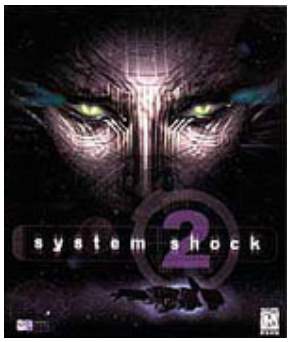
GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Irrational Games' *System Shock 2*

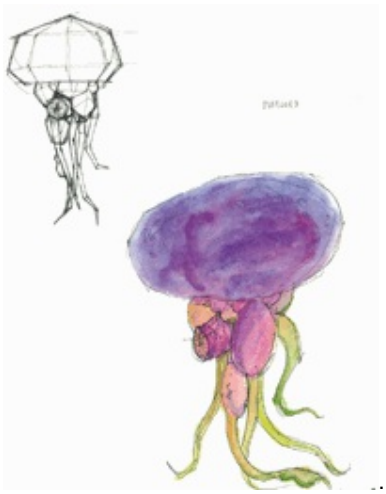
By Jonathan Chey

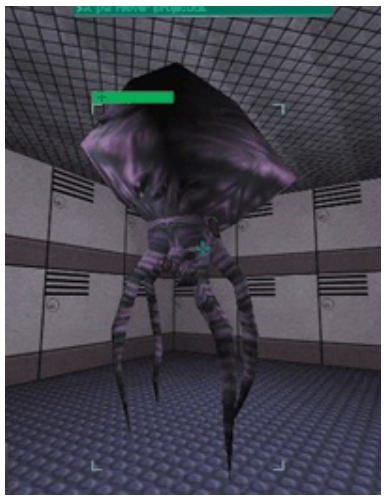


This is the story of a young and inexperienced company that was given the chance to develop the sequel to one of the top ten games of all time. The sequel was allotted roughly one year of development with its full team. To make up for the short development cycle and correspondingly small budget, the project was supposed to reuse technology. Not technology in the sense of a stand-alone engine from another game, but individual components that were spun off from yet another game, *Thief: The Dark Project*. The *Thief* technology was still under development and months away from completion when our team started working with it. To cap everything off, the project was a collaborative effort between two companies based on a contract that only loosely defined the responsibilities of each organization.

Add to these gloomy initial conditions the fact that the game from which our shared technology was derived slipped more than six months from the initially estimated date, that several developers quit during the project, that we didn't bring the full team up to strength until six months from the final ship date, and that we struggled with financial and business problems during the entire project. Having learned this, you might anticipate the worst. Strangely, *System Shock 2* shipped within two months of its targeted date and will, I hope, be recognized as a sequel worthy of its esteemed ancestor.

Let's step back and trace the origins of the companies and the project. Looking Glass Studios is familiar to many as the creator of a series of highly innovative titles including the original *System Shock*, the *Ultima Underworld* series, the *Flight Unlimited* line and *Terra Nova*, among others. Three years ago, Ken Levine, Rob Fermier and I were developers at Looking Glass, struggling with the aftermath of *Voyager*, an aborted *Star Trek: Voyager*-licensed project. At the time, Looking Glass was in financial and creative disarray after a series of titles that, though critically acclaimed, had failed to meet sales expectations, the latest being *Terra Nova* and *British Open Championship Golf*. Frustration with the 18 months wasted on *Voyager* and a certain amount of hubris prompted three of us to strike out on our own to test our game design and management ideas. We wanted to nail down a rigorous and technologically feasible design, focus on game play, and force ourselves to make decisions rather than allow ourselves to stagnate in indecision. We wanted to run a project.





Concept sketch of the Psi-Reaver (left) and the final product(right)

So we formed Irrational Games. After some misadventures with other development contracts, we unexpectedly found ourselves back at work with Looking Glass as a company rather than as employees. Initially, our brief was to prepare a prototype based on the still-in-development Thief technology recast as a science-fiction game. The scope of the project was very wide, but we quickly decided to follow in the footsteps of the original *System Shock*. Our initial design problem was how to construct such a game without the luxury of the actual *System Shock* license, since no publisher had yet been signed. Our initial prototype was developed by the three of us working with a series of contract artists. Our focus was on the core game-play elements: an object-rich world containing lots of interactive items, a story conveyed through recorded logs (not interaction with living NPCs), and game play realized through simple, reusable elements. This focus enticed Electronic Arts into signing on as our publisher early in 1998 - a fantastic break for us. It meant we could now utilize the real *System Shock* name and characters.

