GAMASUTRA
The Art & Business of Making Games

PRINT

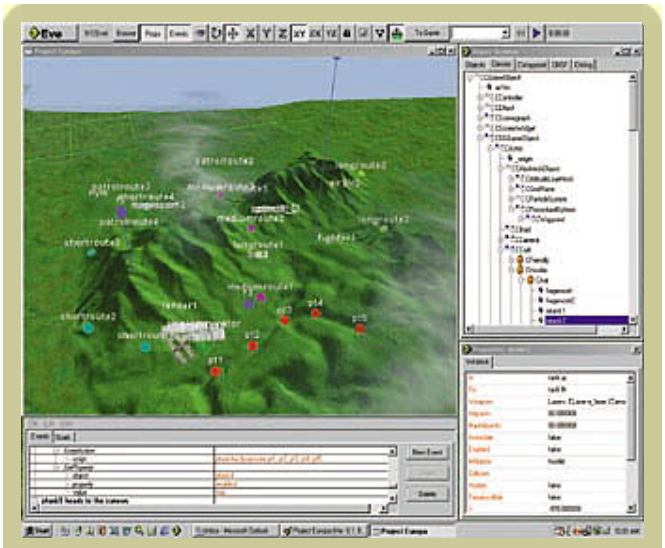
Postmortem: Lucas Arts' *Star Wars Starfighter*

By Chris Corry

Work on Project Europa — the internal codename for the development effort that would eventually metamorphose into *Star Wars Starfighter* — began in earnest in April 1998. A small crew of programmers, headed up by director Daron Stinnet, began preproduction work on a *Star Wars: Episode I* PC title that had grand ambitions. As one of Lucas Arts' great unsung talents, Daron had previously led the *Dark Forces* and *Outlaws* teams to much critical and commercial success. Now, following in the footsteps of Larry Holland's *X-Wing* games, Europa was to bolster Lucas Arts' presence in the space-combat genre and support the new film franchise. While embracing much of the *X-Wing* series's simulation-oriented aesthetic, the team also wanted to deliver the visceral, sweaty-fingered arcade experience that we were starting to see in early builds of *Rogue Squadron*.

During the early months of 1999, a well-known designer who was in the market for a new lead programmer and lead level designer for his company's overdue project secretly approached two members of our team about the possibility of jumping ship. Although obviously conflicted, the allure of working with a famous industry heavyweight proved too tempting, and within a few short weeks, we had lost our main graphics programmer and level designer. Shaken but undeterred, we were determined to make the best of a bad situation, but three months later, the project suffered another blow when we lost our second graphics programmer.

This was Europa's darkest hour. The technology development was progressing slowly, and our inexperienced programming staff was still climbing the C++ learning curve. As lead programmer, this predicament was largely of my own making. I had joined Lucas Arts from outside the game industry, where I was accustomed to a corporate R&D environment that valued solid engineering and extensible software architecture over quick solutions that were perhaps less elegant or flexible. Now, with little to show but a creaky Glide-based graphics engine and no graphics programmer, we were at a loss as to what to do next.



The Eve level design tool was a critical part of Star Wars Starfighter's success.

As if things weren't bad enough, we were also floundering on the game design side of the fence. Although we had a lot of excellent concept art, few of us had a clear idea about exactly what type of game we were making. We were painfully starting to discover that while it is easy to characterize a title as being a cross between *Rogue Squadron* and *X-Wing*, it's another thing completely to describe what that actually means.

At this point two events occurred that I'm convinced saved the project. Our multiplayer programmer, Andrew Kirmse, who had already proven himself as a remarkably capable technologist, teamed up with two of our other programmers to create a graphics-engine "tiger team," a small subteam dedicated to attacking a single task with unwavering focus. In just a few months the three of them delivered a brand-new OpenGL-based engine that was far better than anything we had built previously.

Shortly after the new graphics engine came online we also found the solution to our game design woes. Tim Longo, who had recently helped complete *Indiana Jones and the Infernal Machine*, joined the team as our lead level designer. The change was immediate and profound; five other level designers joined the project at about the same time, and now we had the foundation for a thriving, collaborative design process.

