GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Raven Software's *Soldier of Fortune*

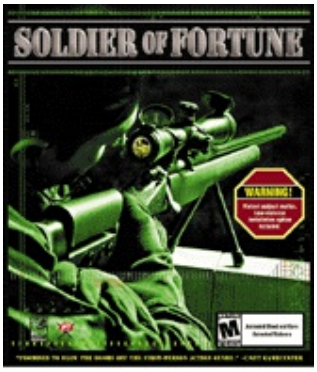
By Rick Johnson,Eric Biessman

The development of *Soldier of Fortune* was rife with questions and uncertainties right from the very beginning. Fresh from finishing up *Portal of Praevus*, the *Hexen 2* mission pack, Raven was ready to dig in to a full-fledged stand-alone product. Unfortunately, no one at Raven had a solid idea for our next project and we found ourselves floating in a sea of ideas without a solid direction. With a full team ready and willing to go, we needed a project and we needed one fast. It was then that Activision handed us the *Soldier of Fortune* license.

In the beginning, what was to become the SoF team was focusing on several different story lines and game ideas. One of these was a somewhat real-world, military-style shooter based in a World War II setting. When we decided not to pursue that game, we began looking for new game ideas. We knew that we still wanted to do a real-world military game, but beyond that we didn't have much of an idea. As soon as we got the *Soldier of Fortune* license, though, the groundwork for the game immediately began to fall into place.

While the license name itself was met with mixed reactions from the SoF team, at its core was everything that we wanted from the game. Action, intrigue, political turmoil, and firepower were key elements of the design from the very beginning. Now we needed to find a story that would complement the license and turn it into a great game.

The name Soldier of Fortune evokes different images for different people. One thing that we could all agree on was that the title reflected the mercenary life; making money at the risk of death. This was something that we wanted to highlight and focus on dramatically throughout the game. However, focusing on this one aspect tended to blind us to the bigger picture of what we were trying to accomplish, and our first few story attempts failed miserably. We focused too much of the gameplay on making money and not enough on finding something that would truly compel the player throughout the game. Nevertheless, even without a story set in stone we began the production of the game. This was a decision that we would come to regret many times throughout the rest of the development cycle.



The bright side to spending a large portion of development time working on a game without a solid story was that most of it was spent on technology creation. The bad part was that many of the levels that were originally planned and created had to be reworked or removed from the game entirely. On top of that, Activision was getting a little nervous that they had not seen any solid gameplay from us yet after almost a year of development. This uneasiness itself caused major turmoil in the development and it took a while for us to settle into the game that we would eventually create.

Luckily, during this time, all of the core technology was implemented and functioning smoothly. Because of this, once we nailed the story down, we were able to jump head-first into the production and quickly create a solid product. In order to achieve a strong sense of realism, we decided to talk to a published author about the script and also to a real-life "military consultant" about how a soldier of fortune truly lives his life. This was one of the major turning points in the development and we were finally able to focus the game into its final product.

As we settled on an action-movie feel, *SoF* finally began to take form. We were able to tie together an appealing story line quickly with several twists to keep the player enthralled. Combining this with the extensive amount of information that our military consultant provided us, everyone on the team was excited about the project again and the true development of the game got underway. In less than ten months, the core of *SoF* was assembled into a fun, viable product. After the game was released this past March, the rest, as they say, is history.

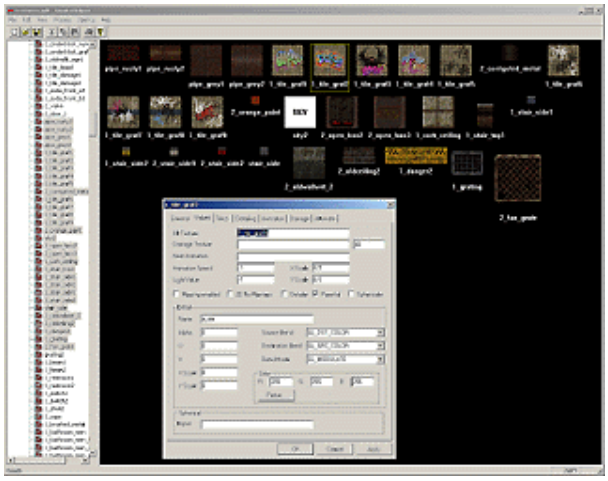
---

1. Familiarity with technology plus powerful tools and enhancements. One of the most important pluses for *SoF* was the team's experience and familiarity with the Quake technology. Raven has been using id Software's technology since its early days of *Heretic* and *Hexen*. This familiarity allowed us to experiment, create, and use tools that vastly sped up the game's development.