Immediately, we went back to our original design, threw away some of the crazier ideas that had been percolating and began integrating more of the rich *System Shock* universe into the title. That was the point at which the real development began.

It's the Engine, Stupid

Nothing impacted the development of *System Shock 2* as much as the existing technology we got from Looking Glass. This fact cannot be classified monolithically under the heading of "what went wrong" or "what went right," however, because it went both wrong and right. The technology we used was the so-called "Dark Engine," which was essentially technology developed as a result of Looking Glass's *Thief: The Dark Project* (for more about its development, see "Looking Glass's*Thief: The Dark Project*," Postmortem, July 1999).

The *Thief* technology was developed with an eye toward reuse, and I will refer to it in this article as an "engine." However, it is not an engine in the same sense as *Quake's*, *Unreal's*, and *LithTech*. The Dark Engine was never delivered to the*System Shock* team as a finished piece of code, nor were we ever presented with a final set of APIs that the engine was to implement. Instead, we worked with the same code base as the Thief team for most of the project (excluding a brief window of time when we made a copy of the source code while the *Thief* team prepared to ship the game). Remarkably, it is still possible to compile a hybrid executable out of this tree that can play both *Thief* and *System Shock 2* based on a variable in a configuration file.

This intimate sharing of code both helped and hurt us. We had direct access to the ongoing bug-fixes and engine enhancements flowing from the *Thief* team. It exposed us to bugs that the Thief team introduced, but it also gave us the ability to fix bugs and add new features to the engine. Because we had this power, we were sometimes expected to fix engine problems ourselves rather than turning them over to Looking Glass programmers, which wasn't always to our benefit. At times we longed for a finished and frozen engine with an unalterable API that was rigidly defined and implemented - the perfect black box. But being able to tamper with the engine allowed us to change it to support *System Shock*-specific features in ways that a general engine never could.

What Went Right

1. The Irrational Development Model.

In our hubris after leaving Looking Glass, we formulated several informal approaches to development that we intended to test out on our projects. Most of these approaches proved to be successful and, I think, formed the basis of our ability to complete the project to our satisfaction.

First, we designed to our technology rather than building technology to fit our design. Under this model, we first analyzed our technological capabilities and then decided on a design that would work with it. This process is almost mandatory when reusing an engine. Sometimes it can be difficult to stick with this when a great design idea doesn't fit the technology, but we applied the principal pretty ruthlessly. And many of the times we did deviate we had problems.

Another feature of our development philosophy is that everyone participates in game design. Why? Because all three of the Irrational founders wanted to set the design direction of our products, programmers were able to resolve design issues without having to stick to a design spec, and we strongly emphasized game design skills when hiring all of our employees and contractors. In all our interviews, one of our most pressing questions to ourselves was "Does this person get games?" Failure to "get" them was a definite strike against any prospective employee. Ultimately, the team's passion for and understanding of games was a major contributor to the design of the final product.

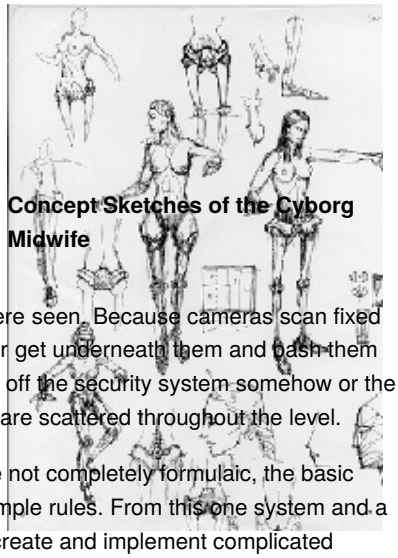
The final goal of our development process was to make decisions and hit deadlines. We focused on moving forward, and we didn't allow ourselves to be bogged down. We desperately wanted to ship a game and believed that the discipline imposed by the rule of forward motion would ultimately pay off in terms of the final product quality as well as delivery date. While there are features in *System Shock 2* that could have been better if we had not rushed them (the character portraits for example), we still firmly believe that the game as a whole was made better by our resolve to finish it on time.

