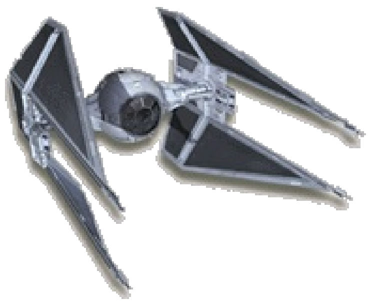
Postmortem: Factor 5's *Star Wars Rogue Leader: Rogue Squadron II*

By Thomas Engel



When I think back to the development of *Star Wars: Rogue Leader*, the first thing that comes to mind is time - or rather the lack of it. Never in the more than 13-year history of Factor 5 have we had a project under greater time pressure than this one.

Many might think that Factor 5's history reaches back only as far as 1996, when the company moved to its current location in San Rafael, Calif., just next to Lucas Arts and Skywalker Ranch.
In fact, Factor 5 was originally formed out of an Amiga hacker group back in Cologne, Germany. In the late 1980s, the Amiga became very popular in Europe, but it didn't have any good action games. It was a port platform, but the machine deserved better; our games, including *R-Type* and *Turrican*, were among the first ones to really push the technology unique to the Amiga.

With the Super Nintendo and Sega Genesis/Mega Drive reinventing the console market worldwide, we moved on to these platforms and got into contact with Lucas Arts, Konami, and Nintendo. During this time, Factor 5 made *Super Turrican 1* and *2* and *Mega Turrican* on the SNES and Genesis, *Indiana Jones - Greatest Adventures* on the SNES, *International Supertsar Soccer Deluxe* on the Genesis, and both *Contra 2* and *Animaniacs* on Game Boy.

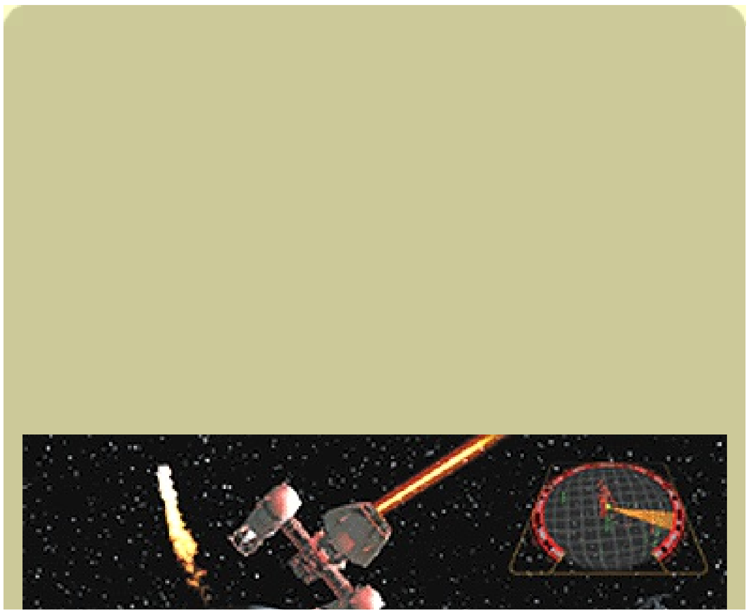
When the Playstation arrived, we started work on *Ball Blazer Champions* and *Star Wars: Rebel Assault 2* for Lucas Arts. However, the 9-hour time difference be-tween California and Germany soon became a problem with CD-based projects. The Internet wasn't fast enough in the mid-1990s to transfer so much data in any practical fashion. We always had to burn versions to a CD and send them via courier.