One of the first tools that we developed was QuakeHelper. As *SoF*'s development progressed, we realized that all of the options associated with the individual textures for the world were becoming too complex to encode into a parsing file. QuakeHelper was created to allow a visual way to assign all of these properties. This included texture scaling, detail texturing, damage texture (next texture to be shown, and the amount of damage it should take), material properties (sound and visual effects for user interactions), and alternate textures (more detailed and unique textures would be replaced by common textures on video cards with lower texture memory). In the end, *SoF* had more than 5,000 unique world textures. QuakeHelper saved the artists a tremendous amount of time in preparing the textures for the game and in adjusting and tweaking their properties.

One of the benefits of working in the Quake community is that the public has access to most of the source code to the tools. In the beginning

of the project, we used QRAD, which was the original tool id developed to calculate the lighting information on the world. Our designers learned of an enhanced version of QRAD that had been developed by Tim Wright. He called the new modified version ArghRad!, which added a Phong-type shading model to the light map calculation, a global sunlight casting point, and several bug fixes. Raven contacted him to arrange to get the source code. In the end, this helped us create better-looking levels by utilizing the wonderful Quake community.

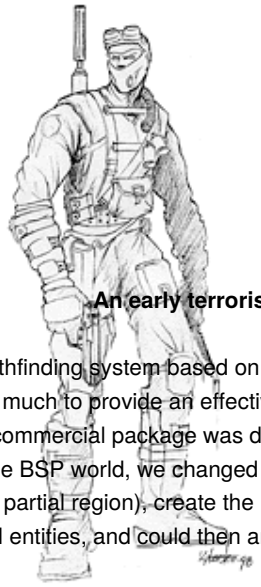


Raven developed QuakeHelper to manage more than 5,000 unique world textures with a visual means to assign properties to them.

DS, or Designer Script, was developed jointly for both *Heretic 2* and *SoF*. The goal was to provide a simple language for designers to help create more complex scenes and puzzles in the game. Those who designed this language rightfully kept in mind whom the language was for. In other words, it was a language created by programmers for designers. While this may seem like a straightforward concept, often this idea gets lost during the development phase of tools or other items that are supposed to assist the desired recipient. Even though this language did have certain limitations (described under "What Went Wrong"), it did help meet our goals for both projects. The following two tools helped extend the scripting language in simple yet powerful ways.

While one of SoF's designers was playing around with Lightwave to create a complex motion path for an entity, he ended up writing an exporter that created a DS script. The script consisted of a series of move and rotate commands to simulate the complex movement animated in Lightwave. While this accomplished the ultimate goal of importing the entity's animation into the game, it was not very efficient. Exporting the movement into a file and adding a command to the scripting language to play that movement file corrected this. This format was known as ROFF (Rotation Object File Format). *SoF* used about 500 of these movement files, from the simulation of helicopter movement and exploding crates, to even creating a flying bird (although you'll have to look really hard to see that one).

Because of the large amount of animation needed for *SoF* and the fact that we were going to be using a mix of traditional hand-animated sequences and motion capture sequences, we needed something that would work well with both. All of our motion capture data was taken by House of Moves, a wonderful motion capture house, and sent to our animators. From there, we used Chimaera, a control rig within Softimage that allowed us to tweak both types of animation easily. It also allowed the animators to utilize both inverse and forward kinematics simultaneously, accomplishing this ordinarily complex task with relative ease. One of Chimaera's most important features was that it allowed the animators to apply every animation to any humanoid model, including models not local to *SoF*. This tool has also been put to good use on our next release, Star Trek - Voyager: Elite Force.