2. Use of simple, reusable game-play elements.

The field of companies developing first-person shooters like id and Valve, among others, is impressive. From the outset we realized that we would have to work smarter, not harder, to make a game that could stand up in this market. It would be a futile attempt to create scarier monsters, bigger guns, or higher-polygon environments. Additionally, we realized that our design time and budget were very tight and that we would not have time to carefully hand-script complicated game-play sequences in the engine. Instead, in an attempt to shift the battlefield, we chose to focus on simple, reusable game-play elements. The success of *Half-Life*, which launched while we were in the middle of *System Shock 2* confirmed our intuitions in this respect. We simply didn't have the time, resources or technology to develop the scripted cinematic sequences used by *Half-Life*. We consoled ourselves with the knowledge that we were not even trying to do so.

This strategy melded very well with our acquisition of the *System Shock* license, as the original *System Shock* had already been down this road. We decided to expand on elements that we liked in

System Shock and then add similar new systems. Each such new system was evaluated rigorously in terms of game-play benefits, underlying technology, and design-time requirements.



Concept Sketches of the Cyborg Midwife

For example, take the ship's security system. Early on we decided that we wanted to continue the surveillance theme from *System Shock*, which we could leverage throughout the game to provide lots of game play for very little implementation cost. We realized that security cameras would be trivial to implement using existing AI systems (they are just AIs pruned of many of their normal abilities) and that once we had cameras that could spot and track the player, we would be able to build several game-play elements out of them. Cameras could summon monsters to the player, so much of the game play consisted of avoiding detection by security cameras and destroying cameras before you were seen. Because cameras scan fixed arcs, the player can utilize timing to sneak by cameras, pop out and shoot them at the right moment, or get underneath them and bash them with a melee weapon. Once a player is spotted, monsters flood the area until the player is able to shut off the security system somehow or the system times out. This introduces the need to deactivate security systems via security computers that are scattered throughout the level.

This type of system was technologically simple to implement and required minimal design effort. While not completely formulaic, the basic procedure to set up a camera and security system could be shown to designers quickly using a few simple rules. From this one system and a couple of associated subsystems, we derived a large amount of game play without having designers create and implement complicated scripted sequences and story elements. When you throw together many such systems (as we did), you end up with a lot of game play.

3. Cooperative development.

System Shock 2 was truly a cooperative development between Irrational and Looking Glass. Looking Glass provided the engine and a lot of infrastructure support (such as quality assurance), while Irrational handled the design, project leadership, and the responsibility for marshaling resources into the final product. Both entities contributed personnel to the development team. Inevitably, some friction arose from this process while we sorted out who was responsible for what. However, this cooperation was ultimately successful because both sides were interested in developing a great product, and we were able to compromise on most issues. (On the most mundane level, Irrational ended up providing late-night, weekend meals for its development team and for Looking Glass on some days during the week.)

Our cooperative arrangement was founded on a contractual agreement, but we avoided falling back on this contract in most cases. We preferred to resolve issues through informal discussions. Conceptually, Irrational was to be responsible for the development of the product and Looking Glass was to provide A/V content and quality assurance services.

During the early stages of the project, a deal was worked out whereby a small number of Looking Glass personnel were subcontracted to Irrational when it was determined that Irrational's development budget could not cover all the *System Shock 2* development costs, and as compensation for the late delivery of the *Thief* technology. Unfortunately, these personnel were not always available on time - a situation which caused us much concern. We knew that this "resource debt" could never really be paid off until *Thief* shipped - nothing is so difficult as prying resources away from a team that is trying to ship a product before Christmas. It wasn't until December 1998 that we first began to see some of these promised resources. However, these "resources" - real people - had just finished up *Thief* and were totally fried following the grueling crunch to ship *Thief*. The saving grace and reason that this arrangement was ultimately successful was that these developers were all talented and experienced and already knew the technology. Their addition to the team gave us a solid boost during the final months in our ship cycle.

