also nothing preventing me from adding some twists on the original's single game mode. I outlined a Challenge Mode and an Endless Mode to accompany "Adventure Mode," my name for the original game mode. Challenge Mode, inspired by levels that players built with *Spelunky Classic*'s in-game level editor, would test players on their knowledge of the game's mechanics in a series of pre-made scenarios. Endless Mode would be a fast-paced version of Adventure Mode, where you raced downward in a single, never-ending level as the ceiling collapses above you.

My hope was that these new modes would make *Spelunky* feel bigger and more enticing to console players. I was also hoping it would get Microsoft excited, and in the design document I wrote "Exclusive to XBLA!" next to these features to drive the point home. At the end, I saved my pièce de résistance: cooperative multiplayer. The other modes weren't given much thought beyond the basic concept, but I spent quite a bit of time figuring out the mechanics of playing *Spelunky* with other players, from the scrolling of the screen to what would happen when a player died. To explain the social aspect, I called it a "party game" that promoted both cooperation and competition.

All in all, this seemed like a reasonable plan. Although I was certainly getting caught up in the excitement of pitching a game to Microsoft, remaking a little

arcade-style game with a few new twists didn't seem that big of a deal, especially compared to the sprawl of both *Aquaria*'s development and its large underwater world. I also stated clearly in my design document that the multiplayer in *Spelunky* would only be offline, since friends had warned me that online multiplayer could double the development time. When pressed, I gave Kevin a "conservative" estimate of about a year to finish the game.

Andy

One of Microsoft's concerns was how I planned on singlehandedly completing an Xbox 360 project within my one-year timeframe. *Spelunky Classic* was made in Game Maker and was not easily portable to consoles at the time. Jon Blow had generously offered to let me build my remake off the *Braid* source code, so it seemed natural to ask him if he was interested in helping me with the programming, but he was already working on his next project, a first-person puzzle game called *The Witness*. I thought of other programmers I knew and sent out emails. All of them were working on other games.

In what can only be described as an act of sheer hubris, I told Microsoft that if it came down to it, I would program the game myself since I had done the programming in Game Maker and had a computer science degree and everything. In hindsight, this was preposterous, but at the time, my confidence was

enough to convince Microsoft that they could take a chance on *Spelunky* by greenlighting it.

During this period, I was talking regularly with my friend Andy Hull, who I met in the Klik & Play community. The Klik community was its own ecosystem—the people who made the games were the same ones who played them, and the same ones who reviewed and criticized them. It had its heyday from the mid-90s to the early 2000s, and in a lot of ways it was a nascent version of the indie game community. Most games were developed by a single person who was responsible for the art, audio, programming, and marketing. On "The Wall," a popular Klik & Play message board, overly enthusiastic advertisements for the latest titles would be posted in the middle of heated arguments about game design and technical discussions about the software.

Andy was an active member of the community who made games, posted frequently on the Wall, and also hosted a podcast called "The Garbage Men" where he and his co-host roasted some of the "less good" Klik games that came out. One of his releases, a short one-level demo called *Golden Monkey Strike,* was a favorite of mine, and by coincidence it had a cave-exploring theme similar to *Spelunky*.[6]

6 In fact, the Golden Monkey that starred in his game would end up being a secret character in the *Spelunky* remake. Sacrifice a golden idol on one of Kali's altars to make the monkey appear and begin pooping out gold and gems.

Aside from Jon Perry, who had been a friend and classmate of mine since the second grade, Andy was the only other person from the Klik community that I kept in touch with after we drifted away from that scene. Although we wouldn't meet in person until 2004, Andy and I updated one another through periodic telephone calls where we'd discuss our games and careers. I followed him through college, his first game programming job as an intern working on an Xbox title, and his five years as a wooden toy designer at Melissa & Doug. I even attended his wedding—only my second time meeting him in person!

Once Andy offered to help me with *Spelunky*'s programming, the partnership made perfect sense. Although Jon Blow and the other people I had asked before were cool guys and far more experienced coders than Andy, I didn't know them nearly as well, and whether someone works well with you can be more critical than how good they are at what they do. If we were speaking in terms of a fantasy role-playing game, I'd be more than happy to trade a few points of programming skill for a few points of team synergy and comfortability. A teammate is someone you'll be spending a lot of time with, who you'll have to make important decisions with, and who you'll be making money with. Nothing sucks more than having to abandon a project halfway through because you aren't on the same wavelength as your partners.

I often compare the process of finding and working with teammates to dating. In any big project, you're not just looking for a set of artistic and technical skills to fit your own, you're also looking for someone who shares your creative vision, who communicates well, and who will be as passionate and dedicated about the project as you are in the long run. Even after the game is released, your relationship will still matter, since you'll be dealing with marketing, player feedback, patches, ports, and other opportunities. Ultimately, if you're planning on releasing a commercial video game, you are looking for "marriage material"—a committed, stable partner you can get along with for a long time.

You wouldn't want to work with someone who was incompetent at their job, of course, but Andy was far from that. While we were in college, I tested the game projects he made for school and knew beyond a doubt that he was a much better programmer than me. Even his relative inexperience with coding an Xbox 360 game came as a relief, because it made me feel more comfortable talking to him about the technical aspects of the development. Instead of the code being a mysterious black box that only the programmer understood, it was something that Andy and I could learn together.

I never worried about Andy and me not working out. Even though we hadn't met in person many times before, I'd known him to be a dependable friend and someone who worked hard, but at a level I could keep up with. And our involvement in the Klik & Play

community also added a layer of comfort the way it might have if we had gone to the same school together.

As it turned out, it was good that Andy and I got along so well, because *Spelunky*'s development and Andy's role in it would both end up being far bigger than either of us imagined.

Explaining the Vision

There was some culture shock working with a large company like Microsoft. Before that partnership, I made games in two steps: Step One, work on the game. Step Two, release the game. I had written design documents and set milestones in the past, but it was a casual, organic process. Things were done when they made sense and abandoned as soon they no longer did. Now it felt like we had to chisel our plans into stone slabs on a regular basis. We weren't just creating our own process for making a game, we were also part of someone else's process.

Thankfully, we had Kevin, our producer, to guide us. When you're working with a big company, a small team of managers, test leads, and marketers inside that company acts as a kind of membrane to protect you from the enormity of the rest of the organization. And the most important person on that team is your producer. In addition to *Braid*, Kevin was also the producer for *Super Meat Boy* and *Castle Crashers*, making him an indie hit-maker of sorts at Microsoft. After our IGF win

in 2007, Alec and I nearly signed a deal with a different producer to put *Aquaria* on XBLA, but it fell through, in large part because we sensed that the producer didn't respect the game. Kevin was different. Speaking with him, you knew that he cared about the games he worked on and the people who worked on them.

Kevin was also straightforward about his opinions and not afraid to share his ideas on how to improve things. He pushed us to think in new directions, but relented when it was clear we weren't interested. There's a very fine line between producer and meddler. If your producer is too hands-off that's a problem, since there's so much to keep track of during console development. Kevin seemed to understand what our strengths and weaknesses were, reminding us about things we were weak on and hanging back when it came to our strengths. I imagine that one of the most difficult things about working with tiny developers is that they're each so different in terms of style, personality, and work habits. Kevin's empathy and his flexibility are what made him a great producer and garnered him so much respect from the indies he worked with on XBLA.

One thing that he brought up to me early and often was the importance of bringing in new players and explaining to them what was so cool about *Spelunky*. It was a daunting task. Not that I didn't think the game was cool, of course—I wouldn't have made it if I didn't like it. I just didn't know how to put that into words.

In the documentary *The Great Directors*, David Lynch says, "As soon as you finish a film, people want you to talk about it, and, um, the film is the talking. The film is the thing, so you go see the film, that's the thing. It's a whole thing, and it's there, and that's it!" That's how I felt about *Spelunky*.

Before I started working with Microsoft, I'd largely gotten away with not having to explain the game much at all. A big reason for that was how it began: as a freeware experiment. Without the pressure of having to make sales, I was content to let *Spelunky* speak for itself, however obtuse its language might be for new players. Those first few releases had the added benefit of going into the hands of game developers and indie game enthusiasts who, by and large, were eager to tear apart a challenging new game and work out its rules and secrets by themselves. Those initial beta testers became the first ambassadors for the game, helping to convert new players on TIGForums, and those new converts converted other players, and so on, until the game spread to other forums and blogs. The fact that *Spelunky* was free made it easy to try it to see what all the fuss was about, and if players didn't like it, many of them were still inclined to express their opinion, which only increased its appeal.

A lot of people, including Andy, have told me that they tried *Spelunky* and gave up, only to come back later because they kept hearing other people rave about it. *Spelunky* fits the profile of an "iceberg game"— it's easy to stop playing before you figure out that there's more to

it than randomly-generated death traps. It's as though the game requires a friend to tap you on the shoulder and say, "Hey, look again. You may have missed something." On the second time around, people begin discovering the nuances of the rules, the secret content, and the largeness of the game's possibility space.

On Xbox, however, we couldn't depend on word of mouth. Or rather, we couldn't depend on the same word of mouth that had brought *Spelunky* this far. Until that point, *Spelunky* was promoted largely by game developers and indie gaming communities far removed from your average Halo player on Xbox 360. We didn't want to assume that Xbox users would know anything about *Spelunky* when they first encountered the game and we couldn't count on them to be as patient, persistent, and proactive about figuring out a difficult 2D platformer that saved very little of your progress from run to run. Kevin also reminded us that they might not even realize that the levels were randomly-generated. He wanted the game to be as accommodating to new players as we could possibly make it.

Unfortunately, *Spelunky*'s core ideals defy explanation to new players, because the game's indifference to them is so much of its appeal. I thought about how joyfully the players on TIGSource shared their discoveries with one another, and how much they owned their experiences because the discoveries were genuine. So I was skeptical of Kevin's initial ideas, which revolved around rewarding the player in a more permanent way

after each run. He suggested, for example, adding a simple leveling system whereby the player could earn experience each time they played. He argued that this would give new players the incentive to keep playing until they understood the game well enough to play it for its own sake. Another XBLA manager I was working with at the time suggested something else along those lines: a system whereby the player could keep track of secrets that they had discovered, with new secrets being displayed at the end of each run. I worried that these kinds of "micro-rewards," as Kevin put it, would remove some of the player's natural motivation for playing the game and turn it into a grind to obtain experience points and collectibles.

As luck would have it, during this discussion another game developer, Chris Hecker, published an article titled "Achievements Considered Harmful?" that perfectly summed up my fears about these types of systems. Based on a talk he gave at GDC, Hecker's article hypothesized that "tangible, expected, contingent rewards reduce free-choice intrinsic motivation, and verbal, unexpected, informational feedback increases free-choice and self-reported intrinsic motivation." In other words, giving people superficial rewards for already-fun tasks takes some of that fun away.

Kevin argued that the systems he suggested would feed back into the intended experience, since they were just another way to track the player's accomplishments. To him, an experience system was just a way for the

player to gauge their own ability in a more concrete fashion. He suggested a logbook or adventure journal as a place to put them.

I don't have a problem with statistics because they feel more like interesting facts rather than tangible rewards, but I worried that an experience system would put a number on something that was previously unquantifiable. And now that the game was in production, I was even having doubts about the achievements I had pitched in my initial design document. To me, it was the difference between telling a kid how tall her sandcastles are versus giving her a cookie every time she builds one. The former is a fun statistic, whereas the latter is an unnecessary reward that may motivate her to build sandcastles for cookies rather than for its own sake.

It may come down to which players you want to cater to. To carry the sandcastle analogy further, is it more important to get the kids who don't want to build a sandcastle to build one, or is it more important that the kids who already love sandcastles enjoy their hobby as much as possible?

The idea of an adventure journal appealed to me, though. We already had a player profile that collected certain stats that were available in the original game, like how many times the player has played versus the number of times they've died. I felt like we could put the profile into this journal and use it, if not to track experience points, then to track other things that might be interesting to the player and motivate them less

superficially. In an email I wrote to Kevin a couple of months later, I outlined my plan for how to draw in players early on:

1. LOBBY

   It starts with the lobby, after the players select their character. We'll close off the main entrance to the Mines, and have the player pick up a journal that's sitting there. The first journal entry reveals that its previous owner was an old explorer. Once you read through the introduction, you'll be taken into a sepia-toned "flashback," which lets you play as the old explorer in a single level. This will act as the game's new tutorial.

2. TUTORIAL

   In the tutorial you'll "collect" new journal entries, which will teach you how to play the game. We'll try to make the tutorial a little bit nonlinear, and more interesting, but you'll have the immediate goal of finding the journal entries. The old explorer's last entry will tell you to keep his journal and write your own entries in it. When you exit the tutorial, it'll take you back to the lobby, and the door to the mines will open so you can start the adventure.

3. JOURNAL ENTRIES

I think we will try putting in journal entries for the various enemies, traps, items, and places that you encounter in the game. You'll get a little notification when you receive a new entry, and you can press a button (Back?) to open the entry. I think we can do it in a way that's not too intrusive, but gives new (and old) players something fun to look at and will keep them exploring (plus the entries can hint at some of the possibilities of the game). It'll be tied into the tutorial/flashback so it won't be jarring.

4. PROGRESSION

Try to do more to show the game's progression. Have it say "Level X" in the corner of the screen when you enter a new level. Make it obvious in the journal that you want to go deeper, that your goal in each level is the exit door, that there is more than one area, etc. Also, we can put a map on the pause screen that shows you how deep you are in the caves.[7]

7 This map feature never felt necessary after we added the journal and didn't make it into the game.

5. GAME OVER

> On the game over, we'll open up the journal again, and use it not only to show you your game stats, but also encourage the player and get the idea across that dying is something fun and shouldn't feel frustrating.

Once again, one idea was solving many problems. The journal would end up acting not only as a place for new players to track their progress, but also as a pause menu, companion, teacher, and narrative device.

I decided that the journal would belong to Yang, a seasoned and salty explorer who had plunged into the caves for the last time, leaving his journal behind for some enterprising new upstart. Yang not only acts as a stand-in for the friend who taps you on your shoulder, but he's also the type of adventurer that I wanted players to aspire to be: In between tutorial levels he explains through the journal how, led by his curiosity and perseverance, he shrugged off death after death, slowly mastering the fundamental skills he needed to progress deeper into the caves. Kevin also wondered if we couldn't use the game's narrative to explain why the character is able to die again and again, so Yang hypothesizes that it might be "the legendary Curse of Olmec, the one they say traps people inside the caves forever!"

How much did Yang's journal help to draw in new players? It's hard to say. I know that for some people,

it didn't. When the wife of another game developer couldn't get past the tutorial, she asked him if he was disappointed in her. He counted that as one of his worst experiences with a video game.

But that's the thing—since each person approaches a game with their own biases, preconceptions, and expectations, it's impossible to please them all. With *Spelunky*, I was only willing to go so far to offer new players a hand. Beyond that point, I felt like it would hurt the experience of players who were more eager for a challenge. And when you have to choose between those two groups, it's not a hard choice at all.