Daron worked with Tim and the other level designers on an almost daily basis, systematically identifying areas of the game design that were incomplete and working together to come up with concrete solutions.

An example of the statistics that Daron tracked during the game's development, and a page from the programming section of the Star Wars Starfighter internal web site.

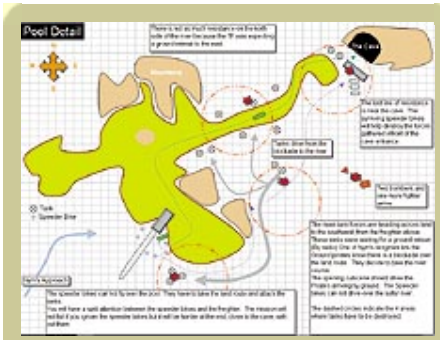
By the end of 1999, the project had performed a 180-degree turnaround, but there was one more significant twist in the road awaiting us. Sony had turned the game industry on its ear with the formal announcement of the Playstation 2 that year, and every major development house was furiously rewriting business plans to accommodate support for the new platform; Lucas Arts was no exception. The biggest problem for the company was that we wanted to have a title close to the system's launch, and Europa was the only project far enough along to be a serious candidate. The thought of throwing the PS2 into the mix made many people very uncomfortable, but when we were able to port all of our nongraphics code in a single 48-hour period, senior management became convinced.

The rest of the project was an exciting and manic blur of activity. Early in 2000, we hit the "snowball point," that period when all of a sudden the tech falls into place, the art production paths are running on all cylinders, and the team is seeing exciting new gameplay on an almost daily basis. From then on, *Star Wars Starfighter* was indeed like a runaway snowball, picking up momentum and new features almost as fast as we could think of them.

What Went Right

1. Good team communication. I've read many *Game Developer* Postmortems that blamed failures on a lack of communication, so I'm particularly proud that we got this one right. From the beginning of the project, Daron worked hard to impress on the programmers that it was the level designers and artists that would ultimately secure the success of Europa. As I became fond of saying, programmers build the picture frame, but it's up to the rest of the team to provide the most important part, the picture.

To bring this to fruition, the programming team needed to understand as best we could the way the rest of the team worked. While Andrew worked with lead artist Jim Rice and our world-builders to understand their workflows, Brett Douville, our AI and mission programmer, filled a similar role with the level designers. Brett scheduled regular "LD Days" with each individual member of the level design staff. This gave each designer the opportunity to meet with Brett on a regular basis and show him the specific challenges and problems that they were tackling in their missions.



This image depicts the design

This image depicts the design schematics for one of the Lok missions. Level designers made elaborate plans such as these for every level that appears in the game.

Europa periodically had full-blown team meetings where we could get together and kibitz about the overall state of the project. However, the most valuable meetings were at the subteam level. Both the programming team and the art teams would meet weekly to discuss the issues of the day, and each of these meetings would have an attendee from the other camp -- a role we referred to as the "exchange student." This meant that if questions came up in the art meeting, for example, that required answers or input from a programmer, there would always be someone present that could give an informed opinion. Likewise, as programmers would discuss issues or new features in their weekly meeting, the art or level design representative would be able to disseminate this information among the other team members.

Finally, we relied on an internally maintained web site as a pivotal communications tool. We tried to make the site as comprehensive as possible, organizing areas along the lines of programming, art, level design, project management, and so on. When artists had questions about how to implement a particular effect, or a level designer needed a refresher on our class-file script syntax, there was usually a web page that they could be directed to that would answer many of their questions.

