GAMASUTRA
The Art & Business of Making Games

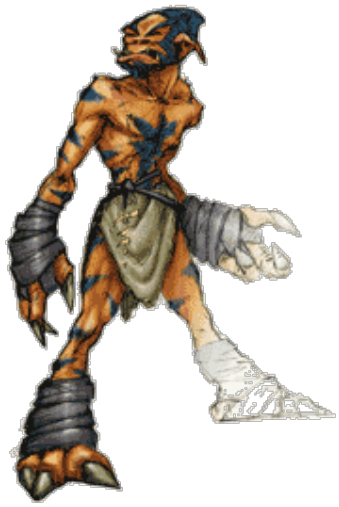
PRINT

Postmortem: Gas Powered Games' *Dungeon Siege*

By Bartosz Kijanka

The development of *Dungeon Siege* was about much more than the RPG you may have played by now*Dungeon Siege* was a Herculean effort by a small group of people who simultaneously started Gas Powered Games, built their first RPG, and made a hit game.

Prior to his career as a prophet, Chris Taylor, Gas Powered Games' president and founder, was responsible for the creation of both the hit RTS game Total Annihilation, as well as Cavedog Entertainment, from which it came. For reasons that are now obscured by the mists of time, Chris was inspired to try his hand at something new, and Gas Powered Games was born. A number of us who worked with Chris in the past, primarily on Total Annihilation, had similar passions, and it wasn't long before we found ourselves cozy once again in a closet office in Kirkland, Wash. We had just finished work on a successful RTS game, a genre we loved working on, but for a number of reasons we opted to try a different genre. We knew we were skilled enough to make another RTS, but the world was swimming in them at that time, so it seemed like a good opportunity to try something new. We all liked the fantasy aesthetic and there were very few good RPG games at that time, so somehow doing an RPG seemed like the natural choice.



We were starting completely from scratch. We had to find an office and buy phones, fax machines, desks, and all those little things most of us take for granted working at an established company. Our first engineering meeting was held at Chris's house with a whiteboard on the floor and a half dozen of us huddled around it. A month later, when we finally did find a space, we had to make it usable. I remember Chris crimping network cables and climbing up on ladders through ceiling tiles running the network wire in the new office. I also clearly remember my first day in that office, when I built my own PC from random parts we bought, unfolding a cheap Costco table on which to set it all up. I loaded up MS Dev and faced an empty header file, which at that moment represented our entire technology base.

Chris repeatedly told us we were in for a wild ride, but I don't think that prophecy sank in until years later. So merrily and naively, we set out on what turned out to be a four-year journey that was as challenging as it was educational.

What Went Right

1.Exceptional team. It's clear to me that the single best thing thing about*Dungeon Siege* is the team that created it. So many things are right about our team that it's difficult to elaborate on this without sounding as if I'm gushing. From the beginning, Chris Taylor set the tone for the company with strong values and an outrageous personality. Chris might be best described as a force of nature; he can help an old lady cross the street, tell you a joke that will make you seek psychological counseling, make a shrewd business transaction, and convince you that an impossible feature is actually easy, all in the span of about five minutes. He is certainly the hardest-working and most driven person any of us know. He instills the kind of respect that is pivotal in binding a team into a single, coherent unit.

The team itself is consistently calm in the face of pressure, persistent in the face of adversity, and absolutely dedicated to success. I have never worked in a group of so many selfless, dedicated, and positive people. At times we can be so focused and single-minded that by some definitions we might be considered a cult.