New Engine

You know that scene in movies where an average person is trapped with a bomb and a bomb expert has to talk them through disarming it over the phone? As the timer ticks down, the civilian—sweating profusely, hands shaking —tries to figure out which of the wires the expert is telling them to cut. That's kind of what it felt like for me trying to get the *Braid* engine running on my Xbox 360 development kit while waiting for Andy, thousands of miles away and by far the better programmer, to obtain one of his own. Although I didn't have much trouble putting a rudimentary version of *Spelunky*'s level generation into the Windows version of *Braid*, I couldn't get *Braid* to

run on the devkit for weeks. It would crash immediately without any error messages.

The time difference from San Francisco, where I lived, to Connecticut, where Andy lived, was only three hours, but because I was going to bed in the early hours of the morning and waking up in the afternoon, we only had a few hours where we were awake together. During this time, I'd explain to Andy over the phone what was happening (or not happening) with the devkit and he would offer suggestions on how to fix things. When I went to bed, Andy would continue working on the Windows build.

Jon Blow had already gone above and beyond by lending us the *Braid* source code and wasn't to blame for any of the problems we had. Instead, it was our inexperience that made it difficult to decipher the inner workings of the engine, which was never meant to be repurposed for another title. *Braid* was also released on XBLA in August of 2008, and it was now November of 2009. Over a year had passed since Jon had looked at his Xbox code—long enough to forget why certain things were done the way they were, and long enough for the Xbox 360 development software to change significantly. So even when we asked Jon for help, he didn't always have the answers at hand.

One night, when Andy was in town for the Game Developers Conference, we invited one of Andy's friends, a seasoned game engine programmer, to come back to my apartment and help us figure out why

our test programs were running slowly. After turning the devkit on, he quickly noted that it was running on severely outdated firmware, an important piece of embedded software that instructs the devkit. Once we flashed the devkit to replace the old firmware with the latest update, things ran a lot better, but it drove home to Andy and me how much we had to learn.

Eventually we got *Braid* running on the Xbox and began converting it to *Spelunky*, but it became clearer to me that our small gains were not mounting quickly enough. My biggest fear was that we would continue along this path for months, only to run into serious technical problems we couldn't easily fix because we didn't understand all of the code. Microsoft's certification process for XBLA games was particularly threatening in that regard. Certification, also known as "cert," is like the final exam to ensure that your game meets the standards for release. On consoles, cert had a reputation for being particularly stringent, long, and costly, with some games being rejected multiple times for small infractions. The idea of rolling into that process with a half-baked Frankenstein monster of a game engine made ditching *Braid* and writing our own engine seem much more appealing.

Writing a game engine from scratch can take years on its own, so it's not a decision to be made lightly. But for us, it was the right choice—not because we thought we could write a better engine than *Braid*'s, but because we needed a *Spelunky* engine that we understood from the

bottom up. Once I made the decision, I felt instantly relieved. As challenging as it would be to write an Xbox 360 game from scratch, it seemed a lot less challenging than untangling someone else's code.

And from that point on, Andy became much more than someone who was helping me with the programming on *Spelunky*—he became *Spelunky's* lead programmer.

Improvements

The game we were now making was not a port of *Spelunky Classic*, but a remake—or, as I called it in the initial design document I sent Microsoft, an "upgrade." In my head, however, I felt as though I was really making a "fan game" of the original *Spelunky*. Real fan games aren't made by the original developers, of course. They're games made by fans that act as unofficial sequels, remakes, or remixes of existing intellectual property. High-level fan games can take large teams years to make and some come close to matching the quality of the titles that inspired them.

It might seem strange to spend so much time and effort on "just a fan game." If you're creative enough to make a professional-grade fan game, why not create your own original game instead? Particularly since the threat of legal action would no longer be hanging overhead.

The reason I think fan games are so popular is that creating an original idea and expanding upon one are such different creative acts. There's a deep-seated uneasiness about confronting an empty canvas—after every failure, you might end up back at square zero. It takes a lot of failing and false starts to come up with something that works, as shown by the numerous pre-*Spelunky* prototypes I left behind.

Reimagining an existing game is carefree, though. You're starting with something tangible that you know works and that you can fall back on when you hit a dead end. That's partly why even professional artists who have created their own worlds enjoy making fan art. It offers the same kind of bliss as lying in the grass and pointing out what clouds remind you of. I was fortunate in that when I was doing the stressful part of creating *Spelunky*—developing the initial "Big Bang" of the concept with lots of prototyping—I was shielded from the pressures of making a commercial game. Then, when I was making another commercial game, I was at least free of the pressures of creating an original idea.

I was eager to approach the problem of reimagining the original *Spelunky*. There was never any question for me that I'd be changing the "pixel art" graphics. I love the way pixel art looks, but to your average person it simply looks old. And technically, it is: Pixel art was originally a product of hardware limitations, not a style in and of itself. Although modern pixel artists still enjoy the challenge of overcoming those limitations, it didn't

seem appropriate for *Spelunky*'s debut on HD televisions around the world.

Another thing I wanted to change was the movement. In *Spelunky Classic*, Spelunky Guy has a floaty jump and slippery run that took time to learn how to control. This was exacerbated by the fact that the game is locked to 30 frames per second (FPS), the default setting of Game Maker 8.1. By the time I realized this, changing the FPS would have required large rewrites to the existing code. As a result, *Spelunky Classic* has some extra choppiness compared to the 60 FPS remake.

I didn't want the player's controls to be another hurdle toward getting into the game, so Andy spent a lot of time tweaking them to feel smooth. Since jumping was such an important part of *Spelunky*, we made it so that you can still jump for a few extra steps off the side of a ledge, to reduce the occurrence of players falling into pits and wondering why their jump hadn't registered (a common frustration of everyone who has ever played platformers). This also made it easier for you to make some tricky jumps, like when your target platform is directly above you.

Thankfully, giving the player more control didn't make *Spelunky* easier to beat, since the enemies received a number of upgrades, including faster movement. I also added new enemies and more difficult versions of the old enemies. For example, the spitting cobras in the Mines are an upgrade to the snakes from the original. Also, unlike *Spelunky Classic*, where all four levels of the

Mines are identical, I created three tiers of difficulty that determine how often monsters appear. As you move from the first level of an area to the fourth, you'll encounter more monsters. Some difficult monsters, like the cobra, may only appear after the first level of an area.

You may recall that when I explained monster generation earlier, I gave a snippet of code from the Jungle that looked like this:

```
if (rand(1,60) == 1)
instance_create(x, y-16, oManTrap);

else if (rand(1,60) == 1)
instance_create(x, y-16, oCaveman);

else if (rand(1,120) == 1)
instance_create(x, y-16, oFireFrog);

else if (rand(1,30) == 1)
instance_create(x, y-16, oFrog);
```

That same block of code looks like this in the remake (altered slightly to make it easier to understand):

```
if (rand_monster(70, 60, 50))
{
    create_entity(i, j, MONS_MANTRAP);
}
```

```
else if (rand_monster(60, 40, 30))
{
    create_entity(i, j, MONS_CAVEMAN);
}
else if (rand_monster(120, 50, 40))
{
    create_entity(i, j, MONS_TIKIMAN);
}
else if (rand_monster(240, 70, 50))
{
    create_entity(i, j, MONS_SNAIL);
}
else if (rand_monster(120, 120, 120))
{
    create_entity(i, j, MONS_FIREFROG);
}
else if (rand_monster(40, 20, 20))
{
    create_entity(i, j, MONS_FROG);
}
else if (rand_monster(30, 30, 30))
{
    create_entity(i, j, MONS_CRITTERFROG);
}
```

The function "rand_monster" takes in three parameters that determine the chance of the monster appearing in the first level, the middle two levels, and the last level. As you can see, the mantrap has an increasingly high chance of appearing in the Jungle the further you progress, with a 1/70 chance of appearing above a floor tile in level 1, a 1/60 chance in levels 2 and 3, and a 1/50 chance of appearing in level 4. You can pull some other interesting tidbits from looking at this code, like the fact that tiki men and snails (both monsters that are new to the remake) are significantly more likely to appear after the first level, but fire frogs are equally rare across the board.

In designing new monsters, I not only wanted to add more complex interactions and challenges, but I also wanted to make the world feel more alive. The human mind excels at finding patterns and making connections, so just adding a few variations on existing monsters gives the impression that they have families. When you see that a tiki man is really a caveman with a mask and a boomerang, you start to wonder if they belong to a tribe where the tiki men are high-ranking members. Other patterns quickly play into that notion, like the fact that tiki men are rarer and only appear in the Jungle, whereas cavemen roam around in other areas, too. Tiki men are already more difficult monsters than cavemen, but these little details cement their elite status and therefore their place in *Spelunky*'s world.

This "alive" quality doesn't depend so much on the complexity of each monster's artificial intelligence. Take, for example, the way players associate the four ghosts in *Pac-Man* with aggression, ambush, capriciousness, and stupidity. In reality, the behavior of the ghosts is determined by simple algorithms that target different tiles to move to based on each character's position. When these behaviors play out together on the same map, however, it seems as though the ghosts are working together, with Blinky chasing Pac-Man into the ambush of Pinky and so on.

I applied a similar principle to *Spelunky*: Combine simple behaviors to give the impression that the monsters are working together. This not only creates challenging situations, but it also makes the world feel more like a living, breathing ecosystem. Wherever possible, I tried to add monsters that attack you from new directions, so that when they were paired with existing monsters the attacks would feel coordinated. In the Jungle, for example, the snail blows acid bubbles that float upward—you can often see these floating up from gaps in the floor, making gaps more dangerous to jump over or into. And in the Ice Caves, the woolly mammoth spits a horizontally-moving freeze shot that creates a cross fire with the UFO's downward-moving projectile. Both the snail's bubble and mammoth's blast are also used elsewhere in the game—the bubble floats up from acid pools in the hidden Worm level and the freeze shot can be fired from a freeze ray the player can

acquire from a shop. Giving its inhabitants a shared language is not just a time-saving measure, but another way to make the world feel cohesive.

One of my favorite *Spelunky* stories is a great showcase of how simple behaviors can lead to complex moments. It comes from Tom Francis, a former editor of *PC Gamer* and now a fellow game developer, about the time Tom was in the Black Market and watched a tiki man wander empty-handed into one of the shops.

Tom sees the tiki man pick up a boomerang that's for sale in the shop. "For a split second, I am amused," he writes. "He's going to buy a new boomerang! Silly tribesman, you don't own material wealth!" But then it dawns on him that something terrible is about to happen, and he scrambles away to a safe corner to avoid what he calls the "shopstorm":

> All nine shopkeepers hurl themselves into the air and start firing their shotguns in random directions. They kill the tribesman. They kill two other tribesmen. They kill frogs, pitchers, snails. One kills the slave he was selling, another kills his own dog. Two of them throw themselves to their deaths in the excitement. Four of them throw themselves into a pit, where their bursts of buckshot cut each other to ribbons.
>
> When the blasts quiet down, I crawl slowly out of my hiding place and walk carefully through the empty shops, collecting everything for free.

> What happened here was: the tribesman walked out of the shop. He walked out of the shop with the shopkeeper's boomerang in his hand, and he walked out of the shop without paying for it.

What makes Tom's story so cool is that a number of disparate systems came together in just the right way to write its plot. First, a tiki man had to lose his boomerang, either by getting hit by something (like a tiki trap) or by throwing it at Tom. Then the tiki man had to walk into one of the Black Market's shops. Finally, the shop had to be selling a boomerang. What happened next was the tiki man's "pick up a boomerang and walk away with it" system met with the shopkeeper's "get incredibly angry if an unpaid item leaves the shop" system and all hell broke loose. Tom's thorough understanding of these systems takes nothing away from his fascination with it. Actually, the opposite is true—his understanding only deepens his enjoyment.

Traditional, linear, static narratives will always be important in video games, but these personal tales are the ones that, to me, separate the gaming experience from other art forms. Unlike fan fiction, where the audience is creating stories based on their favorite worlds, game players are living them. And in the context of these stories, the random number generator is akin to Fate, setting up scenarios that at times seem too incredible to be anything but scripted. No wonder game players

are known to send prayers and curses to the "Random Number God" or "RNGesus."[8]

It's easy to add more and more things to a randomized game and rely on what I think of as the "free value" that randomization offers. In a game with randomized levels, people will keep playing just to see what comes up. But to make it more than a glorified slot machine requires putting together a collection of systems and rules that is worth understanding, behind a world that feels interconnected. If you can manage that, a simple story of rescuing a troll from a pit and making it your friend, an "EPIC MOMENT" when a spider was killed by an arrow trap, or a deadly shopstorm becomes something as meaningful and memorable as hours of dialogue and scripted cutscenes. Sometimes more so.

Eirik

The *Spelunky Classic* soundtrack was composed by George Buzinkai, whom I knew from the TIGSource forums, and my old friend Jon Perry, who had become an accomplished musician since we released *Eternal Daughter*. After I had already enlisted George and Jon's services for the soundtrack, I was approached by a Norwegian composer named Eirik Suhrke who also

8 Pronounced "AR-EN-JESUS."

co-ran an independent music label and community called Pause. With my music needs already fulfilled for *Spelunky Classic*, Eirik never got to write anything for it, but when he approached me again for *Spelunky* on XBLA, I was all ears.

Because the art style of *Spelunky Classic* was low-resolution pixel art, I had requested tracks that sounded similar to music on the NES. For the remake, however, I was planning on using the high-definition painted style that I had developed for *Aquaria*, and needed a new soundtrack to match it. In an email to Eirik describing my vision for the game's music, I wrote:

> To make the music fit the game, I feel it needs to be grounded in some classic retro tunes, but modernized a bit and with its own unique sound. One thing I would like to try with *Spelunky XBLA* is to make the music more dynamic. I'm not sure what the best way to do this is… perhaps depending on different situations the game could fade in different tracks/parts of a track? Or speed it up/slow it down? Both? Something else entirely? e.g. When the player grabs an idol the music might change to something more fast-paced and exciting. Or when you grab the damsel, maybe she has her own sound that fits her but also works with the current level. How do we give each area its own unique theme and

also do something to it to make it dynamic and fit what's going on in the game?

Ideally, I'd like the musician/sound guy for the game to think about this problem and try to figure out some unique ways of approaching it. It's a bit more complex than simply writing a track for each level, but the results should be worth it! Since people are likely to play 1000's of times, I want the music to be as refreshing each time as the levels are.

Over the course of the development, we scaled back the dynamic music idea and changed the music in only a few key places, like when the shopkeepers were angered. Having the music shift dynamically without being annoying was an avenue that we didn't have the resources to explore—it was hard enough simply filling the game with good music.

Eirik's approach to *Spelunky*'s soundtrack was to mix styles and influences, in the same way that the design of the game combined roguelikes and platformers, and the same way that the graphics combined painted artwork with tiles on a grid. He explained his intentions in a post on the *Spelunky* website:

> When I first started out I decided to go with a mix of FM synthesis,[9] sample-based tracker music and live band. In my head I imagine a triangle, where each angle represents one of the above. The tracks in this score vary in sound, but all can be placed somewhere in said diagram. Some tracks are all FM, some are all band, etc., but most are a combination of the three. My reasoning for going with those three styles lies in the fact that *Spelunky* feels, to me, like an early 90s computer game, with an up-to-date presentation. The FM part is inspired by old arcade games and old computer sound chips such as the AdLib. The tracker music is also synonymous with 90s computer games, especially the Amiga and the demo scene. I added in a live band to make it work with the hand drawn, HD graphics.