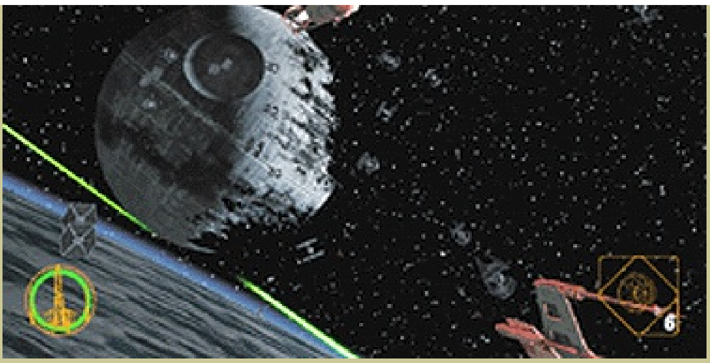
This situation could only go on for so long until Lucas Arts asked us if we might consider moving the company to the U.S. They offered their help in legal matters, and in May 1996 the American chapter of Factor 5's history began.

After finishing *Ball Blazer*, we moved on to our best-known title before *Star Wars: Rogue Leader*, the original *Star Wars: Rogue Squadron*, released in 1998 for the Nintendo 64 and PC. With *Episode I* heading to movie theaters soon after, *Star Wars: Battle for Naboo* was next, followed by our final N64 game, *Indiana Jones and the Infernal Machine*.

Those who saw our Star Wars teaser trailer at Space World 2000 might think that *Star Wars: Rogue Leader* was in development continuously from then until shortly before the launch. In fact, most of the team was busy with *Star Wars: Battle for Naboo* and *Indiana Jones* until late 2000, so we didn't really get started with *Star Wars: Rogue Leader* until January 2001.

Hitting the Gamecube launch meant being done mid-September 2001 - roughly nine months for a 15-month project.




In-game screenshot of a scene during the Battle of Endor. The Death Star featured in the background is actually a big sprite and not a 3D model.

Fortunately, due to our work on the Space World demo and our involvement in the development of Gamecube's audio system, we already knew a lot of things about Nintendo's new platform. While this gained us the invaluable advantage of having a ready-to-use audio driver and some experience on the Flipper graphics chip, we still had many, many things to test and try out - and pretty much everything we did on the hardware was a first.

It wasn't long into the project before six- or seven-day weeks became the absolute norm for everybody on the team. And these were not cozy eight-hour days, either.

What Went Right

1. Think first. The need to come up with a workable schedule seems so obvious, and still it does not work out in so many cases. For us, by far the most important step was to come up with a schedule that - even given all the time pressures we were under - was realistic. This included the overall game concept as well as the details of the technical realization. It was absolutely essential to get level designers, artists, and programmers to talk to each other before final decisions were made, and to keep them talking to each other for the duration of the project.

At times communication broke down, but we always managed to rescue the situation quickly. Our most important strategy to maintain good communication proved to be investigating potential breakdowns in communication at the earliest sign. On a technical level, it was definitely a wise decision not to go for totally new technologies, but rather to employ technologies we were experienced with and use the enhanced power of the new-generation hardware to bring everything to a new level. For example, we used a simple height map to represent planetary surfaces, a technique we already had used in *Star Wars: Rogue Squadron* and *Star Wars: Battle for Naboo*. Our familiarity with the technology allowed us to concentrate on perfecting it while avoiding potentially catastrophic delays in engine development.

2. C++ and other programmer toys. Nothing beats a clearly structured project from a programmer's point of view, and using C++ can be a great tool to achieve this. We took the time to define up-front the class hierarchies and other guidelines for all the programmers working on the project.

Setting the basic concept for the game in stone very early in the project and assigning clear areas of responsibilities to each programmer introduced a clear structure, and C++, with its protected class members and type checking, helped greatly to keep the structure intact. Since the language itself provided the tools to reinforce the structures defined in the beginning, we were able to minimize the amount of work necessary to maintain orderly source code. This freed up time for the leads to attend to other tasks, and also helped a lot in bringing down the number of reported bugs during testing.