An early terrorist sketch.

We originally developed SoFPath to create a pathfinding system based on the BSP of a map. During the development of this tool, however, we discovered that the world was broken up too much to provide an effective means of pathfinding. Our early use of .ROFF files also showed that animating entity movement or rotation in a commercial package was difficult without a good representation of the world. Since the SoFPath utility had a good "understanding" of the BSP world, we changed it to export .IFF Lightwave object files. The designers would basically BSP their map (either the full map or a partial region), create the Lightwave file, and import it into Lightwave. They then had a representation of the world, a rough outline of all entities, and could then animate things accurately. Later in the project, we also added the ability to edit these files in 3D Studio Max.

Both dynamic music and ambient sound systems were designed internally to create immersive environments in *SoF*, but they also allowed the sound designer to add sound assets into the game more easily. Instead of hard-coding the names of the sound files, the tools provided a quick and flexible method of tweaking sonic properties in levels. This process not only took the weight of sound placements off the programmers' shoulders, but also empowered the sound designer with a powerful and creative tool to create unique soundscapes.

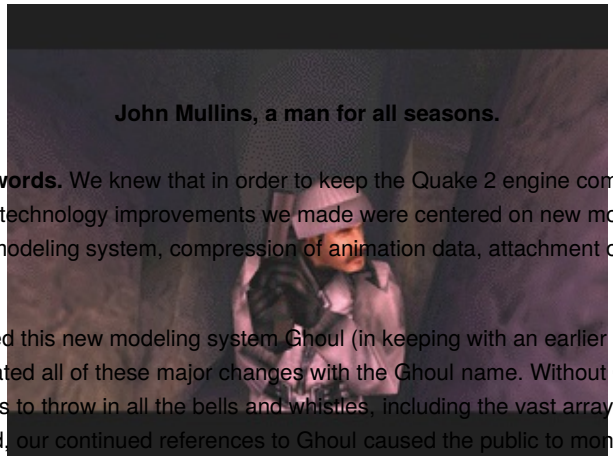
2. Taking time to address violence concerns. From its inception, we knew that *SoF* was going to be a game for adults. Due to its large amount of simulated violence, we wanted to make sure that adults had every opportunity to keep *SoF* out of the hands of minors while still being able to play the game on their home computers. In order to do this, we implemented several different protective measures for consumers.

First and foremost was creating the *Soldier of Fortune: Tactical Non-Violent Version*. A totally separate SKU from the regular version of the game, the low-violence option removed all of the gore, limited the number of death animations, and seriously toned down the game in general. This version used the same box as the regular version, but colored red instead of green and stamped with a large advisory that stated that it was different from the regular version.

For the regular version, we added a violence-lock feature to allow users to password-protect the game and change various options to their liking. The consumer could lock out dismemberments, blood, death animations, adult textures, and other adult content, essentially turning the regular version into the low-violence version. To further inform consumers of the violent subject matter, a large warning was placed on the front of the box and the ESRB rating was enlarged for greater visibility. A "mature audiences" warning was also added to the game's bumper and implemented into the menu system so that no one would be surprised by the game's content.

All of these features and functions helped extensively in the end. We widened our sales platform as stores realized they could order the tactical version if they wanted to, and we showed consumers that we listened to their needs and concerns, giving them a broader choice in their purchase.

3. Outside help. Although your team will most likely not be using real-life mercenary John Mullins to help design your game, outside individuals can be an incredible help in product development. Talking and working with a person who has an exhaustive knowledge of your game's subject matter will help refine your project and add a truly cohesive feel to the final product. As a consultant helping us with the military aspect of the game, Mullins gave instant feedback in areas where our knowledge was lacking and helped round out the areas that needed it. He described how trained soldiers would react to attacks. He discussed what sounds you would expect to hear in battle. He advised us on how the weapons in the game should "feel" to the player. In short, he helped us to create the correct atmosphere in which to immerse players. By drawing on the insights and knowledge of someone with first-hand experience of the action we were looking for, we were able to focus the design of the game.