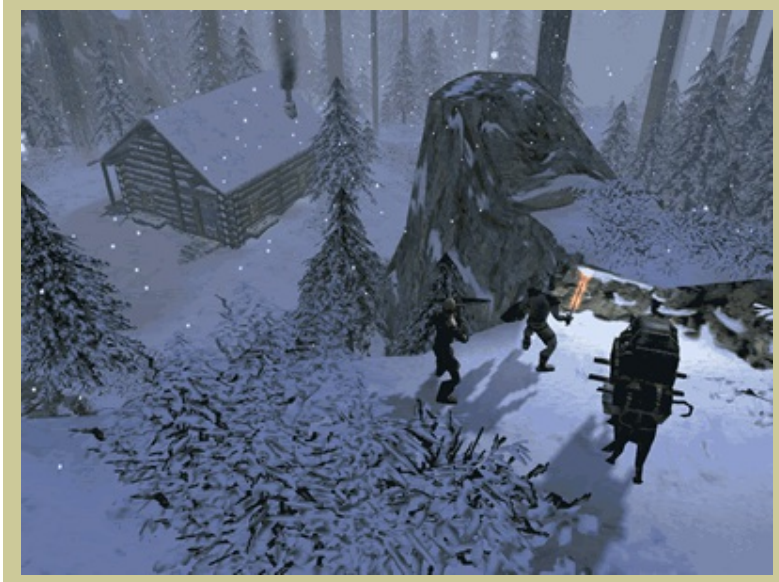
Another positive trait that contributed to the team's efficacy was that we shunned any kind of classism. At GPG, we have a culture where everyone is valued. We don't single out contractors or junior people as different, everyone's ideas get heard, and it's implicitly understood that every single person plays an important role in creating the game.

2.Exceptional art. Somewhere between the prototype and our first E3, something magical happened. It may have been when our technology started to hobble along such that people could finally see what the game might look like, or maybe it was when our tools became really usable. For whatever reason, since our first E3 and for about three years thereafter, the game just kept looking better every day. Art director Steve thompson and his team straddled an exponential curve of outdoing themselves and rode it to the very end. Over time the models morphed from "Hey, is that a bear or a giant rat?" to "Wow, let's see that again!"

Similarly, the level designers did a remarkable job. As the editor settled, their productivity skyrocketed. Near the end of production the volume of their output was stupefying. I'm still shocked when I think of our multiplayer world, which is far larger than the already large single-player world, and how it was built in a tenth of the time. Amazingly, as the level design team picked up speed, the quality of the output and attention to detail also improved.

From very early in the project we had a flexible effects system that was only marginally used because we were short of people, so not many assets took advantage of it. Late into the game's development, the art was already quite far along when we had an additional and surprising growth spurt. Eric Tams, our overworked content engineer, went ballistic and added so many special-effects embellishments that we were, quite frankly, astonished. Suddenly, the effects system was working overtime as swords were flaming, staffs were sparking, and so many other things we don't have names for were happening. It was a nice surprise, and one example among many of team members working with inspiration to make a difference. This sort of inspiration really helped the game evolve on all fronts.

3.Extreme flexibility. As a small company our flexibility is a distinctive advantage over a large company. Compared to what I've seen of many large companies, we can make important decisions 100 times faster. If you're trying to innovate, the ability to make decisions quickly is absolutely critical. At a large company, I've seen people discuss a minor issue for days on end. At GPG, if we need a resource, or if we've identified a project course correction, we meet with Chris Taylor and most of the time we will decide upon and commit to a specific course of action in under five minutes - very refreshing.



One of many serene alpine environments.

One of the more extreme examples of this sort of flexibility was our temporary test team. Deep into the debugging stage of the project, we were experiencing increasing frustration with the process. Up to this point, Microsoft, our publisher, was doing most of the testing, and although they were on-site for some time, they had moved back to the mother ship, complicating matters. Try as they might, the scope of the game was simply enormous. We needed more help. We realized this on a Friday, and by next Monday morning we had configured our own test lab. By the end of the week the lab was fully staffed and running double shifts.

We could be this flexible as a company only because our teammates were this flexible. Everyone carried multiple responsibilities, and some of us carried so many it was hard to keep track. As one example, Jacob McMahon wore the hats of VP, designer, producer, HR, accounting, bill paying, payroll, legal, and who knows what else. He also managed to place all the monsters in the game and gameplay-balance the entire thing, while only occasionally speaking in tongues. Such effort is representative of what we had to do in order to succeed.