At the time, it was popular for arcade-style indie games to employ simple, catchy, high-energy "chiptunes," electronic music that's so-named either because it's made using the sound chips found in old video game consoles or because it emulates that sound. That retro sound seemed like a bad fit for *Spelunky*, though. We needed something that captured the game's

9 FM synthesis is a technique for creating digital music that was popularized by the Sega Mega Drive, also known as the Sega Genesis in the U.S.

unpredictable nature and its eclectic influences: not just console and arcade games but also roguelikes and pulp adventure movies. We also had to consider that the player was going to be starting levels over and over again in quick succession—simple, catchy melodies that might work in other games could drive someone crazy in *Spelunky*. (*Spelunky Classic*'s soundtrack had already garnered that criticism.) It seemed like a multi-layered and unique game required a similar soundscape.

Unfortunately, in early builds that I sent to other indie game developers, the music, in particular the music for the Mines, met with disapproval. The FM synth and jazzy chords gave the tracks that played in that area a strong 80s movie vibe that my friends felt were out of place. In general, the Mines were a troublesome area to compose for because we had a hard time nailing down what its overall tone should be as the first area in *Spelunky*. Should it be light-hearted to match the game's optimism and cartoony graphics or more brooding to match the feeling of being inside a dangerous mine? Because of these problems, the Mines tracks were the first ones that Eirik worked on and also the last.

Later, when we released our first Xbox trailer, the music received more criticism. For the trailer, which highlighted the changes from the low-resolution original game to the high-definition remake, Andy and I chose the track Eirik wrote for the game's ending, a cheerful song that incorporated real saxophone. We picked this particular song in part because the other tracks were

short and designed to loop repeatedly. The ending song, which played once as the credits rolled, had a kind of "narrative" that we could fit gameplay clips to. And the song's resemblance to a television sitcom's theme song seemed to go well with the way we were showing off the game's cooperative multiplayer.

In retrospect, this was a bad idea. It was the first time that the outside world had heard any of the new music. As far as listeners could tell, our trailer music was representative of the entire game, and for fans of the original, it went violently against their expectations. Although *Spelunky Classic*'s soundtrack was once criticized as being too repetitive, it seemed that fans had grown fond of it over time. Many of these fans compared the new trailer's music to it unfavorably.[10]

To Eirik's credit, he took it all in stride. For the rest of development, we schemed together on ways to satisfy fans without abandoning our vision. Eirik pulled back on the controversial saxophone and jazzy chords, especially in the early areas where players would spend the majority of their time. In the later areas and the special levels, Eirik felt free to let loose and be a little weirder.

Spelunky's final soundtrack, which totals 47 separate tracks, is much like how I envisioned the game itself to be: rich, challenging, and uncompromising. And like the game, it faced an uphill battle meeting the

10 There was a similar reaction to my new artwork as compared to *Spelunky Classic*'s pixel art.

expectations of players old and new, but slowly won over many diehard fans as time went on.

We only ever made one change to the soundtrack after the release. I asked Eirik if we could, as part of a regular update, remove a track called "Adventure Begins" that plays once, and only once, when you first enter the Mines. It was an upbeat track that was meant to instill in the player a sense of optimism before the first of many deaths occurs. It was also a track that drew a lot of criticism when *Spelunky* first came out, and although Eirik felt like the change might be "invasive," he agreed that the song clashed slightly with the rest of the soundtrack and was willing to take it out.

When we did take it out in the update, however, the response to the change was only negative. One player asked:

> Was that actually requested? I LOVED that track. It always got me pumped the first time I'd play. I'm sad now :(

We restored it in the next update.

Finishing a Game

In September 2010, I wrote an article for my game-making blog called "Finishing a Game." It's a strategy guide of sorts for making a video game, although it could

probably be applied to any creative endeavor. The thesis of the article is that finishing is a skill as much as being able to design, draw, program, or make music, and that finished projects are more valuable than unfinished projects. Most creative people are familiar with the first part of making something, and it's easy to mistakenly assume that the rest is just more of the same. It's akin to repeatedly climbing the first quarter of a mountain and thinking that you're getting the experience you need to summit. Or running a few miles and thinking that you can run a marathon. In truth, the only way to learn how to summit mountains, run marathons, and finish making games is to actually do those things.

The article consisted of fifteen tips on how to finish a game:

1. Choose an idea for a game that satisfies three requirements: it's a game that you want to make, a game that you are good at making, and a game you will wish you had made.
2. Actually start the damn game. Writing design documents and planning doesn't count.
3. Don't roll your own tech if you don't have to.
4. Prototype.
5. Make sure the core mechanics are fun.
6. Choose good partners (or work alone as long as you can).
7. Grind is normal, so factor it into your plan.

8. Use awards, competitions, and other events as real deadlines.
9. Push forward. Don't get hung up on one part of the game for too long.
10. Take care of your mental and physical health.
11. Stop making excuses for starting over. Halfway through any project you'll always feel like you can do a better job if you started over. This cycle will never end until you put a stop to it.
12. Save it for the next game. Do you have a brilliant new idea that requires extensive changes or pushes back your deadline? Save it for another game.
13. Being behind schedule is a great excuse to cut unnecessary content.
14. If you do quit, scale down, not up.
15. Remember that the last 10% is the last 90%. You're never as close to being finished as you think you are.

I'm obsessed with finishing as a skill. Over the years, I've realized that so many of the good things that have come my way are because I was able to finish what I started. *Trigger Happy*, my first released game, was small and rudimentary, but it was a significant step forward and gave me the motivation, confidence, and experience to keep releasing. From there, Jon Perry and I released several other small games with Klik & Play, and these led to *Eternal Daughter*, which led to *Aquaria*, which led to *Spelunky*. Gardens and forests. Irrespective of how

big the project was, each one I finished gave something back to me, whether it was new fans, a new benchmark for what I could accomplish, or new friends that I could work with and learn from.

Along the way, of course, are the discards of many aborted projects and failed prototypes, and while my finished games wouldn't exist without these unfinished ones, I also gained less from them in comparison. Mostly, I found them good for flushing out bad ideas and practicing my programming. Some of them also had parts that were interesting and made it into future games that I released. But they never taught me how to finish what I started or gave me the satisfaction that *Trigger Happy* gave me.

Eternal Daughter was the first big game that I finished. When Jon Perry and I released it in 2002, it was an early example of an indie "Metroidvania"—a nonlinear 2D platformer that took place in a large, interconnected fantasy world.[11] After the game came out, there was the usual grumbling in the Klik community about it being overhyped, followed by promises of "*Eternal Daughter* killers": games inspired by *ED* that would eclipse it. And playing the demos, I had to agree that some of them looked amazing. On the merit of their individual

11 The name "Metroidvania" is a portmanteau of "Metroid" and "Castlevania," the two game series that pioneered the genre.

parts—graphics, music, and technical prowess—they did outclass *Eternal Daughter*.

But those demos never became anything more than demos. And not for laziness on the part of the creators, who worked on the games for at least as long, and often longer, than the two years Jon and I spend on *ED*. But what I saw happen in almost every one of those projects was that once or twice a year a new, shinier version of the same demo would come out. The graphics would be a little better. There would be some fancy new effects or a new battle system. Sometimes a new area of the game would be showcased. But as the quality of the demo went up, the potential for the full game to be released went down. There seemed to be a strong correlation between the number of times a person started over and the likelihood that the project would never be completed.

The reason why people restart projects becomes obvious once you feel the temptation yourself. At some point in any project, you will look back on everything you've done and feel like you could do it all better. Perhaps the code seems like a tangled mess. Or maybe the first area you worked on is not as pretty as the second area because your drawing skills improved along the way. Not at all coincidentally, this is also around the time that you start to see the vastness of the rest of the project spreading out before you like a wasteland. The one level you've made needs to be twenty, and mundane-yet-important things like menu and dialogue systems need to be designed and implemented. It's easy, at this moment, to think that if

you started over and did things the right way that the wasteland might not be a wasteland but a verdant plain you can stroll across to the finish line.

Inevitably, you will hit that same wasteland again no matter how many times you start over, and sadly, it's the talented creators that get hit the hardest by this, because they're the ones that are most critical of their own work. The people working on the "*Eternal Daughter* killers" weren't amateurs. They were phenomenal artists, designers, and programmers who just happened to suck at finishing projects. They obviously learned a lot from reiterating on the same early sections, but they never learned how to navigate that wasteland or overcome that consummate destroyer of dreams: being too much of a perfectionist. As a result, they never learned that with each game you finish, it gets easier to finish the next one, because it feels so good to release something that it'll motivate you to do it again and again.

When I wrote "Finishing a Game," Andy and I were working hard on getting *Spelunky* done, so it was as much a reminder to myself as it was a lesson for others. I thought about *Spelunky Classic* and how naturally that led into this Xbox version—it would have been so much harder if I had tried to rebuild *Spelunky Classic* over and over again until it came out as the remake.

Some people are also natural tool builders instead of game builders. For them, the game is the reason for building game engines and tools, not the other way around, and their ultimate dream is to build engines

and tools that are so efficient, so optimized, and so friendly that the game practically builds itself. To them, the engine itself is a work of art, too, and I'm inclined to agree. In practice, though, it's easy for someone like this to noodle on their game engine ad infinitum.

With *Spelunky*, we wrote almost none of our own tools to help us with development, opting to trade future efficiency for the time saved not writing tools. That's not something I recommend for everyone, but for us, it worked. In *Spelunky Classic*, the level generation was coded directly into Game Maker using its built-in Script Editor, and for XBLA, I ported that code into Visual Studio. Because Game Maker Language (GML) is based on the C programming language that we used for Xbox, it wasn't too difficult. What was trickier was managing our source files—for the entirety of the project, neither Andy nor I used any form of source control, a way to back up source code that allows multiple people to work on it simultaneously. Instead, we did things the old-fashioned way: by passing zip files back and forth through email!

Andy and I complemented each other well: We could both draw, program, and design games, but my focus was on art and design whereas his was on programming. When Andy was let loose on a problem, like making water ripple nicely or making the AI of computer-controlled players behave realistically, he made fast progress on it. But for a project to progress as a whole, the individual parts have to progress at relatively equal speeds. You have to know when what you're working on is good enough

to put on hold and it's time to move on to something else, and that's a skill that came more naturally to me. Sometimes, I would suggest to Andy that he move on to another area of the game, and I could tell that the same obsessiveness that made Andy such a great problem solver also made it hard to leave a problem not fully solved.

Kevin played that role for me, too. He watched us from even higher up and pointed out things I wasn't noticing because I was more concerned about the game as a game versus the game as an XBLA project. In that way, with Andy at the technical level, me at the game level, and Kevin at the project level, we slowly but surely made our way across the wasteland toward a release that at many points seemed as though it might be a mirage.

Multiplayer

As Andy and I missed deadline after deadline, I realized that it was time to take a step back and think long and hard about everything we were adding. When you're working on a big project like this, it's easy to fall into the lull of the grind. Grinding isn't particularly fun, but it can be a welcome respite from making difficult decisions. As if zoning out on a long drive, you can quickly find yourself somewhere that you don't want to be unless you blink your eyes and look around once in a while.

The Challenge Mode I had included in my initial pitch to Microsoft was the first feature from that original

design document to go. Partly inspired by the levels that players had made with *Spelunky Classic*'s in-game level editor, the idea was to include a series of pre-made rooms that tested the player's reflexes and their understanding of the game. In one room, for example, the player might have to use a damsel or monster to carry a sticky bomb somewhere where the player couldn't reach themselves. Or they might have to let a yeti throw them in order to get to a faraway platform. Something about learning an improvisational game like *Spelunky* through pre-made levels didn't sit well with me, though, and Andy rightly pointed out that designing each level would take a lot of extra work. It was cut without a second thought.

Endless Mode, on the other hand, went through some development before I decided to cut it. It was devised as another randomized mode to sit next to Adventure Mode, only it would be a single continuous level instead of multiple levels that you progressed through one-by-one. Above you, the ceiling would slowly descend, forcing you to move downward as quickly as possible to avoid getting crushed. The way I imagined it, this mode would use all of the same monsters and traps that the main mode did, and even follow the same progression of areas. As you descended further, the Mines would gradually give way to the Jungle, the Jungle to the Ice Caves, and so on, generating new rooms and passageways on the fly, ad infinitum.

I had gotten the real-time level generation working in a rudimentary way by the time I reached the same

conclusion about it that I had with Challenge Mode: It was superfluous and taking too much time to develop. *Spelunky* didn't need a new twist on Adventure Mode—it needed Adventure Mode to be as good as it could be. We were also starting to become heavily invested in cooperative multiplayer and a new Deathmatch mode, so it made sense to cut out extraneous features so that our time and energy could be redirected toward those.

If single-player *Spelunky* was intended to be a hardcore arcade game, then co-op was intended to be a casual party game. It seemed pointless to create leaderboards for two, three, and four players playing together—I imagined the number of people that would compete at *Spelunky* in teams would be very limited. Instead, I focused on making it as enjoyable as possible for players of any skill level playing together. My goal was to minimize how much a new player would feel like a burden on an experienced one.

One way I approached the problem was to let players pick one another up, the way they could pick up an item, a damsel, or a stunned monster. Not only was this in-line with that fundamental roguelike rule of making everything share a fundamental ruleset, but it also gave players a direct way of helping each other out. A skilled player could carry someone through a tricky area or if one player had a cape or jetpack, all players could benefit from it.

Another problem with multiplayer games is that it's frustrating when one player dies and becomes a spectator. If the skill levels vary by a large margin, the

weaker player might be left out for a long time, and the better player might even end the game early so that both players can keep playing together. My solution for this was to have the dead player reappear as a ghost that can fly through walls but is otherwise limited to blowing puffs of air. The living players can revive the dead player by breaking open a coffin in the following level.

These changes make multiplayer much easier in some ways, since you can keep reviving one another as long as one person stays alive. But since multiplayer in *Spelunky* is mostly about enjoying the company of your friends, I see nothing wrong with that. It's also balanced out by the fact that, no matter how many people you play with, your entire team only gets a total of four bombs and four ropes. So if one player is weaker, they'll be taking resources away from the stronger players. It's easy to get in each other's way, too—unlike a lot of games where cooperating players can't harm one another, in *Spelunky* you can accidentally whip your friend into a trap or blow them away with a shotgun.

On the flip side, both the carry system and the ghost system can be used maliciously. A player who's picked up another player can throw them into a pit of spikes just as easily as they can carry them over the pit. Likewise, ghosts can cause mischief with their breath ability by triggering monsters and traps, or by flying around the screen to try and distract them. I thought this also fit with the spirit of the game, both its dark sense of humor and the way each action can accomplish multiple things.