John Mullins, a man for all seasons.

4."Commando" marketing and buzz words. We knew that in order to keep the Quake 2 engine competitive in the FPS realm, we had to add significant technology. Many of the technology improvements we made were centered on new modeling technology, which featured, among other things, a completely new modeling system, compression of animation data, attachment of models (bolt-ons), multiple skin pages per model, and advanced networking.

Our lead technology programmer dubbed this new modeling system Ghoul (in keeping with an earlier in-house technology proposal called Specter). In the public's eye, we associated all of these major changes with the Ghoul name. Without Ghoul, *SoF* would have been a mere shadow of the final product. It allowed us to throw in all the bells and whistles, including the vast array of enemies and the high degree of gore. As *SoF's* development progressed, our continued references to Ghoul caused the public to monitor the changes and build up their expectations. Ghoul became an important marketing word for *SoF*.

Two views of Sergei Dekker, the quintessential bad guy.

Besides normal marketing channels such as magazines and print ads, we decided to try our hand at "commando" marketing. By using our .plan files, giving web interviews, supporting the wonderful fan sites that were popping up, and making ourselves available through e-mail and online chats, we established a strong presence in the Internet community. This proved invaluable for consumer feedback. With the release of the demo and the early OEMs, players gave us instant feedback on what they liked and disliked and we were able to change the game accordingly. One example where this feedback came in handy was with limited saves. Originally, players were limited in the number of saves that they could make based on their present difficulty level. Many people who played the demo disliked this feature, so we added the ability to customize the number of saves that players could make, thus adapting the game directly to consumers' preferences.

5. Good planning and scheduling. One of *SoF*'s saving graces was that it was planned and scheduled well. The sheer volume of animation, art, programming, and levels forced us to update our schedules on a frequent basis. With a concrete animation naming system, an incredibly large and detailed database for animations, storyboards for every cinematic sequence, and a well-designed QA system, *SoF* did not suffer much inefficiency. The only area that endured some wasted time was the design due to the various story and game changes.

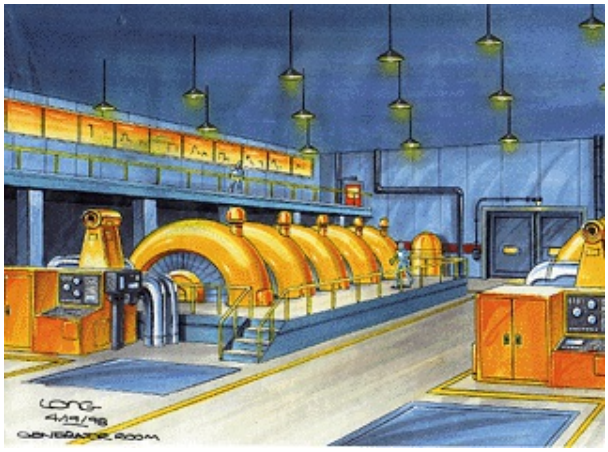
Once the story was finalized and had the green light, establishing and maintaining good planning and scheduling for the design process helped finish the game in a timely manner. We created total level walkthroughs, with each room and encounter written out. Flowcharts were used to draw the preliminary levels, and concept art was used for key location elements. Perhaps the most important lesson we learned from *SoF* was that preplanning is the most important aspect of game creation.

---

1. Unfocused design. The single most damaging problem during *SoF*'s early development was that the original game lacked a truly focused design. We knew what the fundamentals of the game would be, but we did not have the specifics that we needed to create a solid, cohesive product. The game's overall story changed five times before it was finalized - at one point we had even changed the basic game concept to a team-based tactical shooter, similar to *Rainbow Six*.

One reason for this indecisiveness was that, at the time, our original marketing team was wondering what the "hook" would be for the game. This was a major roadblock in creating the game because we knew that if marketing wasn't behind the idea, *SoF* wouldn't get the marketing money that it deserved. On top of that, without the backing of the marketing division, the senior management at Activision wouldn't get behind the title, either. We had to constantly sell and resell the idea that a high-octane, action-movie-like, real-world combat game would be enough of a hook. At times, it went so far that we were making design decisions not for the fun or betterment of the game, but to find the hook that we felt we were missing. The last straw came when we found ourselves working on a tactical team-based shooter, a complete 180-degree shift from our original design. We then decided to return to the game's roots and started banging out a new story.