2. Project discipline. *Star Wars Starfighter* was a well-organized project. In the heat of battle it's all too easy to let requirement lists and schedules get lost in the shuffle of the moment. We were determined not to let this happen. As soon as our technology began to take shape we started to follow an iterative process of milestone planning and execution. These milestones were typically four to six weeks in duration, with no milestone extending longer than eight weeks. Milestones were also required to demonstrate some visual or gameplay aspect of the game. As a consequence, we had very few milestone tasks that looked like "complete the Foobar class"; instead we would have a milestone task that might read "Explosion smoke trails," and the assigned programmer would know that completing the Foobar class was an implied requirement. By keeping our attention focused on a discrete and relatively small body of work, we were able to avoid the cumulative errors that invariably creep into longer schedules, while still allowing for demonstrable progress.

Most of the milestones were driven by the progress of the technical team. Programmers were solely responsible for estimating the duration of their tasks. We would occasionally adjust these estimates outward but would never change an estimate to be shorter. Tasks were structured so that the shortest scheduled task was never shorter than a half-day. Even if a programmer was certain that a task could be completed in less than half a day, experience clearly showed that the time would be lost elsewhere. Using these simple rules of thumb, we were consistently able to build schedules that were fair and accurate. Out of eight scheduled milestones, we never missed one by more than a handful of days. Best of all, most team members completely avoided extended periods of crunch time. Like most game teams as they approach their ship date, everyone was working hard and often into the evenings; however, this period of time was short, and we never had to resort to all-nighters.

We also closely managed the process we used to distribute new binaries to the team at large. Since most of our development occurred on the PC even after making the decision to ship on the PS2, it was important that team members have timely access to stable builds of the game. We accomplished this through weekly public builds. Once a week we would package and distribute the current code as a full-blown Install Shield-compiled install. This provided team members with debug and production versions of the game, along with level design tool and art exporter updates. Predicting that public builds would become critically important, we tried to be as ruthless as possible about maintaining the build schedule. As we got closer to our ship date, the frequency of these public builds increased until we were performing new builds as often as three times a week. By this point we had a full-time staff member dedicated to managing the public build process and ensuring that the distributed code met quality and functionality expectations.

3. A well-executed PC-to-PS2 transition. Making the decision to move the project to the PS2 could have been a complete disaster. Yet, despite paying little attention to portability during the earliest stages of the project, the Europa code base was well positioned to make the jump to the PS2 platform. With the aid of strong and generally stable development tools provided by Metrowerks, the core port went off without a hitch. The biggest trick was on the graphics side, because this was clearly where we were most vulnerable. None of our programmers had any console experience, and none of us was up to the task of tackling the PS2's infamous low-level vector units. Enter LEC gl.

LEC gl was the brainchild of Eric Johnston and Mark Blattel, two of Lucas Arts' most senior console programmers. They had recently shipped Star Wars: Episode I Racer for the N64, and they welcomed the opportunity to tackle a problem temporarily that was one step removed from the day-to-day pressures of a project team. Although Europa was the most immediate recipient of their efforts, Eric and Mark were never officially on the project. Instead they worked in a support role, providing us with regular LEC gl library drops and immediate "on-call" PS2 graphics support.

There was another, more subtle problem that we had to conquer when we made the decision to adopt the PS2 as our primary platform. Most of the team members were big PC game players, but very few of us played console games. Intellectually, we knew that there were huge philosophical differences in game design between consoles and the PC. Much of our original game design had used the *X-Wing* games as a conceptual leaping-off point, wandering into the arcade action of *i* only when it suited us. Now that we were on the PS2 we recognized that our design priorities needed to be completely flipped. Instead we would use Rogue as our primary point of reference and work from there, layering on gameplay elements borrowed from *X-Wing* as needed. As such, I think the final game demonstrates our successful indoctrination into the console mindset. We were having so much fun blowing things up that we had little desire to start adding sim-like features to the

gameplay experience.

4. Macromedia Flash. As we approached the end of summer in 2000, we realized that we had a serious problem on our hands. Despite our best efforts, we still had not addressed the issue of our out-of-game user interface. We had a 2D virtual-page system that we were using for our HUD (heads-up display) symbology, and we had always planned to evolve that into something that could be used for what we called the "administrative interface." However, in August, with the quality assurance department nipping at our heels and our ship date looming ominously in the distance, things were not looking good.