In Deathmatch, there's no doubt about anyone's intentions, since the goal of the mode is to kill one another in a series of single-screen arenas. This mode was never part of my original plans for *Spelunky*, but its design seemed more straightforward than the two modes I cut and it didn't feel like it stole any of Adventure Mode's thunder. When Andy suggested it in the middle of the game's development, it reminded us both of the fun times we'd had playing competitive games like *Super Bomberman* and *Street Fighter II*.

Andy and I played Deathmatch extensively in our down time, but it never seemed to catch on once *Spelunky* was released. In a review for Giant Bomb, Alex Navarro wrote that, "while there's a nice variety of maps, the gameplay [in Deathmatch] is just too frantic to ever cohere into a playable experience," and in a video made for the site, other editors back up that claim by cranking up the number of bombs each player starts with to 99, causing each match to last as long as it takes the first bomb to detonate. An IGN video showing off the mode simply states in its description: "WTF is going on here? Utter madness."

In my battles with Andy, we had plenty of matches that ended in chaos, but many more where skillful play determined the victor. Whereas most players chuck bombs and hope for the best, Andy and I became adept at placing bombs at our feet, picking them up, and then timing our throws so that they would explode precisely when we wanted them to. This made maneuvering and positioning very important. Many of our matches were

also won based on who could get to high ground faster, with ropes serving as weapons to knock the other player down as well as ways to climb up.

Most players didn't approach *Spelunky* as a competitive fighting game, so it's not surprising to me that the mode wasn't taken seriously. Not that we ever intended it to be totally serious—my main goal for *Spelunky*'s multiplayer mode was for people to play and laugh together, whether they were introducing someone to the game or trying to see what happens when you fill a small arena with hundreds of bombs. If these modes managed to convince a few players that dying in video games is not only okay, but fun, then they served their purpose.

The Damsel

You start with four hearts in *Spelunky*, and you can lose them quickly, possibly all at once if you fall far enough or get hit by the wrong trap or monster. I wanted to give the player opportunities to gain back their hearts, but to make those hearts feel valuable, they needed to be few in number, easy to lose, and also hard to get back. In most games, health is given back through power-ups that take effect as soon as you touch them, but that seemed too simple for *Spelunky*'s most valuable currency. As with the shops, this was a good opportunity to force the player to make more difficult decisions.

The solution came quickly: Why not employ the tried-and-true cliché of the damsel-in-distress, which fit neatly with *Spelunky*'s pulp adventure setting, its dark sense of humor, and its primary mechanic of carrying items around? Instead of simply touching a heart container or pork chop to gain back health, the hero could rescue a helpless young woman and carry her to the exit the way they might carry a gold idol, earning a restorative kiss in the transitional hallway to the next level. As inspiration, I thought of characters like "Willie" Scott from *Indiana Jones and the Temple of Doom*, a blonde actress whose lines were mostly composed of various high-pitched shrieks as she contended with elephant dung, monkey brains, insects, and death traps. My *Spelunky* damsel would be just as irritating as Willie was in that film—I made her run screaming when the player set her down, oblivious to any danger around her. In practice, this behavior fits the cliché quite well, since she runs off ledges and sets off traps, forcing the player into precarious predicaments unless they find a safe spot to put her. Visually, she's as out of place in the dusty caves and ancient ruins as damsels usually are, with her bright yellow hair, red dress, and high heels.

The problem with the damsel-in-distress cliché is that, while including a damsel-in-distress in your game isn't the same as saying all women are helpless, it does reinforce a stereotype that is prevalent, not only in games, but in every art form. So even though a single instance of it might not be strongly felt on its

own, eventually those drops do fill up the bucket. And which buckets we notice filling up depends on which stereotypes affect us most—as an Asian man who plays video games, I'm used to seeing people who look like me and share my interests being depicted as awkward, unattractive nerds in movies and on television shows.

Unfortunately, even by the final official release of *Spelunky Classic*, I hadn't done much to address the issue. At the time, the Xbox remake was underway and as far as the freeware game was concerned, I was mostly interested in ironing out the remaining bugs and fleshing out the game's level editor. To balance things out a bit, I added a "changing room" that the player could unlock by rescuing eight damsels in a single run. After entering the changing room, Spelunky Guy will exit as the damsel and he'll become the one who needs to be rescued instead. The damsel never dons the appropriate clothing for adventuring, though—it's only a simple sprite swap.

The criticism of the damsel after the game's release was pretty even-handed. I thought the following post by a user on TIGForums was particularly insightful:

> I think it's always good to point out sexism where you find it. In the case of *Spelunky*, you've got a female character who is literally an object that you use. That's fairly undeniable. Using a person as an object in and of itself isn't demeaning, mostly just funny and irreverent, but if the default way

> of playing the game was to play as the woman and use the man for the same purposes, people would comment on how that's the reverse from what you'd expect. It's only an issue because it fits into a larger pattern where we are familiar with female characters in games usually being glossed-up game mechanics instead of subjectivized characters.
>
> It doesn't bother me as much in *Spelunky*, mainly because it seems to be less sexist and more just silly, irreverently playing with the archetypes of the genre, so a shrill damsel stops pretending to be a character and actually becomes an object. But playing with an archetype still keeps the archetype around, and the best way to engage with that is to just recognize it and be willing to talk about it.

That silliness and irreverence in *Spelunky* was something that differed from *Eternal Daughter* and *Aquaria*, which both had female protagonists. I was glad that people saw how I intended the damsel to fit into *Spelunky*'s world—where, despite the light-hearted visuals, the "hero" is clumsy, greedy, and destructive, and the shopkeepers won't hesitate to punish thieves with cold-blooded murder. At the same time, I understood that including a damsel was adding to the mountain of books, paintings, movies, games, and other media that also had them. I had to weigh this trade-off carefully moving forward to XBLA.

Since multiplayer was going to be a key new feature on XBLA, the plan was to have four playable characters available at the start of the game, with sixteen more to unlock later. This seemed like a good opportunity to add some diversity to the cast, since I wasn't satisfied with the changing room from *Spelunky Classic*. I don't think there's anything wrong with adding some "diversity for diversity's sake," since diversity is generally a good thing in and of itself, whether you're talking about race and gender or monsters and traps. Regardless of whether it gives representation to more types of players, it also adds variety to the game's world.

It seemed obvious that "Spelunky Guy," the red-nosed protagonist from the freeware title, would continue to be the hero, but there was plenty of room for new characters of various genders and races. For one, I chose my friend and fellow game developer Colin Northway, who not only provided valuable feedback and encouragement for the original *Spelunky*, but also had fantastic orange mutton chop sideburns that looked great with the blue pith helmet I gave him. I based the second character on Sikh men, who wear turbans and usually have a moustache and/or beard. Finally, the fourth and final starting character was a woman vaguely inspired by Marion Ravenwood, another character from the Indiana Jones series (and one who stands on more equal footing with Indy than Willie Scott). I gave her a green bow in her hair. The rest of the cast, which included a cameo from Meat Boy, was populated in a similar fashion.

For the damsel, I decided to let players select different types from the options menu. Aside from the original lady, you could choose to be kissed by a hunky, shirtless dude or get licked by a cute little pug dog. In keeping with the color scheme of the original damsel, I gave the man blonde hair and a red bow tie, and the pug a red collar with a yellow dog tag on it. All of the damsels would be equally under-dressed and helpless.

Following the game's release, response to the damsel option was quite positive. In a Kotaku article titled "Spelunky's Best Feature Lets You Rescue Whoever You Want," writer Evan Narcisse wrote, "While such tweaks don't affect the basic mechanics of the game, it does provide a slight tonal shift to whatever stories spinning in the back of your head." For the most part, this solution was satisfactory to players who were unhappy with the damsel in *Spelunky Classic* as well as players who hadn't cared about it one way or the other. The pug damsel, or "pugsel," went on to become one of the game's most beloved characters. It seemed like I had made the right decision.

On August 1, 2013, however, the popular video series by Anita Sarkeesian called Tropes vs. Women in Video Games released its third episode, "Damsels in Distress: Part 3," which criticized *Spelunky* for its inclusion of the female damsel while also acknowledging the addition of the male and pug damsels:

[D]amsel'ed female characters tend to reinforce pre-existing regressive notions about women as a group being weak or in need of protection because of their gender, while stories with the occasional helpless male character do NOT perpetuate anything negative about men as a group since there is no long-standing stereotype of men being weak or incapable because of their gender.

To help illustrate this point let's quickly take a look at the indie game *Spelunky*. Originally released in 2009 the game included a stereotypical damsel in distress as a gameplay mechanic whose rescue rewarded the player with bonus health. The 2012 HD remake of the game for Xbox Live again features the stock character damsel (complete with newly upgraded boob jiggle). However, this time an option was added to the menu that allows players to select a replacement for the default woman in peril by switching to either a Chippendales-style hunk or a dog instead.

Setting aside the fact that if a female character is easily interchangeable with a dog then it's probably a pretty good indication that something is wrong, merely providing an optional gender-swap is not a quick and easy fix, especially where stock character style damsels are concerned.

The two may appear the same, but they don't mean the same thing in our culture. This [damsel] is still a problem while this [dude] is not. Again

because one reinforces pre-existing stereotypes about women, while the other does not reinforce any pre-existing stereotypes about men.

Interestingly enough, the very same episode also praises *Aquaria* as a rare example where a woman rescues a man. When I mentioned that fact on Twitter, it led to a brief, friendly exchange with Anita where she clarified that she thought the rest of *Spelunky* was awesome.

Almost immediately, other Twitter users jumped in to chime in that Anita was "hyper-feminist," "crazy," and "ridiculous." I explained to these users that, despite the positive changes I had made to the game, I wanted to think about her criticism more. The fact that it was about gender representations and not graphics or gameplay didn't make it any less valid to me. I was surprised that her video drew such vehemently negative reactions given that, as the person who was being criticized, I felt it was pretty fair.

Part of the problem is that we have a tendency to see discussions as battles between good and evil rather than learning opportunities. Under that pretext, it's very hard to find common ground. Mix in fear, like the fear of "censorship," and suddenly, all nuance is lost. It becomes impossible that Anita Sarkeesian might think *Spelunky* is awesome while also criticizing a part of it. It becomes impossible to understand that she might think that my portrayal of the damsel is sexist without thinking that I am a sexist. And it also becomes hard to understand why I might appreciate her criticism.

I still believe that art is a place for ideas, particularly offensive ones. I still have my sense of humor, the one that thinks it's funny when a yeti throws me against a wall until I die. But what I've begun to realize after working on my own games is that not every potential idea is equally vital to the creator's vision. In something as complicated as a video game, it's impossible to give each aspect of the project the same amount of care and attention—some ideas are simply there because they were the first ones to fit, not because they were the best solution. This is true of bad game mechanics and this is also true of clichés, which, by definition, are devices to fall back on when you can't think of anything more original. In most cases, clichés can be easily replaced by taking a single step toward a more innovative idea. Does a platform game have to have another princess to rescue? Does a gritty game have to begin with another rape or murder? I can't speak for developers other than myself, but I imagine that in many studios, these questions aren't always being asked.

I want to ask these questions about my own games, as well, although familiarity makes it challenging—over time, we become fond of even a game's flaws. The original *Spelunky* was so steeped in its pulp adventure inspirations that it's hard for me to imagine the damsel being something different, but in the remake her absence wouldn't have as large an impact, since she's not the only damsel, but one of three.

Moving forward, the decisions are easier. I want to make my casts even more diverse and less stereotypical,

because I think it will make the games better. I'm also going to ask myself whether an idea is vital to my game, or simply the first solution that solves my problem. I'm planning to do these things in the same way that I would make sure that my future games run at 60 FPS instead of 30. Once you're made aware of a problem and realize that it affects players, it's something you want to fix.

Being more aware also makes flaws more noticeable. If you become used to playing games at 60 FPS, your favorite old games may bother you when they never bothered you before. But would you trade your increased discernment for the bliss of ignorance? Probably not. With increased awareness also comes the potential to enjoy things on a deeper level. Were I not, for example, disappointed with where the Zelda series has gone, I might not appreciate how much the Dark Souls games have taken up its mantle. Classic games will always be classic games, but we should recognize that they're products of their time, both technologically and socially.

One person has changed the way I view this topic more so than any critic: my two-year-old daughter. As a parent, I'm forced to see the world from her perspective, whether it's to figure out if a dangerous object is within her reach or to decipher what she means as she tries to express her ideas. Spending time with her, I've noticed things I've never noticed before, like when she pointed out that the exit sign in our building's hallway looks like a robot. And I've noticed many things that I'd like to make better for her, like how, when people

see an animal or toy that's ambiguous in gender, they'll almost always make it male. I smile sadly when little girls correct their moms that the dog, fish, or teddy bear is actually a sister, not a brother.

But I'm also noticing more and more games, toys, and books that have women in them who are multidimensional major characters, who aren't in distress, and who are wearing appropriate clothing and armor instead of bikinis and ripped skirts. And in the same way that it's not obvious you're breathing stale air until you step outside, I never realized how much more vibrant and inviting our creative spaces could be until these changes were made. I'm appreciative of the people who spoke up about it, including the fans of *Spelunky* who respected me enough to challenge me about my designs.

Last 10%

The final tip from my "Finishing a Game" article is to remember that the last 10% is the last 90%. This is a tip that I credit to my dad, who told me to double my estimates for how long a project will take. This notion is also popularized in a computer science aphorism called the "ninety-ninety rule." Attributed to Tom Cargill of AT&T Bell Laboratories, the rule is: "The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time."

The idea is that we nearly always underestimate how long it takes to finish a project, particularly the last part. It happened to me and Jon with *Eternal Daughter*, it happened to me and Alec with *Aquaria*, and it happened to me, Andy, and Eirik with *Spelunky*. There's also a paradox called the Dichotomy Paradox, which states that it should be impossible to get from one point to another, since, no matter how close you are to your destination, you can always divide the remaining distance by half. Trying to finish a game feels the same way. In our case, it seemed like we were always "six months" away from finishing the game for the entire duration of the project. Why does this happen?

There are a lot of reasons. The biggest reason is that we tend to come up with estimates assuming that we'll be working under perfect conditions the entire time: perfect health, working all day and all night without distractions, and everything going smoothly on the first attempt. We don't plan on getting the flu or getting burned out. We put aside loved ones who deserve our attention. We don't think about getting stuck on a programming bug, or being paralyzed by the sheer number of tasks that lay before us. We're not human beings when we estimate how long something will take—we're unstoppable robots fueled by caffeine and passion.

Another thing we don't think about is the grind of polishing our games. When we're asked how long something is going to take, we imagine the fun parts, like designing and programming level-generation

algorithms, or creating concepts for new characters. It doesn't cross our mind that we'll also have to design menus and loading screens, or that each new character needs to be painstakingly animated. In *Spelunky*, for example, a single playable character had over a hundred frames of animation that were copied over and re-painted by hand. It took me several days to fully paint a character, and even longer to polish it.