Eventually, a new marketing team came on board that recognized exactly what we had been saying all along. *SoF* had more than enough to stand on its own, and they worked with us to find the right angle for marketing the product. This new team fit right in with the development team and things started to roll. On top of that, since we urgently needed to nail a story down in a short amount of time, they recommended that we meet with a hot-selling writer (Gonzalo Lira, author of the spy novel *Counterparts*) and John Mullins. Although the full story that Gonzalo Lira wrote for us was never used, some elements of it were, and the process made us realize exactly what we wanted from this game and how to get it. John Mullins contributed an element of realism to the game that we were missing at that time.

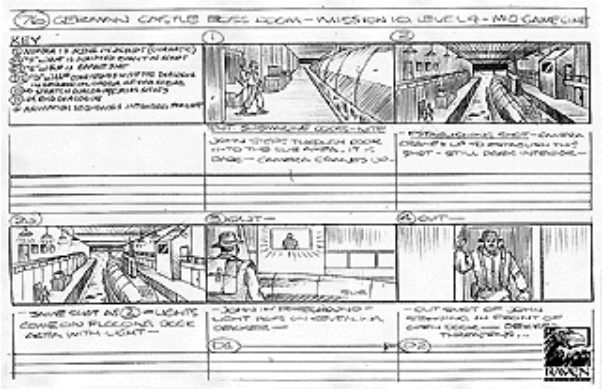


Off the coast of Russia lies a terrorist chemical weapons plant.

In short, working on everything at once was not the way to go. For the projects that we currently have lined up we are designing the entire game from start to finish before we begin physically developing it. The SoF team learned the hard way that a day of preplanning saves a week of rework. Also, getting a green light for everything before starting development saves having to back-pedal later on. Both of these lessons will be applied to our future projects.

2. Technology. creation took longer than expected to visualize gameplay. A sure way to sell your product is to have a working prototype at an early stage in its development. Since we had decided to give the Quake 2 engine an entire overhaul, we realized that we would really have to come together and work as a team to make sure things were completed on time.

One of the major enhancements for *SoF* was the Ghoul modeling system, which replaced the entire Quake 2 modeling system, and turned out to be quite the undertaking. Throughout the entire life of the project, tweaks and changes were made to Ghoul to make it more flexible and powerful. Unfortunately, this also meant that for a substantial part of the early development, we had no game to look at - only individual components. It's one thing to be able to look at a model in a model viewer, or at a level with nothing in it, but it's essential to be able to see the model in the world and interact with it.



Every cinematic sequence was conceptualized with storyboards first.

Another problem was the huge number of animations planned for *SoF*. Since we had so many animations (more than 600 sequences) we had to limit which animations would appear on a specific level due to memory constraints. Limiting the animations on a per-level basis was a nightmare in itself, not only for the animators but also for the AI programmer (who had to make the AI work within the animation constraints) and the designers, who had to create scripted and cinematic sequences using only the animations available for each level. As the game drew nearer and nearer to completion it became increasingly difficult to bring new animations into the game without ruining someone else's work by removing an animation that was already in use.

The final problematic technology was the AI. Developed throughout the entire course of the project, the AI went through many different incarnations. We decided early on that the pace of the game should be fast and furious with a large number of enemies attacking at once. Enemies were to be reactive, but not too intelligent. There were three major areas that caused AI problems: developing the models, developing the intelligence, and working with the scripting language.

The main enemy model for the game was very complex. Incorporating all of the animation sequences and consisting of nearly 4,000 polygons, the model contained every piece for various body builds, coats, and other items that differentiated the enemies. Because of this complexity, we were forced to preprocess enemy sets for each level. These individual enemy sets looked at what enemy pieces and animation frames were needed for the level because we did not have a skeletal system in place. This directly impacted the AI because not every move was now available on every level. In turn, the AI could only call animations that were generic across the levels.