We had heard that a small San Francisco-based company named Secret Level was adapting Macromedia's Flash technology for use in PS2 games. After meeting with company representatives, we were excited by the prospect. The Macromedia content-authoring tools were far more elaborate than anything we could come up with in the same time frame. We also suspected that there was a wealth of Flash authoring expertise available from out-of-house contractors, which would help us smooth out the work load. Most importantly, we were very impressed by the intelligence and games savvy of the Secret Level staff. When we realized that building our user interface in Flash would significantly ease our localization efforts, we decided to take the plunge.

Early concept art for the mercenary Vana.

Soon afterward, we hired a design firm named Orange Design to help us implement our administrative interface in Flash. Orange not only had a ton of experience with Flash, but they also brought a technical perspective to the table. We knew that this technical emphasis would be critical for working with our programming team on integration issues.

Integrating Flash into the Europa engine was not a completely smooth process, however. Performance in the first-generation Flash Player was poor (current generations of the Player are now much faster), and we had to spend a lot of time integrating the user interface Flash movie with the core game systems. That said, the five months that a single half-time programmer spent on this task ended up yielding a user interface that was far beyond what we would have been able to custom-code in the same period of time.

5. Good debugging systems. Our programmers built several tools that greatly helped our pursuit of high-quality code. One of the most instrumental was a Windows-only library that provided detailed stack-tracing information. This library was largely based on the code and concepts covered by John Robbins' "Bugslayer" column published in the Microsoft Systems Journal.

As is standard practice on many games, we built a custom memory manager that could detect when the application was leaking memory. However, unlike most implementations, when our memory manager detected a leak it could provide a comprehensive stack trace of arbitrary depth, leading directly to the leaking code statement. This capability represented a significant advantage over other implementations that could only provide the immediate location of the allocation request. If the memory allocation was being made by the Standard Template Library (STL) or one of our widely used utility classes, it was usually not enough to know what part of the STL or which one of our utility classes was the culprit. What we really needed to know was what class called the STL method that caused the leak. In fact, the leak was usually several steps up the call chain. Our stack-tracing library made finding these cases almost trivial.

We also incorporated stack tracing into our exception- and assert-handling systems. When the game encountered a hard crash, we trapped the exception and generated a complete stack trace; a similar process occurred when our code asserted. This information was initially reported back to the user in dialog form. However, we also packaged up this same data and had the game send the programmers an

e-mail detailing exactly where the problem occurred. This ended up being an invaluable tool for us. As a matter of practice, the Europa programmers got into the habit of checking the assert mailbox regularly. In addition to appraising the current stability of the code, we could also use this data to spot trends and note when people weren't being diligent about installing new builds.

In the end, we had an exceptionally smooth QA process because the bugs we did have were generally easy to track down and fix. There were no last-minute "heart attack" bugs that required us to set up camp and track a single problem for hours or days at a time. This made life easier on the programmers, but it also made things easier for the testing team and improved morale across the entire project.

What Went Wrong

1. Staffing. As you can probably tell by now, staffing was easily the biggest problem the project encountered. Try as we did to manage staff retention, the team experienced an alarming amount of turnover, both in the programming and art departments. This invariably made life harder for the people left behind, because the amount of work remained constant, but team members could not be replenished as quickly as they were lost.

This also meant that many of the team's junior staff members missed out on valuable mentoring or experienced spotty supervision by their leads. On the programming team, senior programmers were so busy that we had little time to train new team members. This led to a stressful sink-or-swim mentality, which was difficult for new hires. Even relatively simple quality-control procedures such as code reviews were never instituted, since every moment of every day was dedicated to making forward progress on the game.

The staffing issue continued to dog us throughout the project. Even after we had regained some momentum, we still ended up losing two programmers and a handful of artists, all to the same online gaming startup. Although nine programmers contributed to the main code base at one point or another, the vast majority of code was written by the core group of four programmers who stayed with the project to completion.