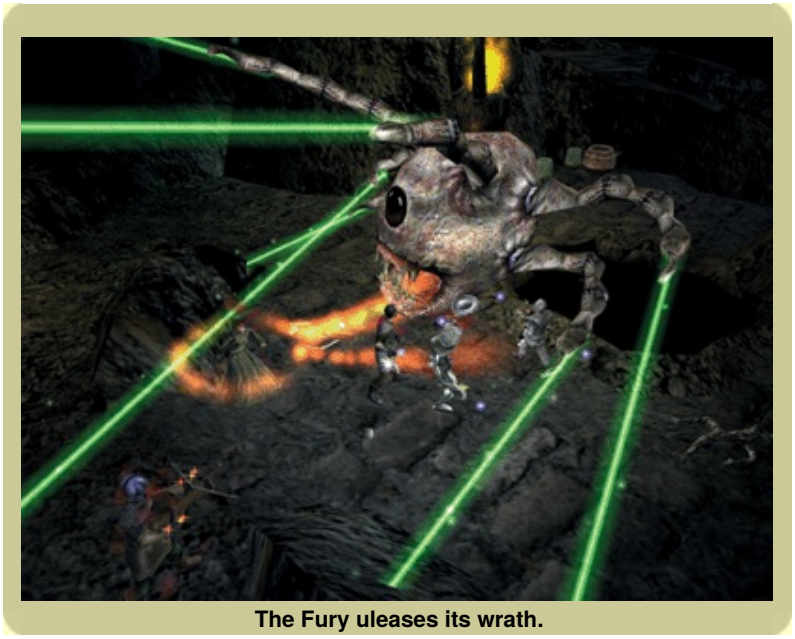
4.Solid architecture. The engine underwent numerous architectural changes during development, and ultimately we were left with an engine that's both powerful and flexible. Because development included a lot of research, the architecture was forced to evolve through repeated reengineering to support new discoveries as well as new requirements.

Based on our experiences during the development of Total Annihilation, we knew even before we wrote the first line of code that we had to focus on a data-driven design. So although much changed about the architecture over time, the critical concept and goal of a data-driven engine persisted. Today it seems more fashionable to take this approach, but four years ago there didn't seem to be much focus on this issue. At that time, many game engines would inevitably stumble onto being data-driven, but it seems that few game engines were doing this

in a premeditated and planned fashion.

Our goal of data-driving the *Dungeon Siege* engine was to support all the features of an action RPG but not hard-code how these features interact in order to create the look and feel of the game. Aside from the actual art assets, the look and feel of the game is defined by a combination of configuration, text files, and scripts. We used a general-purpose scripting language for AI, animation control, and spells, and a specialized language for scripting special effects.

The scripting system, Skrit, played a key role in the architectural success of *Dungeon Siege*. Written by Scott Bilas, it is implemented unlike any other scripting system we've seen to date. The language itself is conventional, but extending it by exporting engine functionality is easier than ever. Exporting a function is as simple as adding a single tag in the C++ code. Additionally, because Scott had to create a powerful back end to support Skrit, it became the basis for a remote procedure call (RPC) mechanism that made implementing multiplayer much easier. We leveraged the same technology for save and load and for many other systems.



The Fury uleases its wrath.

Based on another early architecture decision, the editor was essentially built on top of the game. Specifically, it was built on top of what we called the "world" layer of the game. The world contains all of the game mechanics, less the user interface. The world layer has no knowledge of a user interface, or even high-level gameplay goals such as quests. It's simply the game environment containing the map and all its life forms. The actual game executable is built by using the world as a foundation and adding a game component above it. The game component gives you the user interface, creates your hero, and has all the other trappings required to present the gaming experience to the user.

Alternatively, the editor is simply the world layer with an editing component on top instead of a gaming component. The editor component allows you to create and manipulate the world, whereas the gaming component restricts you to a particular experiencing of it. The benefit of this approach was that the editor took advantage of much of the work we did for the game engine. The drawback was that a careless change in the world layer for the sake of the game component could break the editor component. Chad Queen, who built the editor, was relentless in keeping the editor running such that this vulnerability wasn't ever much of a problem.