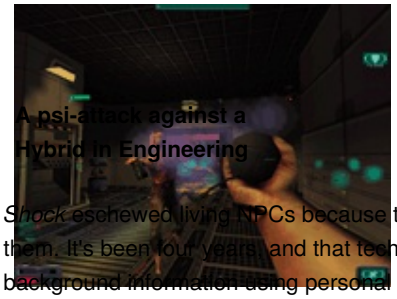
The other benefit of the cooperative development agreement between Irrational and Looking Glass was that our respective engine programmers could share knowledge. The ability to walk over and quiz engine programmers about systems proved to be an invaluable benefit that more than compensated for the lack of a rigorously specified and documented engine. Without a formal understanding of the engine, we had to resolve engine issues in a personal and informal manner. This process relied on the personalities of the responsible individuals on the engine team. Thus, the Irrational programmers balanced their time not only according to the complexity of their tasks but also according to how much support was available from the engine side.

Overall, Irrational's relationship with Looking Glass was an unusually close one and ultimately successful as a result of our mutual respect and willingness to work with each other. Despite our partnership being based on a formal contractual arrangement, it was our ability to work flexibly above this legal level that enabled the development to proceed smoothly.

4. Design lessons from *System Shock*.

Though the *System Shock* license was wonderful, there were some problems. The biggest was simply the challenge of living up to the original. Fortunately, we had the freedom to pay homage to *System Shock* legitimately by reusing elements from it. Additionally, we had access to some of the original developers, including our own lead programmer Rob Fermier.

As with most sequels, we faced the challenge of keeping the good elements of the original game while not blindly copying them. We knew that most players would want a new story set in the same world, with the same basic flavor as the original game, yet we also wanted to reach out to a broader audience. We resolved these issues by identifying the key elements that made *System Shock* so good and reinterpreting those elements using current technology. Some elements made it through largely unchanged (for example, the storytelling logs and e-mails, the über villain Shodan and her



A psi-attack against a Hybrid in Engineering

close involvement with the player throughout the game, and the complexity of the world). Other elements were reinterpreted (such as the look of the environment, player interface, and techniques for interacting with the world). A small number of items were simply cut, most notably the cyberspace sequences - we were fairly united in our opinion that these just didn't work well in the original.

Notably, as with the original *System Shock*, we opted to omit interactive NPCs in the game. *System Shock* eschewed living NPCs because the technology of the day was simply inadequate to support believable and enjoyable interactions with them. It's been four years, and that technology is still not available. So we continued the tradition of *System Shock* and provided players with background information using personal logs and e-mails gleaned from the bodies of dead NPCs.

Perhaps our biggest deviation from the original revolved around the player interface. It's commonly accepted that *System Shock's* interface, while elegant and powerful once understood, presented a significant barrier to entry. Our primary goal was to simplify this interface without dumbing it down. We devoted more design effort to this task than to any other system in the game, and it required many iterations before we were happy with it. We adopted a bi-modal interface in which there are two distinct modes (inventory management and combat/exploration) between which the player can toggle. This was a risky decision. This bi-modal model was mandated by our desire to keep the familiar and powerful mouse-look metaphor common to first-person shooters while retaining cursor-based inventory management. How we switched between modes became our biggest design challenge. Sometimes these mode changes are explicitly requested through a mode change key, and sometimes they are invoked automatically by attempting to pick up an object in the world. So far this system seems to be working well, though only time and user feedback will tell whether we really got it right.

5. Working with a young team.

The *System Shock* team was frighteningly young and inexperienced, especially for such a high-profile title. Many of our team members were new to the industry or had only a few months' experience, including the majority of artists and all the level builders. Of the three principals, only Rob had previous experience in his role as lead programmer. Neither Ken, the lead designer, nor I, the project manager, had previously worked in these roles.

It's not totally clear how we pulled off our project with our limited experience. Partially, it must have been due to our ability to bond as a team and share knowledge in our communal work environment ("the pit"). To a certain extent, inexperience also bred enthusiasm and commitment that might not have been present with a more jaded set of developers. We also worked hard to transfer knowledge from the more experienced developers to the less seasoned individuals. Rob worked on an extremely comprehensive set of documentation for the functional object tools, as well as a set of exercises ("object school") to be worked on each week. These kinds of efforts paid back their investment many times over.

This is not to say that our progress was all sweetness and light. The art team, for example, floundered for a long time as we tried to integrate the junior artists and imbue a common art look in the team's psyche. We had a lot of very mediocre art midway through our project and the art team was stagnating. Ultimately, management had little to do with the art team's success - they were largely able to organize themselves and create a solid, original look.