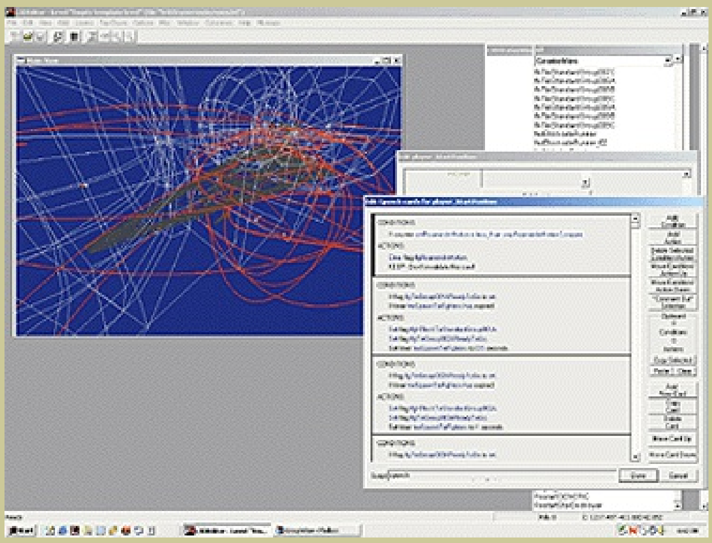
Although we added it in the final month of the project, writing our own virtual memory kernel on top of the OS was another decision that proved to be very helpful in the end. One might ask what virtual memory might be used for on a system that features no writeable mass storage device such as a hard drive.

Dealing with Gamecube's two-part memory architecture, which has 24MB of "fast" (very fast, actually) RAM and 16MB of "slow" RAM that is pretty close to a small ROM cartridge in terms of access and speed, can be a bit of a hassle. This is especially true if one has to make multiple subsystems - implemented by multiple programmers - using the ARAM at the same time. Using the main processor's virtual memory unit, we mapped a section of the ARAM area into the normal address space. We ended up using this dynamic mapping system to avoid having to deal with code overlays by moving code into this virtual memory area, as well as to make access to data in ARAM much easier and more flexible then with manual ARAM DMA transfers.

The time implementing this system was well spent. The last weeks of development saw a number of situations in which we would have lost hours and hours implementing specialized code, but instead the virtual memory system took care of all of them nicely.

3. Scripts without scripts and other level designer magic. The level designers where greatly aided by our proprietary scripting language, CPunsh. CPunsh handles the tasks of a classical scripting language without really being a language in that sense. Rather then implementing a classical computer language, we designed CPunsh as a drag-and-drop-based system of virtual cards. Each card contains a collection of instructions or decision points. The idea behind all this was that we hire our level designers for their expertise in designing fun levels, and not so much for their understanding of programming.

CPunsh's design, while not perfect, in fact helped avoid a lot of bugs in the scripts authored by the level designers and also made them easier to debug if problems occurred. Another bonus was that our level designers already knew the system. It had been part of L3D, our in-house level editor, since *Star Wars: Battle for Naboo*. While we had to add some additional features to support *Star Wars: Rogue Leader*'s new AI system and grander scope, the knowledge that the level designers already had accumulated while using the system earlier proved to be of great help.



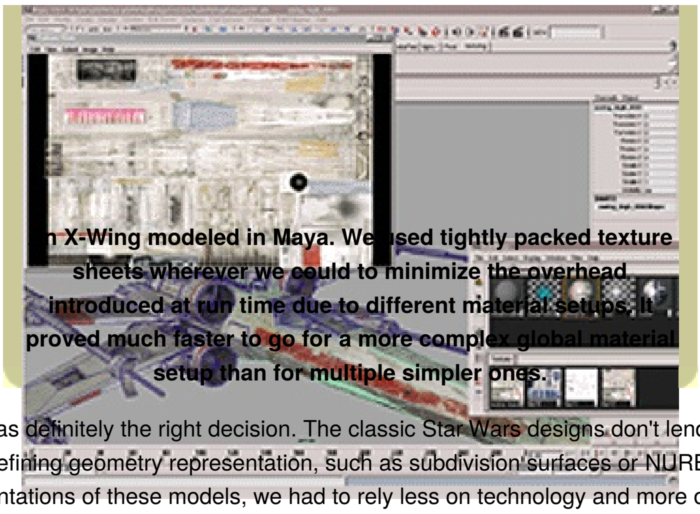
Rather then implementing a classical computer language, we designed CPunsh as a drag-and-drop-based system of virtual cards, each containing a collection of instructions or decision points.