5.Instant messaging. We faced the typical set of communication challenges that plague most development teams. However, we had at least one success worth mentioning.
Consider the typical means of communication in the office: Normally when you urgently want to speak to someone, you call them. At other times, when you need a more formal and structured medium for communication, you e-mail someone. And sometimes, when you're sitting next to someone and just want to ask a quick informal question, you look over your shoulder to see whether they're in a good place to be interrupted.

In our first year, as a small group of fewer than a dozen people, we worked on the same floor and essentially in the same space. Because of our proximity, all of our informal communication happened in person. (It's amazing how much important work is facilitated through informal communication.) But when we started to grow we found ourselves in a communication conundrum. When engineering moved to a different floor of the building, that informal channel of communication with the other groups in the company was lost.

After a brief period of disorientation we identified the problem and called upon ICQ instant messaging to the rescue. It's a powerful tool and at times can actually be more effective than informally speaking to someone. Because ICQ is so informal, you can be very direct without being perceived as terse. You simply type in your question and you're done. No chitchat or small talk needed, no pressure to introduce yourself or your topic gracefully.

Instant messaging works very well in conjunction with formal e-mails and semiformal phone calls. It allows us to direct without offending

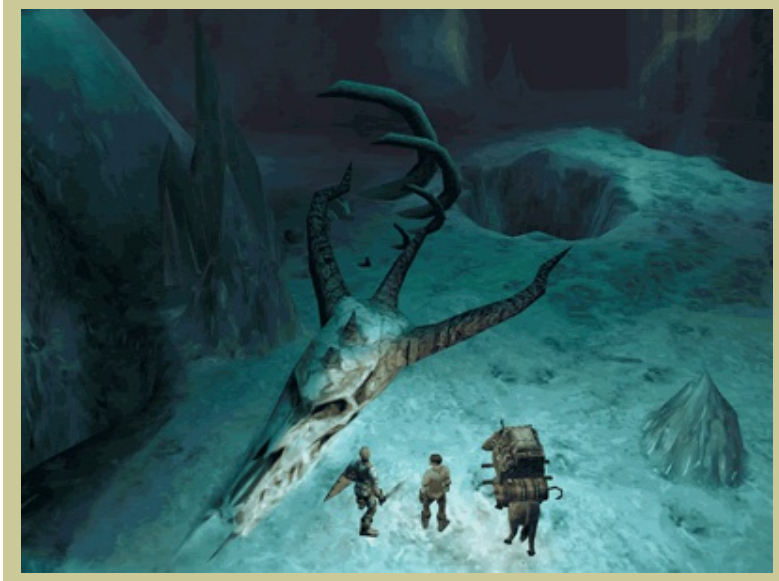
anyone and is more efficient than waiting in line to speak to someone. If it's not a good time, the recipient of your message can simply ignore your request until his or her next convenience. Instant messaging had such a positive impact on our work that I can't imagine ever working without it.

What Went Wrong

1.Extreme ambition. Extreme ambition can be empowering, but it can also bring much pain. A large portion of Gas Powered Games consists of extraordinarily ambitious people. Every-one has their own reasons, but collectively, we all want to win, and want to win badly. While this drive helped us persevere through some of the challenges we encountered during development, it also created some new ones.

Our ambition translated into a number of painful symptoms such as feature creep, over-optimism, and a project scope that was ultimately larger than our ability. The problem was compounded by the fact that nobody on the team had really worked on an action RPG before. We looked at the "leading brand" RPG and said to ourselves, "Hey, this is a fun genre. Let's make something like this except, much, much bigger, in 3D, without loading screens, and push the technology to its absolute breaking point!"

Clearly we couldn't have imagined the sheer amount of work required to build the kind of RPG that we wanted. And we wanted all kinds of interesting things: lip-synching, cooperative networked level editing, dual monitor support, wavelet terrain compression, deformable terrain, and many other gems. Some of these features ultimately didn't fit into the vision of our game, while others were simply extraneous. To our credit, we killed most of these before even starting development, but we were left with many others to be enthusiastic about for months.