These tasks can add up quickly. One day, Andy was working on tracking all of the player's statistics, and we had the following conversation:

Andy: ok
ugh-- this is a lot more to think about than i thought initially

score per level, if you killed a monster or if it just died, etc

Me: yes, it's a pain
it was a pain the first time i did it

Andy: i thought i would knock this out in a day… :\
not so likely now… we'll see

i must say-- it is making me appreciate Spelunky in a whole new way working on this :)

Me: HA!
you thought it was just some easy-peasy pixel platform game, didn't ya? ;)

Andy: haha-- i guess i did in some sense
i mean-- i appreciated the level generation

but not the more subtle stuff
i didnt realize how much work had to go into them, since I have never made it to the subtle feature part of game making

Me: yeah it's the little stuff that gets ya in the end

Until you've finished a game you don't realize how time-consuming all of that "little stuff" is. It's the polish and the details. It's making menus that transition smoothly and contain all of the options that players expect from a released game. It's adding that puff of dust that appears when the player hits the ground, or the wobble to objects and UI elements that makes them feel tactile and alive. All of that is another 90% of work, on top of the 90% you've already done.

And releasing a game on a console is perhaps another 90%. To meet a console's technical requirements, you have to make sure that your game can handle itself elegantly through almost any situation barring the player taking a hammer to it. Testing involves trying every possible thing you can think to do, like plugging in and unplugging controllers, memory cards, and

ethernet cables at each juncture to make sure that the game displays the correct error messages.

To be on a console, you also have to get your game localized into several different languages. *Spelunky* was localized to German, Spanish, French, Italian, Portuguese, Japanese, Korean, and Chinese. With localization, handing your text to translators is only part of the task—the differences between languages can easily break the in-game formatting. German has some particularly long words, and a piece of text that fit perfectly in English might get cut off or extend into other parts of the UI. For example, "Royal Jelly," the miraculous health recovery item that bees drop in the Jungle, translates in German to "Königlichen Nektar kaufen," which is more than double the length. And the Asian languages pose another problem by requiring an entirely new set of characters.

You'll have to get your game rated, too, which unfortunately means submitting it to multiple agencies: the ESRB for North America, USK for Germany, PEGI for the rest of Europe, COB for Australia, GRB for Korea, and CERO for Japan. Submitting a game for rating usually involves writing game descriptions, creating gameplay videos showing potentially objectionable content, and filling out forms where you have to admit that your game features a monkey that can poop gold and gems.

As you approach the finish line each time, nothing else matters. Your days are consumed with work until

the finish line moves a little bit farther away, whereupon you allow yourself a little break to rest and catch up with friends and family. At some point, though, the finish line stops moving, and instead of relief, it's panic that sets in. You realize that while it might be frustrating to try and hit a moving target, it's *terrifying* when it stops because that means you're out of opportunities to improve the game. Everything that was "good enough" to leave as it was now needs to be perfect because this is the version of the game that players and critics all over the world are going to see.

So while Andy was under the table in his home office, plugging in and unplugging memory cards to try to replicate the problems that Microsoft would check for in certification, I was filling out forms and testing *Spelunky* repeatedly for any bugs or weird behavior, understanding for the first time the downside of having a randomly-generated game. My biggest fear, actually, was not that we would fail cert, but that, given the random nature of the game, we would pass cert with some kind of horrible bug that the testers failed to catch. I thought of the first release of *Eternal Daughter*, where a last minute change had resulted in a crucial platform being unreachable, preventing players from progressing—but at least then we were able to upload a fix immediately. On XBLA it could take weeks.

Each person was anxious about his own part in the development during this final phase. Andy worried

about whether his programming would do justice to *Spelunky* for its fans. I obsessed about the artwork, examining every sprite from different angles and tweaking them before examining them again. Good habits, like working off of a separate, higher resolution image when making changes to artwork went out the window—by this point I was painting directly onto textures used by the game. There was no turning back.

The final moments before certification were a daze as Andy and I worked with the testing team at Microsoft to whittle down the bug database and make sure that *Spelunky* would satisfy all of the technical requirements. My sleep schedule was so off at this point that there were many days when I'd wake up and run down the street to try and catch the last rays of the sun as it set. Back home, I'd continue playing the game nonstop, challenging myself to beat it all the way through and trying to enjoy the experience even though I was scrutinizing every pixel on the screen. It dawned on me how large the game actually was. Although one could easily think of *Spelunky* as only two main modes, Adventure and Deathmatch, there was so much more to it than that: menus, cutscenes, tutorials, leaderboards, and multiplayer, all of which had to work flawlessly for players when the game released. It was impossible to keep it all in my head at once.

Andy, his wife, and I were eating frozen yogurt near his house in Connecticut to celebrate finishing the

game when we found out that we failed our first certification. *Spelunky* crashed when you tried to launch it from a memory card while the console's region was set to Japan or Europe. We went home immediately so that Andy could fix the bug and send the game right back in.

We passed our second certification on June 21, 2012, just shy of three years from when Andy first asked me if I needed help with *Spelunky*. My original estimate to Kevin right after we signed was that I would hit final certification in July of 2010.

So I was off by a couple of years.

I don't think it's a bad thing to underestimate how long a project will take. Had I estimated that *Spelunky*'s development time would be three years, I may not have picked up the project. Andy may not have picked it up. Microsoft may not have picked it up. This sounds strange to say, given that we were all okay with delaying the game for years, but psychologically, it's a lot easier to accept a six-month delay several times than it is to accept three years of continuous work. A closer target is easier to aim for. Were we given three years from the outset, would we have finished the game before then, or would we have had to tack another six months to the end of that? It's hard to know for sure, although I have to admit, with a cringe, that it might have taken longer. So that initial overconfidence might actually be a good

thing, provided that it's paired with the determination and resources[12] to see things through until they're done.

Expectations

On August 13, 2012, a regular column on Gamasutra that examined the volume of sales on XBLA came out with the headline: "Spelunky console debut doesn't hit indie expectations set by Fez, Braid." Ouch. The column stated, "Despite *Spelunky* being a well-known indie darling, it hasn't seen the same popularity of games like *Limbo*, *Braid* and *Fez*. The game added only 17,818 players in the first week for 33,107 players for the month. It is the best-reviewed title, however." By comparison, *Fez* had sold 76,312 copies its first month and *Limbo* had sold 78,044 copies its first month.

The column, which used leaderboard data as a way to determine the number of players who had bought the game, didn't offer answers as to why *Spelunky* performed worse than expected, but in forums and comments sections there were lots of opinions. Some people felt that *Spelunky*'s 1200 Microsoft Point price

12 During the three years that we worked on *Spelunky*, I was supported by the continued sales of *Aquaria*. Andy was supported by what he had saved from his five years as a wooden toy designer and by his wife's work. Eirik had a day-job and also took on other freelance work as a composer.

(equivalent to $15 USD) was too high to make it an impulse buy, especially compared to *Fez*, which released earlier that year at 800 points ($10 USD). Some felt that the game wasn't marketed well enough by us or by Microsoft. Other people thought that perhaps the free demo, with its high level of difficulty and randomized levels, may have offered so much entertainment for the average player that they never needed to buy the game.

Some simply pointed to it as the end of XBLA as a promised land for indie game developers, especially since 2011 had seen a lot of public grumbling about Microsoft from indies. In October of that year, Ron Carmel of 2D Boy revealed in an article titled "Is XBLA Past Its Prime?" that the number of developers working on games had shifted from XBLA to Sony's PlayStation Network (PSN), in large part because of the difficulty of working with Microsoft as a company. (Kevin Hathaway was mentioned as a notable exception to this.) A few months later, Team Meat explained in a GDC talk that they had crunched hard to release *Super Meat Boy* in time for a promotion, only to have Microsoft reduce their placement on the dashboard at the last minute and pair them with another game release the week they were supposed to have it to themselves.

For my part, I found Microsoft's promotion of *Spelunky*—a fifth place slot on the homepage of the dashboard—to be merely adequate, but we were never given the same promises that Team Meat was. Instead, I thought a lot about *Spelunky* itself, since I've always felt

that, while marketing is important, it's only a multiplier of how good your game is. I came to the conclusion that we never made that strong first impression that Kevin had pushed us so hard to try to make. *Limbo*, *Braid*, *Fez*, and *Super Meat Boy* each have something about them that stands out quickly in screenshots and trailers. In *Limbo,* the moody black and white graphics immediately popped out at you when scrolling through websites or flipping through magazines. In *Braid*, the inventive "time rewind" mechanics caught the attention of both the gaming media and the rapper Soulja Boy, who posted a video of himself and his friends playing it. In *Fez*, the charming pixel artwork and the way you could rotate its 2D world to reveal a third dimension created a visual style that was simultaneously familiar and new. And *Super Meat Boy*'s brash and revolting sense of humor dripped, like grease, from the title all the way to the game's manic soundtrack. On the other hand, *Spelunky*'s screenshots, while packed with details, look mostly like a lot of rocks and dirt, while the interesting aspects of its randomization are hard to get across in even a few play sessions, let alone a non-interactive trailer.

Of course, beneath *Spelunky*'s unassuming surface are hidden big secrets and shiny treasures. And in that "iceberg" quality I recognized something the game had in common with *Limbo*, *Braid*, *Fez*, and *Super Meat Boy*—it reflected its creators. I thought about Andy and Eirik, both hiding extraordinary talents beneath

laid-back and unpretentious personalities. As for myself, I struggled to explain my vision for *Spelunky*, preferring instead that people seek it out on their own.

I shared the sting of that headline with fans who saw that the game was not selling as well as other indie games that it was being compared with. That's the problem with expectations—although I had girded myself for anything, I had also gotten caught up in the promise of raking in millions like these other titles had, and when it didn't happen, it was hard not to feel a little bit like we had failed. Worst of all, it was making it more difficult for me to cheer on my peers in their successes.

There were a lot of reasons to feel great, though. First and foremost, we had successfully shipped an Xbox game that met our personal standards and the standards of the many fans that were pulling for us. The game was also, as the article had pointed out, a critical success—not only did *Spelunky* have a Metacritic rating of 88 out of 100, but it also won an Excellence in Design Award at IGF 2012. As part of the festival's feedback process, an anonymous IGF judge wrote a lovely summary of the game:

> With *Spelunky*, you are never learning a "piece" of music… It's still a game about repetition and learning, but what you are learning is the overall composition, understanding the overall system and how it works, and becoming fluent in that. At first I disliked the way the ghost worked against the other elements of the game,

> encouraging recklessness in a game where recklessness was fatal. But eventually I came to love this tension, and appreciate it as the beating heart of the game. *Spelunky* looks like a game of execution, but it's really a game about information and decision-making. How good are you at looking at a situation and understanding what it means? You can't memorize, and you can't take time to carefully analyze, you must rely on your literacy of the system, and this is a kind of holistic knowledge that feels great in my brain, a wonderful new flavor for a single-player game, and a deeply promising direction for further exploration.

And then the column missed out on another important angle: our sales, while lackluster in comparison to the best-selling games on the service, were not terrible by any means.

To help deal with the negativity, I withdrew from the spotlight and tried to focus on what mattered most to me: game creation. The Gamasutra article had an air of finality about it, but really we were just getting started. Sometimes it takes years after a release for a game to get its due recognition, like it was for *Rogue*. In our case, we felt like we still had at least one other shot to make it big, on the platform where it all began: the PC. With XBLA's reputation as a great place for indie developers in rapid decline, Steam—Valve's digital distribution service on

the PC—was being heralded as a new promised land. We only hoped that we wouldn't be too late this time.

Raising the Stakes

Life outside of game development picked up right after we released *Spelunky* on XBLA. I proposed to Frances. Andy and his wife were soon expecting their first child. Then Frances became pregnant with our daughter!

In the meantime we, particularly Andy as the lead programmer, had a lot of work to do stripping out the Xbox-specific code and preparing the game for release on Steam. Thankfully, the conversion was not nearly as difficult as the first creation, and although Andy was sleep-deprived as a new father, he found it easy to integrate Steam into our game engine. All we worried about was whether there would be enough excitement about *Spelunky* on Steam after our lukewarm reception on Xbox, especially since players had had access to a fun freeware version of the game on PC for years now.

In October, four months after *Spelunky*'s release on XBLA, I received another serendipitous email, this time from a game developer named Doug Wilson. The email would lead to a whole new way of playing *Spelunky* (and perhaps video games in general). In the email, Doug asked me to check out a blog post he wrote about playing *Spelunky* with his roommate Nicklas Nygren, better known as the game developer "Nifflas." Nifflas and Doug

had come up with a unique new way of playing *Spelunky*: Each day they would play one, just one, game of *Spelunky* each, while the other person watched and cheered along.

This reminded me of something Hideo Kojima, the creator of the Metal Gear series, once said in an interview: that his dream was to create a game where, when the player died, the game would physically destroy itself, preventing the player from restarting. We had toyed with similar ideas while working on the Xbox version of *Spelunky*, like letting players purchase, via the mechanism of buying extra downloadable content (DLC), a single run through the game that would put you on a separate set of leaderboards. Ultimately, though, these kinds of radical ideas were hampered by the complexities of the console development process.

On Steam, however, such ideas were given renewed hope. Steam, we knew, could be home for a wonderful rebirth of *Spelunky*. We had heard great things from other developers about it, like Edmund McMillen and Tommy Refenes of Team Meat, who, after expressing their grievances with Microsoft out loud, happily announced that they had sold twice as many copies of *Super Meat Boy* on Steam than on XBLA at the time. We were also finding that Steam's reputation for ease of publishing was turning out to be true. There was no grueling certification, filled with countless corner-case requirements that needed to be fulfilled, or even a ratings requirement. Patching was also a breeze—to release a patch, you simply upload it. On Xbox, patches, called

"title updates," cost the developer a fee after the first one and sometimes didn't release for weeks after they were ready because of how rigorous and regimented the approval process was.[13]

I was excited about Doug and Nifflas's ritual. In Doug's words:

> This ritual has been deeply enjoyable for several reasons. First, the tradition gives us something to look forward to every evening. Second, the "stakes" of the game feel so much more real when you only get one shot. One error and you're done for the day. Nerve-wracking, but invigorating! Third, and perhaps most importantly, I find that it's far more rewarding to play the game with somebody spectating—a witness with whom to share your triumphs and tribulations. After all, *Spelunky* is all about the stories you the player end up producing. As my hero Hannah Arendt puts it: "The presence of others who see what we see and hear what we hear assures us of the reality of the world and ourselves."
>
> Jokingly, before each run, we make a little prayer to Derek Yu, the game's creator. (For example: 'Derek Yu, please grant us plentiful

13 They eventually changed this policy after the creators of *Fez* made it public.

bombs and protect us from dark levels. Amen.') The prayer has itself become a key part of our daily ritual—to the point where we feel like we've almost created a *Spelunky* religion/cult. Like, why do bad things happen to good *Spelunky* players? And does Derek Yu even exist? *Spelunky* theology is tricky!

As it turns out, I do exist! I even make occasional trips out of the house, as I did one sunny day in San Francisco, when I met Doug to record a video segment with Anthony Carboni called "Why Game Developers Love SPELUNKY!" Before and after the recording, Doug used his boundless enthusiasm to sell me on a new mode to include in the Steam version of *Spelunky*. In this new mode, which was based on his daily ritual with Nifflas, a fresh series of levels would be randomly generated each day and each player would only get one attempt to beat it. Once a player's run ended, either by beating the game or dying, they would have to wait until the next day to play again. We ended up calling it the "Daily Challenge."