Star Wars: Rogue Leader's completely rewritten AI system offered a whole new set of possibilities to the level designers. On the N64 we always had to be overly performance- and memory-conscious. Both *Star Wars: Rogue Squadron* and *Star Wars: Battle for Naboo* used close to nothing else but enemies that were running along on predefined splines. This made it quite difficult for level designers to control large quantities of enemies and also make it seem as if the enemies actually would react to the player's actions. With *Star Wars: Rogue Leader*, this system got its long-overdue revamp. In this title, enemies are still guided by splines, but most of the action is controlled by flocking and other algorithms and is highly aware of the player's actions. The added creative freedom for level designers was truly a great asset.

4. Art: Painting by polygons. One thing our work on the original Space World demo really helped with was to get a thorough understanding about the basic art requirements of this title. The demo offered us a test run in terms of getting artwork out of Maya into the run-time engine. Only the basic animation and geometry pipeline developed for the Space World demo ended up in the final product, but this proved to be a key asset in speeding up the development of the shader data path later on, since we didn't have to start entirely from scratch.

Both the programmers and the artists had a clear understanding of what they wanted, and the specifications for geometry in particular were clear long before the bulk of the team moved over to the development of *Star Wars: Rogue Leader*. This technical groundwork, together with the exceptional work of all the artists on the team, helped *Star Wars: Rogue Leader* achieve the visual quality one sees in the final product.

One of the strengths of Gamecube's hardware is multi-texturing. Using well-understood techniques for our geometry representation and generation, we decided to concentrate our efforts on the texturing aspect.



An X-Wing modeled in Maya. We used tightly packed texture sheets wherever we could to minimize the overhead introduced at run time due to different material setups. It proved much faster to go for a more complex global material setup than for multiple simpler ones.

With respect to craft models, this was definitely the right decision. The classic Star Wars designs don't lend themselves too well to the modern ways of compressing and refining geometry representation, such as subdivision surfaces or NURBS, due to their boxy and angular structures. To get accurate representations of these models, we had to rely less on technology and more on first-class modelers.

For the landscape, which was represented by a height map, the texturing was the single most important aspect of all. Only with multi-texturing was it possible to achieve the organic and natural look we were going for. The landscape texturing consists of multiple layers of repeating, general patterns. The trick was to combine all these layers with what we called "mix-maps," a set of simple grayscale textures that defined how the different types of patterns were to be combined. To add even more flexibility, we also allowed the mixmaps and patterns to be rotated against each other. Besides offering good looks, the use of mixmaps also gave the textures a small memory footprint, since we could easily hide the repetition of the patterns with clever setups for the mix-maps. Bump and detail maps finished off the effect.

All these texturing technologies were integrated either into L3D, our in-house level editor, or Maya's shader controls. This way, the artists and level designers had an easy-to-use interface in which to create all the texture artwork.

Some issues remain to be solved for our next game. For example, there was no fast and easy way for the artists to preview their work on the real hardware. Unfortunately things look quite different on a PC monitor than on a 6403480 TV display, but we were still pleased with the results.

5. Making some noise. With regards to audio, we had a good start. Factor 5 had an important role in the development of Gamecube's audio hardware, and we developed the MusyX audio system in-house. Because of this, we were able to build the audio part of Star Wars: Rogue Leader on a solid, fully understood API and tools.

On the creative side of things, it helped a great deal to have access to the Lucas Arts and Lucasfilm archives. While a lot of effects and post effects for voice recordings had to be redone and redesigned by our sound effects designer, having access to this data was very important in keeping things sounding authentic. We are in the lucky position of having two dedicated and very experienced sound designers at Factor 5, assets that can't be overvalued if one has to work on a tight schedule.