Early explorers Fedwyrr and Klars discovered much of the route traveled in *Dungeon Siege*

Our ambition when mixed with our pride also set us up for some hard lessons. The first year of Gas Powered Games was the honeymoon period. Everyone was full of ideals and had the boundless energy required to discover their inevitable pitfalls. Many of us were excited to have shed the shackles of Big Brother software companies and finally be in charge of our own destinies. No more lame meetings, no more stupid schedules, no more dog-and-pony shows. We were not going to adopt any of the stale, dated, and oppressive habits of game companies past.

Four years later, we have a deep appreciation for organization. Chain of command, ownership, discipline, and planning are things that we hold in very high regard and are further developing for the future. We still don't like Big Brother, but we realize that Little Cousin may help us get more work done.

2.Aborted efforts. We spent too much time exploring dead ends and implementing unnecessary features. The game's basic gameplay principles and aesthetic goals remained consistent from the beginning of the game to the very end. Unfortunately, although we were clear on how we wanted the game to look and feel, the technical requirements weren't obvious. We were feeling optimistic, and anything and everything sounded like a good idea at the time. Design meetings felt like a shopping spree at the supermarket. We loaded up the cart and set out on a rather long road of trial-and-error punctuated with discovery.

The animation editor we attempted to build is a good example of the trial-and-error approach. At the time we had a developer who was both skilled and extremely passionate about creating our own animation tool. Given our ambitious state of mind, we were easily sold on it, and the developer spent a better part of a year working on the tool. Unfortunately, a year into it his calling took him elsewhere, and the editor work came to a screeching halt. This serious setback forced us to reconsider the necessity of this custom art tool. Almost immediately we reexamined what was possible with 3DS Max as senior programmer Mike Biddlecombe dove in to create a few plug-ins that did the job beautifully. Lesson learned: always use third-party tools when they're available.

Although the aborted editor is our poster child of inconclusive work, there were many other technologies we spent a lot of time on but didn't use. For example, we started on OpenGL but ultimately moved to Direct3D. We also wrote level-of-detail systems and sophisticated collision algorithms that we ultimately scrapped.

3.Complex engine. We have created a powerful but ultimately complex engine. To our credit, the engine is stable, relatively well commented, and the source is consistent in terms of coding standards. However, the engine is sufficiently complex to make debugging and maintenance times often longer than expected. Complexity arose from several factors.

First, the engine is large enough that although in the local sense things are documented, in the global sense there are implicit rules that are not. This unfortunately translated to a larger number of nonobvious bugs. The nonobvious bugs tended to be in the "incorrect result" category rather than in the critical program-failure category, which made them harder to track and slowed debug times. The implicit rules in the system made it more likely that a seemingly simple change might create an unwanted side effect down the road even if we did get correct behavior in the short term. The best examples of this problem were single-player changes that worked absolutely great but broke something in multiplayer or simply failed to work in multiplayer.

Additional complexity can be attributed to a few elements of the engine that should have been abstracted but weren't. One example of this is our node-based coordinate system. The coordinate system reflects the segmented nature of our terrain. Every position primitive includes an ID of a terrain block (node) and a 3D coordinate within that node. Global coordinates aren't valid for more than one simulation, so if you want to work with a point outside of your particular node, you have to do space conversions. This introduced extra steps to every space calculation and increased the risk of human error. We got used to working with this system and shipped the game without abstracting these artifacts away, but we paid a high price in maintenance of affected code. In the future we hope to wrap this up in a cached global coordinate system that would make our vector math look like the math to which everyone else is accustomed.

4.Slipped schedule. When I started this Postmortem, I wanted to avoid talking about the two most common problems faced by development teams, communication and schedule. Every single project seems to have trouble here. I've managed to steer clear of communication rants, but I must mention our schedule, since we made a spectacular game but spectacularly late. Originally we were aiming to finish in just about two years, but ultimately we shipped in almost four.

A panorama of castle Ebb

During this time, however, we weren't just building a game, we were also building a company. We spent a considerable portion of our time simply learning how to be a company and learning how to make an RPG, something that we had never made before. Learning and ambition aside, our schedule first slipped considerably when three out of the six engineers left the company about a year into the project. They each had their own reasons and they didn't leave in unison, but the blow dealt to our planning put the engineering schedule into a permanent state of shock.