Adding the Daily Challenge to *Spelunky* wasn't trivial. First, Andy had to make sure that after the levels were generated each day they would be exactly the same for each player. This required using a new way to generate random numbers and ensuring that the contents of treasure chests and crates would be the same for everyone who played (like the box holding

Schrödinger's cat, the contents are normally determined when they're opened). Andy also had to verify that it was possible to generate new leaderboards on Steam for each Daily Challenge. Thankfully, it was.

Behind the scenes, Andy and I began testing the Daily Challenge with Doug and Zach Gage, another game developer who helped us refine the idea. Already, the mode was showing its merits—since the levels were the same for everyone, we could compare runs and easily swap stories with one another. It also meant that we couldn't blame bad runs on anything other than our own screw-ups. On top of that, the "one a day" rule raised the stakes of each run, so we were completely focused while we were playing. In Adventure Mode, it's only when you're close to accomplishing something new—reaching a new area, beating the game for the first time, or getting a high score—that the hairs on your neck start to rise, but in the Daily Challenge, every moment was thrilling. This was starting to feel like it was perhaps the definitive way to play the game.

There was one aspect of Doug and Nifflas's ritual that we couldn't simulate with the Daily Challenge: having a witness to share your triumphs and tribulations with. For that, we'd have to depend on players to share their experiences online themselves. On Twitch, streamers interact with their audience through chat rooms while they stream video games. Other times, players talk over a playthrough in private before releasing the video straight to YouTube. These videos, which evolved from

people sharing screenshots and written commentary on forums, are commonly called "Let's Plays," and the players "Let's Players" or "LPers." To promote the Challenge and give them some exposure, I asked five streamers who supported us on XBLA if they would be interested in playing the mode against one another for the week before the Steam release. The five players—Northernlion, MANvsGAME, Foxman, BaerTaffy, and RedPandaGamer—released videos through their own channels and challenged one another on social media with some friendly trash-talking.

Spelunky's release on Steam went great. In our first month, we sold 61,408 units on Steam, twice as many as we sold on Xbox 360 in the first month, and two years out, we've sold 577,185 units there in total, thanks to the streaming community and the incredible reach of Steam's annual sales events. The game ended up winning *PC Gamer*'s Game of the Year award for 2013.

Sony also began publishing *Spelunky* on PSN shortly after, pairing us with a team of super coders called Blit Works to port the game and the Daily Challenge to the PlayStation 3 and PlayStation 4. It was also ported to PS Vita, making my original dream of putting *Spelunky* on a handheld system a reality. On PSN, *Spelunky* has sold 458,534 copies, not including the copies that were given away to PlayStation Plus members.

Sales have been steady on XBLA, too, with the game selling 158,727 units since its debut there.

All told, *Spelunky* has sold over a million copies across all platforms.

And every day, players are still drinking their daily cup of *Spelunky* coffee by playing the Daily Challenge.

PART IV:
SPELUNKY TRANSCENDED

Since you worked so hard to get here,
I'll bet you're expecting a large reward.
Well, with all the time I've spent down here,
I've come to realize a few things…
That the journey is its own reward
and mastery is the greatest treasure of them all!

Well, okay, so gold is pretty nice, too!

— Yang

Beyond the City of Gold

I DISLIKE BOSS BATTLES that follow a rigid pattern, because I think it stifles the creativity of the player. A common example of rigid boss design is the boss that's impervious to your attacks until you solve a simple puzzle to reveal its weak point. These puzzles might involve flipping switches or tricking the boss into injuring itself against the environment. After striking the weak point a few times, the boss will cover it up and become invincible again, restarting the cycle. Most of the time, this is more of a chore than a challenge. Once the puzzle is solved, it becomes a purely repetitive exercise, and the "weak point" mechanic feels artificial.

My ideal bosses are more like upgraded regular enemies. In *Spelunky*, there are several mini-bosses that roam the levels, like the giant spider in the Mines that spits webs, or the Yeti King in the Ice Caves whose yell can knock the player over and send blocks of ice hurtling down from the ceiling. These bosses are larger than regular enemies, have more health, and deal more damage, but they're susceptible to the same attacks that hurt regular enemies and can be avoided altogether if the player is careful. In many cases, they can also be killed instantly by being bombed or crushed. That way, *Spelunky*'s mini-bosses seem like part of the game's

ecosystem instead of a mechanical device that was added just to block the player's path.

For Olmec, the final boss, I decided to do something slightly different. Olmec, inspired by an ancient Mexican civilization of the same name, is four times larger than the Yeti King and sixteen times larger than the player. He's impervious to normal attacks, but has no weak points that open up—the only way to beat Olmec is to get him to drop into the large lake of lava at the bottom of his level, beneath a floor that's ten tiles thick. It is possible, if you have enough bombs, to bomb a large hole for him to fall into, but you can also exploit one of Olmec's attacks to do it for you. This attack begins with Olmec leaping into the air and then dropping down when the player is underneath him. Once he slams on the ground, he destroys the top layer of floor tiles.

I'm pleased with how the Olmec battle uses destructible terrain and how much variety there is to it. Even though your goal is always the same, there are many ways that the battle can go, depending on the equipment you bring with you and the chaos of the enemies that Olmec spawns during his attacks. Small holes that appear during the floor generation can also change the battle dramatically by trapping you in the hole with Olmec. Much of the boss fight is about intimidation, too—Olmec's jumps are well-telegraphed, giving you plenty of time to move away, but his immense size makes it a tense fight every time.

In *Spelunky Classic*, beating Olmec in level 4-4 heralded the end of the game, but before reaching him, you could visit the City of Gold in 4-3 for an even more impressive and lucrative win. For the remake, however, I knew I had to add a link to the Chain to make it even more secretive and challenging, especially for returning players. There also had to be a "true final boss" hiding in the shadows behind Olmec.

It didn't take long to decide that the player should go straight to Hell to fight this true final boss.

In the West, we commonly think of Hell as the one from the Bible, stereotyped in popular culture as a place of fire and brimstone, with Satan, red-skinned and hoofed, presiding over the punishment of nonbelievers. In *Spelunky*, I chose to go with Di Yu, the hell of Chinese mythology. Di Yu was not just a place of punishment, but also, because of Buddhism's influence on the myths, a place where souls could be given a second chance through reincarnation. The cyclic nature of Chinese Hell appealed more to *Spelunky*'s design and to my personal philosophies, since I've always found the idea of eternal damnation to be distasteful. Visually, it offered up some incredible imagery, like the fearsome Ox-Head and Horse-Face, who bring sinners before King Yama to be judged. These characters seemed to fit better with *Spelunky*'s melting pot of mythologies.

Hell would be the fifth area of the game, after the Temple, and I'd have to make players go through the City of Gold first in order to connect it to the Chain.

At that point, you'd have collected the Udjat Eye, the Ankh, the Hedjet, and the Sceptre—all Egyptian-themed items. For the final object, I needed something that had a connection to both ancient Egypt and Hell. Fortunately, it was obvious what that object should be: the Book of the Dead. The Book of the Dead worked thematically on many levels: Not only is it a funerary text from ancient Egypt, but it also appeared in *NetHack* as a way of entering the final area of that game.

Spelunky's Book of the Dead was also influenced by the Necronomicon, a fictional book of the dead that was invented by H.P. Lovecraft and cemented in popular culture through the Evil Dead horror movie series. Bound in human skin with a grotesque face on its cover, the book is located in a central room in the City of Gold, where it's guarded by Anubis's second form. This time, the jackal-headed god has a new trick up his sleeve: He can create an army of bloodstained skeletons to attack you.

Once you've brought the Book of the Dead to Olmec's level, you can use it to reveal the location of the door to Hell, hanging two tiles above the lava at the bottom of the level. The Book of the Dead is proximity-based, like the Udjat Eye, and the closer you are to the door, the faster the book's cover squirms. But unlike the Udjat Eye, the Book of the Dead is required to enter Hell—without it, the door is shut. I felt like this was appropriate, given that it's the final link in the Chain. The Book of the Dead is also the one item in

Spelunky that only serves a single purpose, since you go to Olmec's level right after you obtain it.

Getting to Hell's entrance requires breaking through the floor as you normally would, but there's another problem: The door is hanging in mid-air and the game won't let you exit a level unless your feet are on the ground. This trifling little check became the major twist for entering Hell. Although you can't jump through the door during freefall, you can stand on Olmec's head as he sinks into the lava, using him as a glorified platform. It's a fitting way to enter the final area of the game.[14]

Once you pass that threshold, the game's visuals change dramatically, from earthy colors to an unnaturally bright red. Hell feels stuffy, partly because the passageways are small and partly because the mist effect that we used in previous areas is reused here as steam. But what really stands out about it is that the indestructible walls that make up the border of the levels aren't the usual flat, gray stones—they're packed with skulls, instead. In fact, everything about Hell seems extra threatening, from the vampires and minor demons that call it home to the lava and spiked ball traps waiting to punish your missteps.

---

14 Some enterprising players have also managed to enter Hell by using two stone blocks stacked on top of one another. This is the basis of a difficult, player-created "pacifist" run where you beat the game without killing anything directly.

Surprisingly, some players beat Hell on their first attempt, even though as the true final level, you might expect it to be insanely difficult. Also because, you know, it's called "Hell." In video game design, we call the increase in difficulty from the beginning to the end the "difficulty ramp." A challenging game might ramp up quickly to a high level of difficulty and an easy game's ramp would have a much more gradual incline. But is the idea that games should increase steadily in challenge necessarily correct?

In *Spelunky*, most people feel that the second area, the Jungle, is the hardest area in the game, and I would agree. It's packed with difficult enemies who have erratic movement patterns and can kill you instantly, either by touching you, like the man-eating plant, or by knocking you into tiki traps. Furthermore, the levels feature a lot of narrow passageways and long drops, so it's easy to get cornered or plummet to your death. There's also a high possibility for pools and lakes of water to be generated, and while it's impossible to drown in *Spelunky*, the physics are so different in the water that many players have trouble fighting off the piranhas and other enemies that have also fallen in.

The next area, the Ice Caves, offers much more room for maneuvering, and its primary enemies—UFOs, yetis, and mammoths—are slower-moving, so if you're careful and well-equipped, it's an easier area to get through. Whereas the rest of the areas are made of rooms and hallways, the concept of the Ice Caves

is more like platforms suspended in thin air. It's also the only area to feature a bottomless abyss below it. In effect, the Ice Caves is not only a break from the Jungle's reflex-heavy challenges, but it's also a break from the level structure.

Not surprisingly, a large portion of players give up in the Jungle. According to the publicly viewable *Spelunky* achievements on Steam, 42.4% of Steam players have reached the Jungle, but only 23.3% have reached the Ice Caves. 17% have reached the fourth area, the Temple. 7.3% have then beaten the Temple and therefore the game. So less than half the people who own the game on Steam have beaten the first area, and only half of those people have beaten the Jungle. On the other hand, nearly 75% of the remaining players beat the Ice Caves.

The more I play and create games, the less convinced I am that the difficulty of games should be thought of in terms of a linear or exponential ramp upwards where, as the player gets stronger, you need to make the opposition increase proportionally in strength. It reminds me of something Commissioner Gordon says about escalation in the movie *Batman Begins*: "We start carrying semi-automatics, they buy automatics. We start wearing Kevlar, they buy armor-piercing rounds." While some form of escalation certainly feels good in a challenging game, Gordon reveals something futile and perhaps nihilistic about endlessly cranking a single knob that goes from easy to hard. Rather, I believe it makes more sense to think about difficulty in terms of

the game's overall pacing. Difficulty should ebb and flow, and make room for other aspects of play.

There's also the player's mental and physical exhaustion to consider. I think of the difficulty of *Spelunky*'s areas as mirroring my experiences running cross-country when I was in high school. During races, there was always a period of intense pain right after I began running, as my body adjusted to the exercise, followed by a period of relative calm as endorphins flooded in. In a way, it makes sense to frontload the pain in the early stages of the game because that's when players have the most energy to deal with it.

So while Hell is still a challenging area—it's dense with deadly traps and monsters—it's not a terrible jump in difficulty from the rest of the game. By this point, I assume the player is tired and anxious, and that tiredness and that anxiousness are as much a danger to them as any of the monsters or traps I could throw at them. Tough decisions will only feel tougher, and most people will play conservatively after fighting so hard to get there. Instead of testing their reflexes, I play with their emotions, making Hell look menacing with red hues and spikes, and accentuating the claustrophobia of the area's small, twisty passageways with steam. Eirik's chiptune-inspired track for this area is urgent and fast-paced, the exact opposite of what we went for with the rest of the areas.

Hell has another unique twist: In each level there appears to be two damsels, but one of them is actually a

succubus, a shape-shifting monster that will jump on you and steal a heart if you get too close. By causing players to view something that was previously beneficial with suspicion, the area again preys on their agitated state.

King Yama, sitting on a throne at the top of the fourth level of Hell, is designed with a similar sensibility. At first glance, the battle seems tricky. Unlike most of the other levels, the Yama fight moves upwards, with iron chains to climb if the player doesn't have ropes or a jetpack. The dangers here are vampires that can spawn on the bottoms of blocks, spikes that can spawn on the tops, and Yama himself, who smashes his fists onto the armrests of his throne, causing skulls to rain down from the ceiling. Before you can begin climbing, you have to deal with Ox-Head and Horse-Face, who attack with tridents and bombs.

In actuality, I intended Yama to be a more forgiving fight than Olmec. Ox-Head and Horse-Face are easily dispatched with sticky bombs or a shotgun, and supply the player with two dozen more bombs once they're defeated. As for the climb up to Yama's throne, the vampires, spikes, and skulls can all be avoided if you're patient enough. The vampires can be led to the ground for a safer fight and the skulls that Yama showers on you will break harmlessly so long as there's something above you. Yama himself is also vulnerable to sticky bombs and shotgun blasts.

So while Hell is still a challenging area, it can seem more challenging than it already is given where

it appears in the game. Once Yama is defeated, a door opens up and leads you to a room where Yang is waiting with treasure and a few final words of wisdom. By rescuing him, you can safely say you've unlocked the game's greatest secrets.

Or can you?

Eirik's Song

In any shop, there's a 5% chance that an item will be an $8,000 mystery box that looks like a wrapped present with a bow on top. Inside the mystery box is a regular shop item, and at $8,000, you could just as easily get ripped off (e.g. by getting a bag of bombs) as score a nice deal (e.g. a jetpack). But the mystery box has another purpose that is unexplained by the game: If you carry the unopened box to an altar of Kali and sacrifice it, you'll be rewarded with a purple eggplant and a secret song. The eggplant is more or less useless, although the "more or less" part of that statement was enough to make it the center of one of *Spelunky*'s deepest mysteries.