We also implemented a little tool that allowed the sound effects designer to mix the audio much like a sound designer in a movie would do. Using this tool, the sound designer is able to manipulate most parameters of sound effects in the game while the game is running. Since mixing sound effects is as important as designing their basic characteristics, this tool was truly worth the work spent on it.

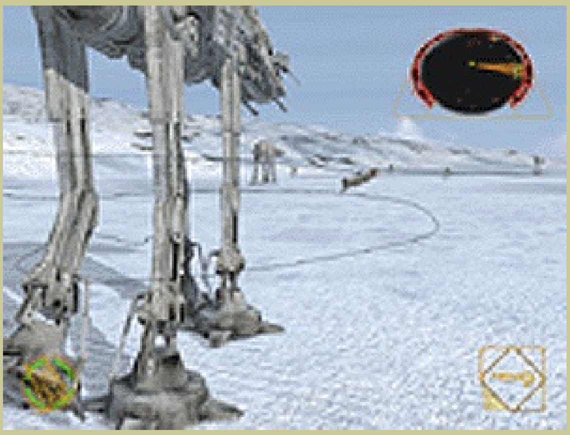
Star Wars: Rogue Leader was the first game ever to feature a Dolby Pro Logic II (DPL2) surround sound encoder. When Dolby showed us the results one could produce using DPL2 while staying 100 percent compatible with the widespread Dolby Pro Logic system, we were truly amazed. Because we started with such a solid audio base, we could invest the time to develop our own DPL2 encoder for use on Gamecube.

What Went Wrong

1. Make data and coffee? Some data conversion runs took forever. Since everybody in the team to some extent depended on the data conversion, the impact of long data conversion runs multiplied quickly.

There were multiple reasons for the slow speed, most of which can be traced back to two fundamental problems. First, we did not implement a global-caching system for converted data. Because of this, multiple people would convert the exact same data on their local hard drives while the converted data was actually already distributed around in the company. For some data types, such as textures and shadow maps, we later introduced caches on the server, but a more global scheme would have helped greatly.

The second major problem had to do with interdependencies between diffrent data types and even program code. Inter-dependencies between data resulted in rebuilding of a lot of data, even if just small amounts of data changed. This could have been avoided with a different setup for the data, and perhaps by the introduction of some kind of linker for the data conversion. All this never happened for this project. Nobody had the time even to plan for something at the point when the problem became obvious. From this we learned one thing: Do not underestimate the amount of data one has to handle in a game for the (now) current generation of game consoles.


A Snow Speeder is busy "cabling" an ATAT. A physics simulation drove the cable's movement.

2. Cutscenes. As we did in our earlier titles, we wanted to avoid using FMV for cutscenes, to avoid breaking the continuity and style of the game. The playback of these animations was handled quite elegantly by our standard run-time animation system, which we used for all types of animation throughout the game. The internal structure of this system closely resembles Maya's animation system, which made it very flexible and straightforward to control from within that tool. It also was really easy to use at run time. This is where the fun ended, however, and things started to get ugly.

On the programming side, we had to spend a lot of time organizing the data flow. We wanted to make the amount of data to be loaded or kept in memory for cutscenes minimal, which meant recycling data that was present in the level data currently loaded. This sounds easier - and we fell for it, too - as it is actually was. However, our main problem was that we weren't able to begin implementation of the cutscene playback until pretty late into the project, and the programmers responsible for the task were confronted with a data structure that was pretty much set in stone. This is the one area in which our planning at the beginning of the project did not quite work out.

In addition, the animators could not start until relatively late in the project. This was due to both the unfinished system implementation as well as their simply being occupied with other tasks. This time frame put the animators under a lot of pressure. To make things even worse, they had to suffer quite a bit under the slow data conversion. The cutscene data included so many cross references into other types of data that changing a cutscene frequently meant rebuilding major parts of the data set.