After that first year, the turnover at the company was very low, and it certainly wasn't the only culprit regarding the slipping schedule. In order to make a competitive title that was original, we essentially had to invent new features that we would use to compete with existing games. This goal simply required a lot of research, which is inherently difficult to plan or schedule. It also tends to lead to . . .

5.Epic crunch. Our crunch was not your garden-variety crunch. Our crunch was not measured in months, but in years of long hours and few free weekends. It was a crunch of sheer astronomical proportions. The kind of crunch that will make you weak in the knees to think about and give you a thousand-yard stare when you're done with it. The kind of crunch that bites the top of the beer bottle off and spits it out at you before taking a drink.
This kind of crunch can only take root in fertile soil, so we certainly are responsible for our own discomfort. Being the ambitious bunch that we are, not only did we want to make a great game, we wanted to make the best game we could afford, given the time and resources. This line of reasoning, taken to its limit, translated to most of us simply living for the game.

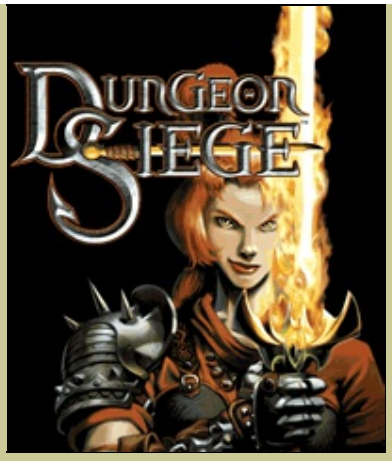
Our ambition was so strong we neglected our bodies, our relationships, and our spiritual well-being. I know many of us did this past a point that most professionals would consider healthy, sane, or possibly ethical. The level of focus on our work was simply amazing, but the sheer self-neglect this fostered was simply irresponsible. We didn't crunch to make up for lost time, we crunched out of uncertainty. Since this was our first RPG, we crunched to implement all those extra features. And at some point we crunched because we could no longer remember doing anything else.

A wise person once said that your passion can become your prison. We were so passionate about making *Dungeon Siege* that we completely lost ourselves in its creation. Having a tremendous desire to succeed is a noble trait, but being unable to reconcile that desire with the actual cost of attaining your goal is not. It's difficult to be critical of ourselves when we all cared so deeply about the game and worked so very hard to build it. But we must remember that we make games. We are toy makers, and we have a responsibility to our own humanity as well as to our trade. We must strive to live balanced and enriched lives so that we may always have inspiration from which to draw.

Lessons from the Dungeon

Dungeon Siege was an extremely challenging project. Four years is a long time to work on a game. By comparison, it's long enough to earn a bachelor's degree in college, or go back for a Ph.D. Four years is time enough for births, deaths, weddings, breakups, and earthquakes. A lot of life happens in four years, and a lot of learning takes place.

Clearly, we walk away from *Dungeon Siege* with valuable experience, both from what we did well and from our missteps. With experience comes perspective, and our perspective is much broader now than when we started. For better, for worse, and for everything in between, we will feel the reverberations of this experience for a long time to come.



Publisher: Microsoft

Number of Full-time developers: 27 at ship date

Number of Contractors: 5

Length of Development: 3 years, 8 months

Release Date: April 5, 2002

Platform: PC

Development software used: MS Dev C++, 3DS Max with Character Studio, Visual SourceSafe, CodeWright, ICQ, RAID (bug tracking), Photoshop, Excel

Development hardware used: Ranged over course of development from 400-1000MHz CPUs with128-512MB RAM

Notable Technologies: Bink, Miles, SmartHeap

Project Size: Approximately 800,000 lines of source code for game, editor, and associated tools; 60,000 lines of scripts; 21 million total lines of .GAS configuration files; 8,500 textures, 2,000 animations, 2,600 object and actor meshes, 3,700 terrain meshes

Return to the full version of this article