The creation of the eggplant began early in *Spelunky*'s Xbox development, when Eirik told me about "Totaka's Song," a simple, 19-note tune that Nintendo composer and voice actor Kazumi Totaka hid in many of the games that he worked on, including *Mario Paint*, *The Legend of Zelda: Link's Awakening*, and *Animal Crossing*. In *Animal*

Crossing, the guitar-playing dog K.K. Slider is actually based on him—in the Japanese version of the game, the character's name is "Totakeke." If you approach K.K. Slider on a Saturday night and request "K.K. Song," then he'll play Totaka's Song. In most of the games where the song is hidden, however, the player has to sit on a menu screen and wait a few minutes until the menu music ends and Totaka's Song begins playing instead.

Eirik wanted to hide his own "Eirik's Song" into *Spelunky*. The game already had area-themed "easter egg" tracks that have a 1% chance of playing in each level, but he wanted the player to do something obscure to activate Eirik's Song. It seemed like a no-brainer for a game that was already laden with secrets and references, but with other aspects of the development pressing, the discussion of the hidden track was set aside until a team chat right before the release.

Initial ideas that came up and were promptly thrown out included: a secret code (too trite), waiting in the entrance (too easy to discover), and beating up Tunnel Man (too violent). Eventually, we started to approach our answer:

Eirik: Well, if we're gonna put the clue to how to do it on the ost liner notes, it needs to be rather obscure not something you stumble onto like, beat level 2-3 without getting any gold

Me: right
yeah that's pretty obscure

Eirik: haha
so music in the tunnel or following level?

Me: actually thinking about it i
don't know how obscure it is to
not get gold in a level

Eirik: no gold, only idol
how about that?
get 666 gold?

Me: haha i don't think that's even
possible

Eirik: collect a sequence of colored gems
from low to max value
in any level

Andy: 3 rocks on an altar

Eirik: that's pretty cool

Me: there you go

Eirik: holding a damsel whilst standing
on an altar and letting the ghost
get you

THEN wait 4 minutes on the
game over screen

Me: nah too easy

Eirik: haha
3 rocks on an altar sounds good though

Andy: what about if you need to play multiplayer, and place an unwrapped present on the later altar

Me: that's possible in single player with the mystery box

Eirik: yeah

Andy: wrapped present i mean
that is pretty obscure
maybe it turns it into 2 items and the song starts?

Eirik: have a mantrap eat a mystery box/ present, then sacrifice the manplant

Me: ;___;

Eirik: I thought you wanted obscure

Me: dat's not even possible!

Eirik: oh

Me: i like sacrificing a present

Eirik: make it so
yeah
it's pretty unnatural
you'd always open it right away

Me: maybe you get guaranteed bomb box that way
or jetpack
as well as the song

Andy: that might be too good derek
we dont want it to become a good strategy
vlads wings might be good… since they are pretty bad but feel special

Me: you should get something good out of it though
what about golden monkey?
nah
don't know about vlad's wings
i think you should only get those from vlad

Eirik: what about nothing?
you lose it

Me: maybe just a diamond

Eirik: you get mocking music for being stupid enough to sacrifice it
one gold

Me: haha

Andy: diamond is cool

Me: although i think the box costs more than a diamond is worth

Andy: well, could be a few diamonds

Eirik: what about a critter

Me: eirik is trying to think of the most worthless things to put in there

Eirik: haha
what about 3 rocks
or a lit bomb

Andy: what about a slave clone of yourself?

Eirik: that'd be confusing
hah

Me: haha
kinda like that
that is weird as heck though

Eirik: Very

Andy: we could like invert the graphics on them
like your anti-self
haha

Eirik: Bizarre

Finally, I threw out a simple suggestion that stuck:

Me: how about if you get an eggplant in the present

Eirik: are there eggplants?

Me: no
but i could add one easily

Eirik: oh you have space for more stuff
in your sheets?
nice
eggplant would be lovely

Eggplants hold a small but significant place in video game history. Fruit and vegetables have always been popular choices for collectible items in games, and eggplants have occasionally found their way into the art form that way. There have also been a few food-themed titles where eggplants have made themselves at home, such as the strange 1984 NES adventure game *Princess Tomato in the Salad Kingdom*, which features not one but several eggplant characters: Miss Eggplant, the owner of the juice bar in Saladoria, "E.P.," a greedy eggplant man who trades the player a shovel for gold, and finally, the formidable Eggplant Soldier, who claims to be "second only to Minister Pumpkin in Finger Wars." And in the classic point-and-click adventure game *Sam & Max Hit the Road*, there is at least some logic behind procuring an eggplant shaped like a country singer's head from a Celebrity Vegetable Museum.

But it's the games where eggplants seemingly come out of nowhere that have made them so iconic, like the eggplant power-ups in *Ice Climber*, the mysterious masked "eggplant men" that chase you down in *Wrecking Crew*, or the evil eggplant that drains your health in *Adventure Island*. And who can forget the most famous one of them all: the Eggplant Wizard, a giant eggplant who guards the labyrinthine Fortresses of

Kid Icarus and can inflict an "eggplant curse" upon the player that turns them into an eggplant, too. Inspired by *Wrecking Crew*'s eggplant men and his passion for the bulbous purple fruit, *Kid Icarus* creator Toru Osawa purportedly drew the sprite for the Eggplant Wizard in celebration of having received his summer bonus. The wizard's memorable design—the large, singular eye resting above its thick pink lips, the cloak and sandals, the staff with a little eggplant on top—all highlight the absurdity of an ambulatory eggplant appearing in a setting based on Greek mythology.

I've always loved how video games can get away with that kind of strangeness. A mushroom out of a question block. A turkey leg from under an oil drum. A kitchen sink in the middle of a dungeon. There's something empowering about how wackiness can take nothing away from the immersive quality of a video game, so long as the rules that tie it all together are consistent and fun. It suggests that with the right perspective, anything can make sense and anything is possible.

I knew that Eirik wanted the discovery of "Eirik's Song" to serve no other purpose, but I anticipated that players would want to do *something* with the eggplant. We were also trying desperately to get the game out the door, so whatever the eggplant did, I wanted it to involve as little work on my part as possible, and certainly nothing that could introduce new bugs. It was the blank space in the upper-right corner of a sprite sheet called "monstersbig.png" that finally called out to me. In that

space, alongside the sprites for the mummy enemy and the camel you ride away on at the end of the game, I quickly painted a five-frame animation of Yama's face as a big, blinking eggplant. In the source code, I found the function that handles collisions between objects and added a few lines to change Yama's face sprite when he gets hit by the eggplant, playing Eirik's secret theme again in the process.

I only tested the transformation in the most superficial of ways—by editing the game's code to start myself in Yama's room with an eggplant already in hand. When I emailed Andy the build, I announced the inclusion to him unceremoniously in my notes, saying: "Also, you can eggplant Yama for fun, now. Check 'monstersbig' for the eggplant face sprite." After that, I let it go from my mind. We still had a lot of other work to do.

An Eggplant's Chance in Hell

The eggplant was discovered three months after the game came out on XBLA, causing plenty of speculation from the *Spelunky* community about its purpose. On the *Spelunky* forums a thread was dedicated to unraveling the mystery, with players experimenting on the eggplant like mad scientists. Hypotheses like "what if you put it in water or acid?" and "did you try feeding it to a mantrap?" were tested and discarded, one by one.

Eventually, after all other possibilities seemed exhausted, the idea of carrying the eggplant to Hell came up, whereupon users came to the same conclusion I had when I was coding Yama's transformation: While it might be possible to bring the eggplant to Hell in multiplayer, there was almost no chance of doing it in single-player. The main problem is that there are two places in the Chain where you're forced to leave the eggplant behind. The first place is the Moai, when you kill yourself with the Ankh—your carried item does not respawn with you. The second place is at the beginning of the Temple, when you'd need to carry both the Sceptre and the eggplant into the next level. In multiplayer, another player could carry the Sceptre for you, but in single-player, the only possibility was to let a computer-controlled helper called a "hired hand" carry it instead.

I originally added the hired hands to stave off the inherent loneliness of being the hero in video games. Even when AI companions get in your way, as they often do, it feels nice to have someone along for the ride. In *NetHack*, you started with either a pet kitten or puppy, and while it seemed more trouble than it was worth, I always tried to keep them alive as long as possible.

If you've played *Spelunky*, you know how finicky the hired hands can be, how easily they accidentally kill you or themselves, and how frightening it is when they get a hold of weapons. In truth, it was an incredible feat on Andy's part to write AI that can navigate *Spelunky*'s

twisted levels without resorting to the manic, bouncy algorithm that the angry shopkeeper uses. The basic system was robust enough to use in Deathmatch, to allow a single player to fight computer-controlled opponents.

But regardless, giving the hired hand a Sceptre is like giving a toddler a loaded gun, and having him follow you to Hell with an eggplant is like asking a toddler to follow you down the stairs with a porcelain vase. The eggplant itself is fragile—any kind of impact short of placing it gently on the ground will instantly destroy it. On top of that, the chance of reaching the Temple with a hired hand and an eggplant is slim. It requires many things that have a small chance of happening to all happen together: a shop with a mystery box in the same level as an altar and another shop selling hired hands, all appearing after the Moai. So before one could attempt an eggplant-to-Hell run, it could easily take thousands of long runs to get the ideal conditions. In cooperative play, on the other hand, a gift box can be generated by purchasing any item in any shop, making it much easier to sacrifice one on an altar.

For the year that we worked on the PC port of *Spelunky*, it's unclear whether anyone ever ventured to Hell with an eggplant, even in co-op, but once the game hit PC, it was easy for players to hack into the game's files and find "monstersbig.png." From there, a YouTube user named "chaindead" used a cheat engine to give himself the eggplant in Hell, much like how I tested the scenario in the first place. In a video titled

"Spelunky HD easter egg - eggplant can be used on King Yama's head," chaindead tossed the eggplant onto Yama's floating head and revealed the big secret. Two months later, another player and his brother completed the task legitimately for the first time ever, using the cooperative method as described above.

One might think that a single-player eggplant run would have taken years to complete, but that would vastly underestimate the tenacity and intelligence of *Spelunky*'s best players. It only took one month before the Twitch streamer Bananasaurus Rex, with the help of other prominent members from the *Spelunky* community, was able to do what I had thought required either cheats or leprechaun levels of luck. It was the holy grail of *Spelunky*.

The solo eggplant run.

The first thing you notice when watching the run on YouTube is that Rex obtains an eggplant in the second level of Mines, which seems like a waste, since he'll lose it once he sacrifices himself in the Moai level. But actually, Rex always gets the eggplant there, restarting the game over and over again until he obtains a level with a mystery box and an altar. He calls a run where this level appears an "eggplant seed," referring to the number, called a "seed," which is used to initialize a random number generator like the one used in *Spelunky*.

But before I explain how Bananasaurus Rex completed the first ever solo eggplant run, I want to talk about glitches. A glitch in a video game is a type of bug,

and while the two terms are sometimes used interchangeably, a glitch is less harmful and more like a hiccup. If, for a brief moment, a character's head began spinning around rapidly on its neck, that would be called a glitch. The most famous glitch is probably "Minus World" in the original *Super Mario Bros.*, so-called because it allows the player to access an infinitely-repeating underwater limbo that displays "World -1" before it starts. In fact, the game is trying to load "World 36-1," i.e. the first level of World 36, which doesn't exist. Getting to Minus World involves exploiting a collision glitch in World 1-2 that lets Mario slide through the wall and enter the warp zone before it's able to load the correct warp data. By entering the leftmost or rightmost warp pipe before the data is loaded, you'll be warped to Minus World.

Not all glitches are useless, though, and many high-level challenge runs involve exploiting glitches that enable the player to complete sections of the game out of their intended order (called "sequence breaking") or to bypass them altogether. Because of how useful they are in speed running and other challenges, these glitches are highly sought-after by speedrunners and streamers, and are given special names to make discussing them easier.

In Bananasaurus Rex's solo eggplant run, several glitches I had previously never known about were exploited in order to drastically reduce the amount of luck involved in reaching Hell with the eggplant. The first hurdle is the Moai—getting inside it usually requires giving up your carried item, meaning that the

eggplant can't be obtained until after this point. Given that shops are less likely to spawn in later levels, waiting until after the Moai to obtain an eggplant would have stacked the odds against Rex and made finding an eggplant seed tedious.

My original intention was that there would only be two ways of getting into a Moai: by sacrificing yourself with the Ankh or by using a teleporter. But if you're holding a teleporter, you can't also hold the eggplant. Nor will the Hedjet spawn inside the Moai that way—a diamond will spawn in its place. What I didn't realize, though, is that there is a third way to get in.

In September, a few months before Rex's historic run, BaerTaffy and a streamer called bisnap discovered a strange glitch involving a special item known as the ball and chain. The ball and chain can only be obtained by destroying two of Kali's altars, causing her to inflict them on you as punishment. The ball is attached to your leg by the chain, impeding your movement while it drags along the ground, but it can be picked up like a normal item to make traveling easier. Unfortunately, that also prevents you from carrying any other items.

In early tests of the ball and chain, it was easy to get stuck if you jumped off a ledge while the ball was trapped above you. To circumvent this, we added code that would make the ball break the floor tile underneath it if the player was hanging below the ball. What we didn't think about, however, was the possibility that

the ball could also be used to break up the otherwise invincible Moai. As it turns out, the code where the ball and chain breaks floor tiles never checks to see if the block has any special properties, like the "invincible" flag that we mark the Moai blocks with! The result is that any floor tile the player can hang off of with the ball and chain can be broken.

With a ball and chain, then, it's possible to break into the Moai, put down your eggplant, kill yourself, regenerate using the Ankh, spawn the Hedjet, pick up the eggplant again, and carry it into the exit. Done! Or rather, easier said than done, since you still had to destroy two altars and then juggle the heavy iron ball and fragile eggplant from level to level, something that few people besides Rex could do. Fortunately, once the Hedjet is acquired, there's no further need for the ball and chain, so Rex exploits another glitch to destroy the ball: By "dipping" it off the bottom of the screen in the Ice Caves, he makes it disappear into the bottomless pit.

The next problem is bringing the Sceptre from the first level of the Temple into the second level to unlock the door to the City of Gold. This is another place where you have to leave the eggplant behind unless you have help. Unfortunately, shops that sell hired hands are fairly rare—not rare enough to be a problem on its own, but rare enough that you wouldn't want to multiply the chances of one appearing by the chances of getting an eggplant seed.

Rex's solution to this was hiding in plain sight and didn't require exploiting glitches, but it was unconventional to say the least. Although everyone, including myself, had assumed that the only way to get an eggplant from 4-1 to 4-2 in single-player was to enlist the services of a hired hand, there was one other method: Have one of the sixteen unlockable player characters with you to hold an item instead. In *Spelunky*, unlockable characters are discovered hiding in coffins at various locations throughout the game—some randomly, and some not. Once freed, they act exactly like hired hands, since they share the same AI code. The reason I had never considered using an unlockable character for a solo eggplant run, however, was because they never appear again outside of the run where they're freed.