The second area that caused AI problems was the addition of multiple skin pages, bolt-on accessories, gore, and death animations. Although not directly seen by most people as AI, all of these were important features for *SoF*. One of our main goals from the beginning of the project was to have lots of unique-looking enemies. This meant that our model was composed of many different skin pages into which we could swap different faces or outfits. We also implemented what we termed "bolt-ons": any item or feature that was not originally part of the model. These included Mohawks, canteens, briefcases, and side arms, which helped distinguish different characters.

Implementing the gore was also very time consuming. We implemented gore zones that required skin page overlays, bolt-on models of viscera, the ability to remove limbs, and all of the blood pools and spatters that litter the game.

Finally, implementing the various death sequences also hampered the AI. In addition to all of the gore that we created, we also had to play one of several animations when an enemy died. Animations had to be called based on certain circumstances, such as where on the body the enemy was shot and what he was shot with. On top of all of that, adding the violence-lock system that would allow players to lock out the game violence meant that all of the gore and animations had to be able to be shut off if the player wanted.

The third area that caused problems for the AI was its actual development. Along with the problems created by the per-level animation system, the AI also had to work with the game's scripting language. If the AI was tweaked in certain ways, it caused the scripting to break. Many times in the game, enemies had to be "frozen" in place while their script waited to be activated. If a player happened to see one of these suspended enemies before they were triggered, it obviously made the AI appear less intelligent. We had to come up with ways around these problems, and expended considerable time and energy to fix them. To make matters worse, the AI had a slight unpredictability built into it that caused scripted events to occur differently each time. Although unpredictability is good for gameplay, it had to be removed from the scripting element. Finally, a large amount of time was spent with the designers to build in hints for the areas (such as reactions of the enemies) and to specify which areas the enemies could traverse. At the beginning of the development we had "duck," "hide," "flee," and other commands that eventually were removed and taken over totally by the AI. The AI was in development until nearly the end of the project.

3. Too many OEMs and demos. Something that seemed like a great idea at the time but turned out to hurt us in the end was the decision to

make specific OEM releases before the game was truly finished. The main reason for this was that we looked at the revenue that would help the bottom line instead of considering how much it would set back the game.

Because there were both regular and low-violence versions of the game, we needed to make several different builds for the different violence levels and test each build accordingly. In the end, we had roughly 75 QA submissions. While each OEM and demo iteration helped bring more of the game together, it also diverted our attention from the final product. As we were tweaking and fixing the OEM versions, full production would come to a standstill as we focused on getting the smaller versions out the door.

4. No fixed deadlines. Originally, *SoF* was scheduled to ship in July 1999. Activision wanted to avoid releasing *SoF* in the "blast zone" of competing FPS titles that were shipping that year, so they extended the deadlines on the game. As our competitors' titles were pushed back, so was *SoF*. Although within these deadlines we had schedules set up and planned out, this caused a never-ending uncertainty of how much time we had left in the project and how much technology we could add or change within that time.

Cleaning up the New York subway system.

In March 1999, we realized that with our complex models and the amount of animation we wanted, we needed to address memory concerns. Because we thought that we only had three or four months left of core development at that stage, we concluded that switching to a skeletal system would be too risky for the project. Instead, we created a vertex compression system that mimicked the benefits of skeletal compression in a few ways. Unfortunately, this meant that we were not able to provide all of the animations at once, as we would still be over memory budgets. If we had known that our deadlines would be pushed back another six months, we would have added the skeletal system, saving everyone a large amount of headaches and work.

5. Miscommunication about some technologies. Confusion over project scheduling aside, additional technologies were developed during the course of the project that were never truly planned out appropriately, such as the terrain engine, the in-game effects editor, and the scripting system that we used. All of these technologies served to improve the game substantially, yet they could have worked better if they had been properly discussed between the team members.