On the management front, our inexperience was apparent. We blundered through the early stages of development with scheduling and management issues. A large problem was our failure to assign specific areas of responsibility and authority early on. Bad feelings arose as a result, which could have been avoided if we had clearly delineated areas of responsibility from the start.

What Went Wrong

1. Poor level design process

Level design is a clearly defined professional activity in the game industry. It's a profession that mixes artistic and technical skills in equal measure, and the bar is raised on both fronts every year. Despite our understanding from the very beginning that the level building would be a problematic part of the *System Shock* development process, we didn't quite grasp how difficult and time consuming it would be, nor did we expect that it would eventually block the shipment of the game.

In hindsight, our failure to understand the amount of work needed to design levels is reprehensible given that we had seen the same problems emerge on *Thief*, and that *System Shock 2* levels involved substantially more complex object placement than *Thief*. I attribute this error mostly to our denial of the problem - we had a limited budget for level designers and there is a long training time required to get designers familiar with the complex Dark Editor. So we locked ourselves into working with the resources we had. Since each individual task required from the designer (apart from initial architectural work) was relatively simple, it was easy to believe that the sum total of work was also relatively small. What we overlooked was the fact that *System Shock 2* involves so many objects, scripts and parameters. As such, the work load on level designers was excessively large. In addition, we made a classic beginner's mistake and failed to provide adequate time for tuning in response to play-testing feedback. In *System Shock 2* this was particularly important because the ability of the player to re-enter levels means that the difficulty of a level cannot be adjusted in isolation from the rest of the game. Often we had to impose global changes across all levels, which could be very expensive even when the change was relatively minor.

We took a novel approach to the level building process by attempting to apply design levels using a production-line method. Using this metaphor, we attempted to divorce the different stages of work on

Cold comfort in Hydroponics.

the level: rough architecture, decorative and functional objects, architectural polish, and lighting. It was not considered necessary for the same individual to be involved in all stages of this production process. This approach had positive and negative consequences. The advantages were that we could track progress on levels, we could "bootstrap" levels fairly quickly, and we could (in theory) swap individuals in and out of different tasks. The disadvantages are fairly obvious, and most stem from the fact that the various stages of level design are clearly not independent (for example, architecture is ideally built with an understanding of the functional objects that are to be used in the level). Although I think our process was necessary in order to get the game out on time, it probably detracted from the quality of some of the levels. In addition, psychological factors, such as lack of ownership and training issues (stemming from unfamiliarity with levels) speak very strongly against transferring people from one task or level to another. Nevertheless, there were several benefits of our procedure - mostly the ability to employ particularly talented individuals to pinch hit on particular levels, and the psychological benefits of completing architectural work early in the schedule.

Perhaps the rudest shock in our level building process came from our misunderstanding of what part of the process would prove to be most difficult. Architectural work was actually fairly simple, because we intentionally kept our spaces fairly clean and did not attempt anything too unusual. However, placing and implementing our objects was far more complex and involved than we expected. One difficulty that we encountered was educating our designers in what was expected from them in terms of game-play implementation. Most of our level builders had previously built *Quake* or *Unreal* levels and were not familiar with the style of game play that we were trying to build in *System Shock 2*. Partially this was because we were simply exploring a style of game play that we did not entirely understand ourselves. But it reflected a failure on our part to properly educate the designers. Building prototypical spaces, looking at past games and conducting more intensive discussions about game play will all be part of our future projects.

Our other major design hurdle was the instability and inscrutability of Dromed, the Dark Engine editor. Dromed is a cantankerous beast and many man-months were spent struggling with its idiosyncrasies. Perhaps our biggest problem stemmed from the lack of support in one crucial area - the part of the engine concerned with translating the designer-placed brushes into the basic world representation. Like many complex 3D engines, the Dark Engine suffers from troubling epsilon issues (data errors caused by rounding inaccuracies) and other glitches that crop up during level compilation. Because the programmer who implemented the basic renderer and world representation was not available during the majority of the *System Shock 2* development cycle, we had to work around these problems. It was often a frustrating process when the fundamental cause of the problems was not even known. Over time we developed a set of heuristics to avoid the majority of the glitches, but we were forced to lock down much of the level architecture before we wanted to in order to ensure stability.