It was brilliant. All Rex had to do was reset his save file before attempting the solo eggplant run, setting all of the previously unlocked characters to locked again. After that, there were two characters who were guaranteed to spawn before the City of Gold. One was Meat Boy, who appears in the Worm, and the other was the robot, who appears in the special Mothership level that can be accessed at the end of the Ice Caves. Between the two, the robot is the obvious choice, since being able to enter both the Worm and the Black Market is not guaranteed and the Worm is a particularly difficult level to navigate. Not that the Mothership is easy—it's filled with aliens and projectiles and is another level where you have to travel upwards to reach the exit. But Rex

clearly shows that if you have the skill it's possible to clear it with the robot in tow.

There's even more to the run that I haven't talked about here, from deft maneuvers and close calls to missed opportunities. It climaxes with Rex making it to Yama with only one health. In a Polygon article about the run titled "A breakdown of 2013's most fascinating video game moment," Doug Wilson writes:

> One of the most endearing (and enduring) qualities of *Spelunky* is its many secrets. The game is brimming with bonus levels, obscure items, rare music, Easter eggs, references to older video games and more. These kinds of details are more than just dressing; they give the game its personality. Secrets, when done right, are one of the most powerful ways of humanizing a game. They provide a form of texturing, a lasting trace of the creator's hand. Secrets, at least the truly memorable ones, transform a well-designed game into something far weirder, something magical.

Secrets are also a way for players to connect with one another. I think back to that first beta release of *Spelunky Classic* and the way TIGForums members slowly pieced together the game for the first time. In a similar fashion, the streaming and Let's Play communities have worked together to discover secrets in the remake that even Andy and I didn't know existed. These experiences have brought the game to life for us in a whole new way.

Some creators out there might be embarrassed about players discovering and exploiting glitches in their game, but as long as the glitches aren't ruining the experience for average players, there is nothing more exciting. For me, video games have always been defined not by what is known but what is unknown. It's the trepidation of stepping into that first dungeon or sinking into my first warp zone. It's seeing what would happen if I freed a troll from a trap, or if I could gorge myself to death on food. It's coming across a file called "FIFI.EXE" on an old floppy disk and wondering what kind of game it might be. It's debating with my friends during recess whether you could really fight Sheng Long in *Street Fighter II*.

It's why, even though only a small percentage of people will ever reach Hell, we put as much effort into that area as any other area in the game.

You never know what will make it into the history books, though. I added Yama's eggplant transformation almost as an afterthought, but it became one of the most important additions to the game because of Rex's incredible run. He showed me that something that I had created was much bigger than I imagined—the limits I had put in place were not really limits at all.

New Adventure

Gaming is such a different landscape than it was when Jordan interviewed me for TIGSource in 2006. Back

then, I couldn't fathom how a game developer could make a living without joining a big company and working their way up the ladder. It seemed like the days when a few friends or family members could change the world from their bedrooms was over. Little did I know that while I was expressing my pessimism to Jordan, seeds were being planted by a nascent indie games community that would bring back bedroom game-making and turn it into the expansive network that it is today. Games made by one or two people now share the same spaces as games made by companies of thousands and that's the way it should be—a multicolored, textured landscape.

Like all of my game projects, *Spelunky* represents more than just another game released. It also represents a period of my life: the end of my time with Alec making *Aquaria*, a return to solo game development, and a partnership with Andy that was put into motion nearly two decades ago. When I started *Spelunky*, I was sitting in a pile of puzzle pieces strewn onto the floor—my thoughts, experiences, and inspirations—and not knowing what kind of puzzle I was putting together. All I could do was start finding pieces that fit. Now that *Spelunky* is done, what I feel most of all is a sense that I'm part of an even bigger puzzle that includes the people that influenced me, the players who play *Spelunky*, and whomever *Spelunky* has influenced in turn. I'm proud that great games like *FTL: Faster Than Light* and *The Binding of Isaac* exist in part because of *Spelunky*. In

the end, isn't that why we create things? Not just for the power of putting something into existence, but to connect with people and be part of the conversation that is human history. To have something that speaks for us when we're not speaking and even after we're gone.

When I started *Eternal Daughter*, I was 18. When I started *Aquaria* I was 23. When I started *Spelunky*, I was 27. Now I'm 33 with a wife and daughter, and I'm rifling through my puzzle pieces again. The pile is bigger now and I'm struck with the same feelings of excitement and wonder that I have at the start of every new project. In front of me are connections not yet made, worlds not yet created, and secrets not yet discovered.

Until next game!

NOTES

PUZZLE PIECES

The game I met Alec through was *I'm O.K.: A Murder Simulator*, a satirical work inspired by Jack Thompson's infamous "Modest Video Game Proposal": bit.ly/1KDt2A2 Alec found out about the project through a comment on the technology website Slashdot.

GAME MAKER

In a 2000 interview with Firing Squad, John Carmack explains how his "love for programming is a more abstract thing" than a love for game programming specifically: bit.ly/20vI9NM

ROGUELIKE

Jay Fenlason explains the origins of *Hack*'s name in David L. Craddock's 2015 book *Dungeon Hacks: How NetHack, Angband, and Other Roguelikes Changed the Course of Video Games*. An excerpt of the book that contains the relevant passage was posted to Gamasutra: ubm.io/20vIccq

Mark Johnson's "Maniac's Ascension Guide" from 2002 is here: bit.ly/1KO07JB

"Turns" in *NetHack* are more complex than just "player actions." Most player actions (not including checking one's inventory) take a single turn in *NetHack*, but if the player's character is fast enough, they can sometimes move without incrementing the turn counter. Or if they fall asleep or become paralyzed, several turns can elapse before you can act again. You can read more about *NetHack* turns on the *NetHack* wiki here: bit.ly/1TKfvcI. *NetHack* players have also tried converting *NetHack* turns nto a real-life time unit. Using caloric intake and expenditure as a basis for their conversion, they came up with something called "Correlated NetHack Time" (CNT). With CNT, we can work out that Maniac's "fast ascension" of 40,000 is around three and a half days spent questing for the player's character: bit.ly/1STyq5a

EXPLORER.GMK

John Szczepaniak's 2008 write-up of *Kagirinaki Tatakai* for Hardcore Gaming 101: bit.ly/1TiDAqP

Miyamoto reveals how a "good idea" solves multiple problems at once in this 2010 interview with Eurogamer: bit.ly/1mzqJlS

RANDOMIZED LEVELS

Darius Kazemi, who wrote *Jagged Alliance 2* for Boss Fight Books, created a fantastic tool that allows you to generate *Spelunky* levels in your browser and see exactly how it works: bit.ly/1XqEqkE

TIGSOURCE

You can read my full 2004 interview with Jordan Magnuson here: bit.ly/1KhH5LL

FEEDBACK LOOP

The public announcement of *Spelunky* on the TIGSource forums that I made on December 21, 2008: bit.ly/20vIBvw

INDIFFERENCE

Miyamoto's lake story is taken from *Game Over: Press Start to Continue: The Maturing of Mario*, a 1999 book by David Sheff.

The 2003 interview with Swedish magazine *SuperPlay* was translated into English by a member of the IGN forums and can be read in its entirety here: bit.ly/1XqEtNj

Tevis Thompson's "Saving Zelda" article is available here: bit.ly/1TiDHmu

The details of Miyamoto's cave story differ with each retelling. In a New Yorker profile, the author claims that Miyamoto found a "hole" in the ground and came back the next day with a lantern, only to discover that the hole opened up into a cavern: bit.ly/1k74cv8. Most other versions emphasize that he hesitated before going in. Sometimes there is another cave inside the cave. Not all versions mention the lantern. And so on. So far, I haven't come across the story told in Miyamoto's words, but the spirit of the story remains constant each time it's told.

THE JOY OF PLAY

Jeremy Parish's article "Learning Through Level Design with Mario" was posted to 1UP.com in 2013: bit.ly/1PvsxHd

EXPLAINING THE VISION

Chris Hecker summarizes his 2010 GDC talk, titled "Are Achievements Considered Harmful?": bit.ly/1TfSmgS

IMPROVEMENTS

Jamey Pittman's "Pac-Man Dossier" breaks down *Pac-Man*'s inner workings, including its AI, in breathtaking detail: ubm.io/1SA6t1U

Tom Francis's "Shopstorm" story from 2012: bit.ly/1PU2fzI

EIRIK

Eirik's 2011 blog post where he explains the inspirations for *Spelunky*'s music: bit.ly/1QzqH5F

FINISHING A GAME

My 2010 "Finishing a Game" article: bit.ly/1jpZxjT

Andy gave a 2014 talk at NYU's Game Innovation Lab called "Freefalling Through the Goldilocks Zone" about finishing an indie game using just the right amount of technology to move quickly without getting bogged down by engine-building. You can watch it here: bit.ly/1SNJKQ5

MULTIPLAYER

Alex Navarro's 2012 review of *Spelunky* for Giant Bomb: bit.ly/1RAwYTJ

"Someone thought turning on 99 bombs in Spelunky deathmatch was a good idea. They were right.": bit.ly/1oAvlK8

THE DAMSEL

The quote from TIGForums member Peevish about the sexism in *Spelunky Classic* can be found here: bit.ly/1XqEP6J

Evan Narcisse's Kotaku article "Spelunky's Best Feature Lets You Rescue Whoever You Want": bit.ly/1SNJNeJ

The transcript for Feminist Frequency's "Damsel in Distress (Part 3) Tropes vs Women" video is on their website, along with a link to the video itself: bit.ly/1TfSFbn

LAST 10%

Tom Cargill's 90-90 rule was quoted/popularized in Jon Bentley's "Programming Pearls" column in the September 1985 issue of *Communications of the ACM*.

EXPECTATIONS

The Gamasutra article by Ryan Langley where he compares *Spelunky*'s first-month sales to *Fez* and *Braid*: ubm.io/20vJa8A

Ron Carmel's "Is XBLA Past Its Prime?" article from 2011 where he surveys 200 indie game developers about their experiences with the major platform holders: bit.ly/211HSnY

Kyle Orland covers Team Meat's GDC 2011 talk on Gamasutra: ubm.io/1XqEX66

IGF 2012 Jury feedback that includes the lovely summary about *Spelunky*: ubm.io/1QzqVdd

RAISING THE STAKES

Doug explains his daily *Spelunky* ritual with Nifflas on his blog in 2012: bit.ly/1mzrono

Hideo Kojima describes his "disposable game" in this 2003 interview with GameSpy: bit.ly/1PU2x9L

Here's the "Why Game Developers Love SPELUNKY!" video with me and Doug: bit.ly/1oeDvrL

BEYOND THE CITY OF GOLD

The stats page on Steam that shows what percentage of players have earned each achievement in *Spelunky*: bit.ly/1KO0RhH

AN EGGPLANT'S CHANCE IN HELL

The video where chaindead hits King Yama in the head with the eggplant: bit.ly/1LnkuYT

BaerTaffy breaks the Moai for the first time in this video: bit.ly/1TiEhQZ

Doug's amazing play-by-play explanation of Bananasaurus Rex's 2013 solo eggplant run: bit.ly/1cNzOfb

The run itself: bit.ly/1TZJahp

ACKNOWLEDGEMENTS

Another secret: In the lower right corner of the Character Select screen, there is a little heart carved into the rock with the initials "FF" in it. This is for my wife Frances, whose keen eye I've relied on for almost a decade now. Thank you for being there to talk things through with me and for challenging me to do my best. Thanks to our daughter Caroline for being very understanding when Dad had to work on his book instead of playing. Mwa.

And to my mom and dad, for obliterating that Asian parent stereotype by always being supportive of my passion for drawing and video games. Thanks to my dad for giving me feedback on each and every draft of this book.

Spelunky would not have ever gotten a reboot without Jonathan Blow's generosity and Kevin Hathaway's belief in the project. Thanks to Edmund McMillen and Tommy Refenes for all of the advice and support while we were finishing *Spelunky* on XBLA. Thanks to Doug Wilson, Zach Gage, and Nifflas for their enthusiasm and for coming up with the Daily Challenge idea. Thanks to Nick Suttner and the Sony team for bringing the game

to PlayStation. Thanks to BlitWorks for doing such a fantastic job on the PSN port—I'm looking forward to working with you guys again in the future!

And the game wouldn't be what it is without Herculean efforts from Andy and Eirik. It would take many more pages to outline all of their contributions to the game. Thanks to Jon Perry for giving me great feedback on the book early on and for his work on the *Spelunky Classic* soundtrack. Thanks to George Buzinkai for his songs, as well. And to Chris Yamaoka, who was my roommate for most of *Spelunky*'s development: thanks for helping me look over contracts and also keeping the bathroom clean!

Thanks to everyone on the TIGSource forums who playtested the very first version of *Spelunky*, and the fans on the *Spelunky* forums. Thanks to Fangamer for selling the *Spelunky* figurines and developing new merchandise for us. Thanks also to Mike Dailly for his work on Game Maker and GameMaker: Studio.

Thanks to all the streamers, speedrunners, and record-breakers that have pushed past the limits of the game and made it new for me, including Bananasaurus Rex, Kinnijup, BumCommando, Rud-digger, Krille, Pibonacci, and Kenneth Carrillo. Thanks to Northernlion, MANvsGAME, Foxman, BaerTaffy, and RedPandaGamer for their early support of the game and for helping us show off the Daily Challenge. Thanks to Andrew Goldfarb for bringing many of their exploits to my attention and to the *Spelunky* Reddit community

for providing a home for speedrunners and new players alike. To anyone who's played *Spelunky*, whether you've streamed it or not—thank you for helping me make the world come to life.

Thanks to Gabe Durham for inviting me to write about *Spelunky* and guiding me through my first book with great patience. Thanks also to Michael P. Williams for the feedback and suggestions. Thanks to Jim Fingal for reading, Ryan Plummer for copyediting, Joseph Michael Owens and Nick Sweeney for proofreading, and Christopher Moyer for layouts. You don't realize how much you have to learn about the art of writing until you have professional writers edit your book. And to Ken Baumann for the cover design, which is so much more clever than anything I could have come up with.

Thanks to all of the creators that I've mentioned throughout this book. It's been a pleasure participating in this conversation with you.

SPECIAL THANKS

For making our second season of books possible, Boss Fight Books would like to thank Ken Durham, Jakub Koziol, Cathy Durham, Maxwell Neely-Cohen, Adrian Purser, Kevin John Harty, Gustav Wedholm, Theodore Fox, Anders Ekermo, Jim Fasoline, Mohammed Taher, Joe Murray, Ethan Storeng, Bill Barksdale, Max Symmes, Philip J. Reed, Robert Bowling, Jason Morales, Keith Charles, and Asher Henderson.

ALSO FROM BOSS FIGHT BOOKS

1. *EarthBound* by Ken Baumann
2. *Chrono Trigger* by Michael P. Williams
3. *ZZT* by Anna Anthropy
4. *Galaga* by Michael Kimball
5. *Jagged Alliance 2* by Darius Kazemi
6. *Super Mario Bros. 2* by Jon Irwin

7. *Bible Adventures* by Gabe Durham
8. *Baldur's Gate II* by Matt Bell
9. *Metal Gear Solid* by Ashly & Anthony Burch
10. *Shadow of the Colossus* by Nick Suttner
11. *Spelunky* by Derek Yu
12. *World of Warcraft* by Daniel Lisi

★ *Continue? The Boss Fight Books Anthology*