Another thing that slowed us down was the fact that we could only export part of a level's data into Maya as a reference for the animators to work with. In particular, the long cutscenes in the Hoth level, featuring many close-to-ground camera moves, presented a challenge, since the height-map geometry export into Maya always used the lowest level of detail and hence didn't represent enough detail for such tight corner moves.

Previewing music and sound effects for cutscenes proved to be another hassle. The effects and music were triggered from within Maya but could not be previewed there, which meant going through the data conversion step each time we wanted to test things.

More preview capability and a faster data conversion would have solved our problems. It also wouldn't have hurt to have more time, a luxury we simply couldn't afford this time around.

3. What's that engine glow doing on the nose cone? Little things can cause big problems when time is as much a factor as it was with this project. The method we used to add visual effects, like engine glows to the in-game craft models, introduced a high dependency between the code versions used to edit new levels and preview models, and the model data itself. A simple change to the model within Maya could trigger engine glow to appear in the completely wrong spot.
This made it close to impossible for the artists to judge whether they actually did something wrong while exporting the data or whether this was just one of the cases where everything was actually in order and only the code version needed an update. This situation was truly counterproductive in the last weeks of production.

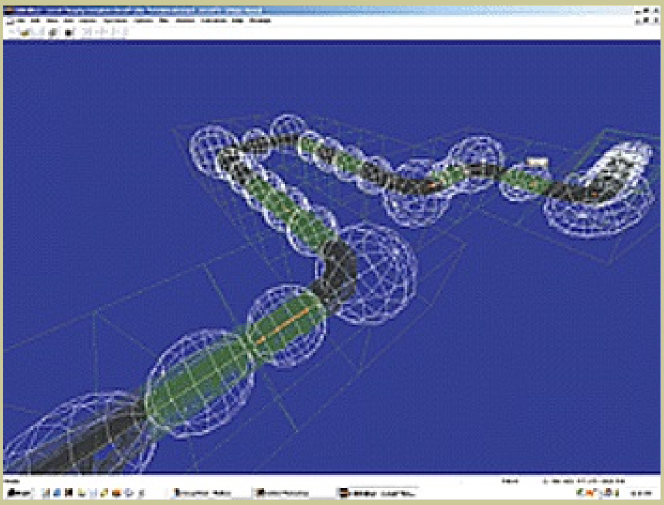
We will have to remove this dependency completely from our next game's data path. To some degree, this will be a side effect of our effort to speed up the general data conversion by removing data interdependencies.

4. "I've got some new songs." While we had pretty good turnaround times getting completely new sound effects into the game - as much as in-game material was concerned - the data path and the way the music was linked into the game did not allow for any fast preview cycles for streamed or MIDI-based pieces of music.

Due to time constraints, we had been unable to implement tools that would have allowed adding or changing pieces of music without

rebuilding the program code. Furthermore, most pieces of music are triggered from scripts designed by level designers, which meant we always had to coordinate three people in the process: a programmer, a level designer, and the musician.
In the end we coped with the problem to some extent by simply setting up a schedule defining when a new music update would be done. The musician was able to audit his score without the integration into the game on a prototyping tool at any point in time and with minimal turnaround times. The schedule made the problem somewhat manageable for all parties involved, but it was still a far cry from being a good working environment.

5. L3D: Trusted, proven and, well, a bit rusty. L3D, our in-house level editor, had been around for many years when we started *Star Wars: Rogue Leader*. The good thing about this was that the level designers knew the tool and it had gone through a lot of iterations of improvements during the development of *Star Wars: Rogue Squadron* and *Star Wars: Battle for Naboo*. But we also knew that it wasn't exactly designed to deal with data sizes as they regularly pop up with this new generation of consoles. We simply didn't have the time to program a new tool, since level designers had to start their work early. We just hoped for the best.


The tunnel sequence of the fight for the second Death Star as seen in L3D. The boxes are added to ease the culling process, while the spheres control certain aspects of the gameplay like triggering of events or lights..