2. Motion capture difficulties

The Dark Engine has a complex creature animation playback system and deformable mesh renderer. We encountered many problems with this piece of technology along our data integration path, and found quirks in the playback systems as well. Primarily, the system was hampered by the fact that data frequently had to be modified by hand, that mysterious bugs would appear in motions during playback which had not been present in the source data, and that few tools were available for debugging and analysis. We were ill-equipped to deal with these kinds of problems, having devoted few resources to dealing with the technology problems.

Our primary animation source was motion capture data. We were nervous about the technology from the start and attempted to minimize our risk by concentrating primarily on humanoid creatures with a small number of interesting variants such as spiders and floating boss monsters. In retrospect, this was a very wise decision, as we had a lot of trouble even with this simple set of creatures. Motion capture technology and capture services were contracted from a local company, but unfortunately this company viewed its motion capture work primarily as a side business and did not display much interest in it. In fact, they cancelled this sector of their business during our project, and we had to fight hard to complete the sessions that we had already scheduled with them.

Our capture sessions were hampered by our inexperience with the technology and by the fact that we did not plan properly for the sessions. We hadn't defined key poses, rehearsed the motions, or ensured that our motions were compatible with the technology. Optical capture technology, the technology that we used, can be glitchy and has difficulty with motions that have obscured markers, as in the death motions that were necessary for *System Shock 2*. Over the course of three sessions, we gradually refined our motions, but we spent a lot of time reshooting failed captures from earlier sessions.

Even in the best cases, most of our captures exhibit strange artifacts (feet pointing down through the ground, hands improperly aligned, and so on), whose causes are still unknown to us. In future projects we will hand-animate almost all of the data, and we will need to understand better what aspect of the conversion process introduced these artifacts into our final game animations, although the irregularities never appeared in our raw data. Motion capture technology, while highly efficient compared to hand animation, must be approached carefully to obtain good results.

3. Implementing scripted sequences

Motivated by the dramatic scripted sequences in *Half-Life*, we attempted to introduce similar elements into *System Shock 2*. In doing so, we broke one of our rules: we tried to step outside the bounds of our technology. Although we attempted relatively simple sequences and ultimately got them working, they were time sinks, and the payback was relatively slight. For example, we scripted a hallucinatory

sequence in which the player character rides through the interior of the alien boss-monster, known as the Many. This so-called "Many ride" was the source of innumerable bugs -the player would be thrown off the moving platform, manage to kill his projected self, bump into walls, and so on. We confirmed our intuition that the Dark Engine does not support complex scripted sequences well because the toolset (AI, moving terrain, and animation) is not optimized for this sort of behavior. The moral is, once again, to work with your technology, not against it.


Xerxes, central computer of the Von Braun

4. Inexperience with multiplayer game development

Early in the project we were asked to identify the major risks associated with the project. Our number one candidate by far was the multiplayer component. This was the only new substantial engine feature that was to be added and it was a complicated piece of work. We were particularly nervous about this technology for a couple of reasons. First, it is usually much harder to make this kind of pervasive change to an existing piece of software than it is to build it in from scratch. Second, Looking Glass had no track record in shipping multiplayer technology and we were not confident that the development was fully understood.

Irrational did not want to introduce multiplayer support into *System Shock 2* because we considered it a tangential feature that did not contribute to our core strengths. However, marketing concerns dictated it, so ultimately we acquiesced. Our lack of enthusiasm for this feature contributed to its developmental problems because we failed to monitor its progress adequately or raise concerns when that progress fell behind schedule.

Because this was the first multiplayer product developed by Irrational or Looking Glass, we did not properly estimate the time required for the multiplayer testing. We did not devote adequate quality assurance resources to this feature. Too much time was spent testing the multiplayer features over the LAN and not enough over the more demanding modem connections.

Given the difficulties posed by the multiplayer technologies, the engine developers working on the task made great efforts, and their early results were promising. However, the early departure of one of the programmers, and the fact that he was not replaced, ultimately doomed any possibility of shipping the multiplayer technology with the initial SKU. Reluctantly, we opted for a patch that would be available at the same time as the single-player box reached shelves. Our cooperative multiplayer game will undoubtedly be fun and will probably be enjoyed immensely by a relatively small number of our customers. However, we wonder whether our failure to deliver a promised feature in the box will ultimately hurt us more than the absence of that feature from the start would have.