The terrain engine, while flexible enough to do various types of visual effects, was never properly coded into the gaming logic. The basic premise of the terrain engine was that the designers would create architecture that represented the portions of the world that the player could interact with. For example, on the train level, the train was created by the designers. The terrain engine would then be responsible for the scrolling polygons, in this case the train tracks and surrounding landscape. When we put this level in, we soon realized that we needed a bunch of special code to handle the various effects, such as when a person falls off a train. We wanted to add more unique kinds of levels like this but we didn't have time to develop a generic physics system for handling other types of terrain, such as water where bodies might float or sink.

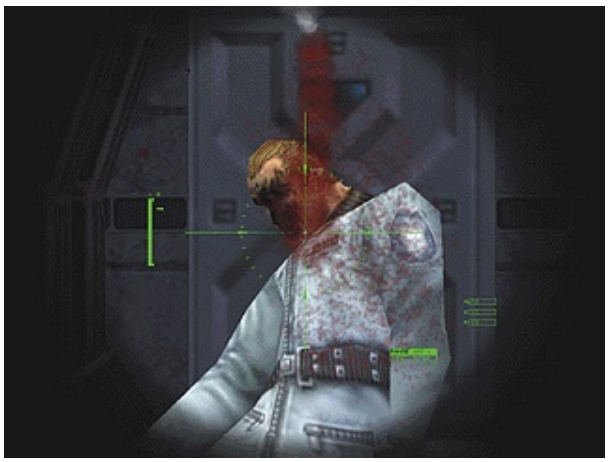
Color, mood, and scale were all considered in concept

art, not just the architecture

The effects editor was created by one of the programmers to help him create visuals for the weapons. The interface, while functional, was crude. Other people wanted to create visuals, including designers, but were hampered by the editor's interface design since it was never intended to go beyond the programmer who created it. Although the in-game editor allowed someone who knew the tool to create a special effect quickly and efficiently, it had a long learning curve for those not familiar with it. This reduced the amount of control that the artists had over the effects.

SoF shared the same scripting language that *Heretic 2* had used. It was originally developed to give designers more control over their levels, but we soon learned that we would need to add more and more power to the scripting system. *SoF*'s complex scripting soon overwhelmed the scripting language and too much time was spent trying to tweak out sequences. With the addition of in-game cinematics (an unplanned feature not included in the design document), we realized that the way we were using the powerful scripting language was wrong. If we had planned better from the beginning, the scripting would have gone much easier. Unfortunately, since the story was planned so late, we didn't know at the time what would be needed.

Originally slated for an 18-month development cycle, *Soldier of Fortune* ended up taking nearly two years, a considerable undertaking that in the end allowed a talented group of developers to really shine. As with all projects, *SoF* had its problems, but for the most part things went well thanks to the efforts of an incredible team of people, and *Soldier of Fortune* has quickly become one of Raven Software's best-accepted titles.



A direct hit.

With strong sales to date and a solid Internet community, *SoF* has exceeded many people's expectations, including our own. We've been very happy for the large number of good reviews, both in print magazines and on the Internet, and we are helping to support the online community as much as we can. From a development viewpoint, *SoF* allowed the Raven team to grow and mature, and many lessons that we learned are now being put to use in our next set of products. Of course, no project ever runs smoothly, but with each new game we gain more understanding of what it takes to make the next one better.

Game Data

Soldier of Fortune
Raven Software
Madison, WI

http://www.ravensoft.com

Publisher: Activision

Full-Time Developers: 20 (on average)

Contractors: 2

Budget: Multi-million-dollar budget

Length of Development: 23 months

Release Date: March 2000

Platforms: Windows 95/98/NT/2000, Linux

Hardware Used: Dell Pentium 550 with 128MB RAM, 18GB hard drive, and a TNT2

Software Used: Microsoft Visual C++ 6.0, Microsoft Visual SourceSafe 6.0, 3D Studio Max 2.x, Softimage 3D, Photoshop notable Technologies: Licensed the Quake 2 engine from id Software (using OpenGL), motion-capture data from House of Moves, force feedback, A3D/EAX 3D sound, World Opponent Network (WON) matchmaking services

Project Size: 406,044 lines of code, 602 files

Return to the full version of this article