Some things got very slow as things got very large in terms of memory. For the most part we could get things to a workable level again by running the tool on relatively high-end systems. Other things just had to be endured by the level designers. The situation was far from perfect. Despite its shortfalls, however, we couldn't have made it without this old, trusted, and slightly outdated tool. Still, a new, streamlined tool would have been much faster and more user-friendly, which in turn would have given the level designers more time to actually design things. As a logical consequence, this is what we are currently working on. With a bit more time on our hands for finalizing our next title, we took the opportunity to take the things we learned from L3D and rewrite a new tool from scratch.

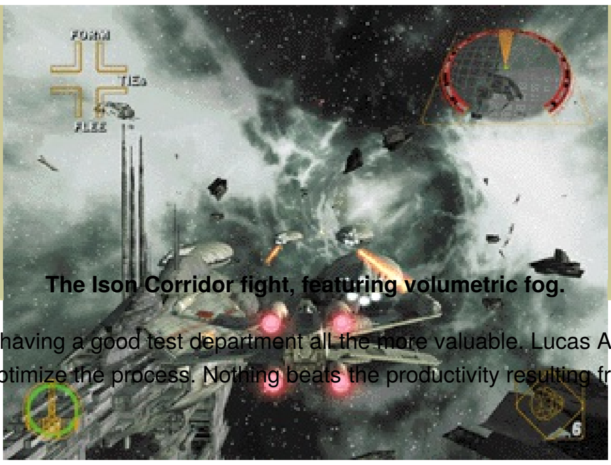
Final Thoughts

It was a ride. It was also a stressful, demanding period in all of the team members' lives, but still an exciting experience that probably nobody on the team would have missed for anything in the world - so long as it's guaranteed not to be repeated anytime soon.

Never have I seen a team so dedicated to a game project than the *Star Wars: Rogue Leader* team. Everybody had just one aim: make the best game possible. Working in a small company without any major bureaucratic overhead also definitely helped to keep everybody going at full speed.

We couldn't have done it without help, though.

The full and unconditional support from Lucas Arts was essential in keeping everybody focused on the task. You don't want to worry too much about your relationship with your publisher when you have a plate as full with other tasks as we had.

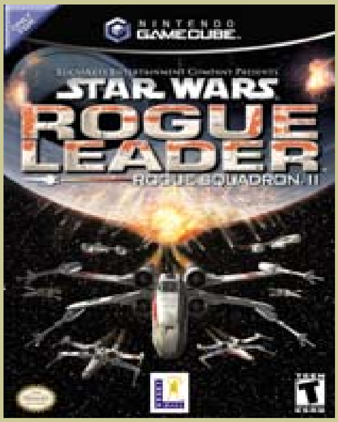


The Ison Corridor fight, featuring volumetric fog.

Also, working on a tight schedule made having a good test department all the more valuable. Lucas Arts' test department really supported us, even moving into our offices to further optimize the process. Nothing beats the productivity resulting from testers always being available on a moment's notice.

Last but not least, we enjoyed the professional and dedicated support of both Nintendo Technology Development and Nin-tendo's Software Developer Support Group. I can't stress enough how important their input was in getting us up to speed on the new hardware.
Looking at the response the game has triggered so far, we seem to have done a pretty good job. In the end, only time will tell whether the game lives up to what people expected from it.

Game Data



Rogue Leader

Publisher: Lucas Arts Entertainment

Number of full-time developers: 30

Number of contractors: 2

Estimated budget: $3.5 million

Length of development: 9 months

Release date: November 8, 2001

Platform: Nintendo Gamecube

Development hardware used: GDEV & 1GHz PC, running Windows 2000

Development software used: SN Systems for Gamecube, Slickedit, Maya

Notable technologies: MusyX 2.0

Project size: 14.2MB of source in 859 files, in-game source data 6.4GB in 10,075 files

Return to the full version of this article