5. Running a company while building a game

As the principals of the company, Ken, Rob and I didn't really understand what it took to run a business and simultaneously work in that business. None of the Irrational founders started the company to be businessmen, and we have always believed that the ultimate health of the company depended on us all staying involved in the development process, which is, after all, what each of us enjoys and wants to do. Unfortunately, as anyone who has run a business knows, there is a lot more to starting and maintaining a company than sitting around at board meetings smoking cigars. From the mundane matters of making payroll, organizing taxes and expense reports to business negotiations and contract disputes, there is substantial overhead involved in running even a small company such as Irrational. In our naïveté, we did not factor these tasks into our schedules and the result was that they mostly became extra tasks that kept us in the office late at night and on weekends.

As a result of our misjudgment, we just had to work harder. Rather than enduring a crunch period of a few months, the entire last year of the project was our crunch time, as we struggled desperately to fulfill our jobs as programmers, designers, and managers as well as keep the money flowing in (and out) of the company. Our tasks were complicated further by the need to reincorporate the company from an S-corporation to an LLC during the final two months of the project (a legal maneuver designed to allow me, an Australian national, to be allocated company stock).

As well as destroying our personal lives, our failure to judge the magnitude of our task meant that we had to devote less time than we desired to every aspect of our work. My programming time was severely curtailed and I was able to spend far less time on *System Shock's* AI than I wished. Simultaneously, I was unable to provide the level of direct management that I wanted, and I was forced to postpone company financial work until the end of the project or hurry it through. The results were less than optimal all around.

Ultimately *System Shock 2* turned out better than I ever hoped it would. the final vindication for me was sitting in my office and playing the game in the final couple of weeks of the project, while waiting for EA to approve our final build. Despite the lack of sleep, the near-complete breakdown of my nervous system and the 18 months of time I spent working on the project, it was still fun to play. I like to think that we have managed to capture the feel of the original game by putting more game play into what initially looks like a fairly straightforward first-person shooter. It's been a great first project for Irrational Games and we look forward to doing even better the next time around.

Jonathan Chey was the project manager and a programmer on *System Shock 2*. He is also

System Shock 2

Irrational Games LLC
Cambridge, Mass.
(617) 441-6333
http://www.irrational-games.com

Looking Glass Studios Inc.
Cambridge, Mass.
(617) 441-6333
http://www.lglass.com

Release date: August 1999

Intended platform: Windows 95/98

Project budget: $1.7 million

Project length: 18 months

Team size: 15 full-time developers, 10-15 part-time developers

Critical development hardware:

one of the co-founders of Irrational Games. Prior to founding Irrational Games, he worked at Looking Glass Studios and prior to that he received his Ph.D. in Cognitive Science from Boston University. He is currently working on his tan back in his hometown of Sydney, Australia. He can be reached at jon@irrational-games.com.

Pentium II machines, 200MHz to 450MHz with 64MB to 128MB RAM, Nvidia Riva 128, Voodoo, Voodoo 2, TNT cards, Creative Labs' sound cards, Wacom tablets, Windows 95/98. Also used SGI Indigo workstations.

Critical development software: Microsoft Visual C++ 5.0, Opus Make, 3D Studio Max, Adobe Photoshop, Alias|Wavefront Power Animator, DeBabelizer Pro, RCS, Filemaker Pro, and Adaptive Optics motion capture software.

The Team at Irrational Games (left to right):

First row: Steve Kimura/Artist, Jonathan Chey/Project Director, Justin Waks/Multiplayer Programmer, Mauricio Tejerina/Artist, Rob "Xemu" Fermier/Lead Programmer, Dorian Hart/Designer, Lulu Lamer/QA Lead.

Second row: Ian Vogel/Level Designer, Scott Blinn/Level Designer, Michael Swiderek/Artist, Rob Caminos/Motion Editor, Nate Wells/Artist.

Third row: Mike Ryan/Level Designer, Ken Levine/Lead Designer, Mathias Boynton/Level Designer.

Not shown: Gareth Hinds/Lead Artist.

Return to the full version of this article