2. Initial lack of detailed design. Europa was always envisioned as having some sort of Star Wars: Episode I tie-in. During much of 1998, however, it was difficult to predict to what degree Lucas Licensing would allow this to happen. One of the barriers we encountered was the intense veil of secrecy that surrounded any Lucas-owned company involved with the movie property. Some of us had access to the script and the occasional rough-cut screening, but particularly during the first half of 1998 it was virtually impossible to learn the important details about the film needed to build a solid franchise title.

Several of the ship models developed for *Star Wars Starfighter*, and an early screenshot of the Havoc pirate bomber being chased by a Naboo Starfighter.

Initially we had assumed that the game should stay as far away as possible from the events of the film. Because we were going to be telling one of the first original stories set in the time line of the new film, we had no feeling for where the boundaries were with respect to planets, characters, vehicles, and the like. We were intimidated by the pervasive atmosphere of secrecy and general sensitivity of the *Episode I* story lines; the first game designs described a pirate war far divorced from the events of the film. In fact, the Naboo Starfighter was one of the only elements that could be found in both the first design and the film. As this design started to circulate, however, Licensing contacted the team and explained that the design contained too many pirate elements; they wanted the game to contain more elements from the film. The "moving target" nature of this exchange ended up being very disruptive and effectively paralyzed the design effort for weeks at a time as we wandered from idea to idea, wondering what fit into continuity with the film and what was straying into areas that we should keep away from.

The Europa team also had some pretty big shoes to fill. It didn't take long for us to realize that whatever we did was going to be directly compared to Larry Holland's previous *X-Wing* titles. The Totally Games guys had been making games like for this for the better part of a decade, and they had gotten very, very good at it. Game players could rely on Larry to produce large, sophisticated games with well-designed features and compelling gameplay. This success had, in turn, incubated a dedicated and enthusiastic fan base that we knew would mercilessly scrutinize *Star Wars Starfighter*. Frankly, we were in a no-win situation: if we deviated too far from the Totally Games designs, we risked disenfranchising some of our most loyal fans, but we also didn't want simply to copy Larry's last game either. Fortunately, once we decided to ship on a console, the design shackles fell away and we were free to chart our own path. While we realized that the hardcore *X-Wing* players might not appreciate Star Wars Starfighter as much as the Larry Holland games, they were no longer our primary audience.

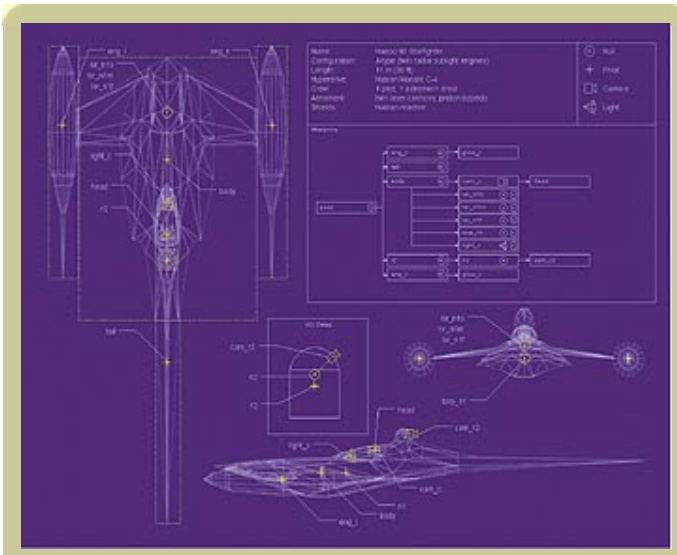
3. Naïve approach to memory usage. As quickly as we were able to get Europa up and running on the PS2, it took the programming team much longer to fully embrace the creed of the console programmer. Since Europa was originally intended to be a PC title and our programmers only had PC experience, it's not surprising that most of the code suffered from a bad case of "PC-itis." I use this term to refer to programming practices that, while potentially portable to a console, are definitely not console-friendly. Our approach to memory allocation is a perfect case in point.

For starters, we relied on the STL for all of our container classes. On one hand, we benefited from a bug-free and robust set of standardized collection classes. As an integral part of the C++ Standard Library, the STL contains a powerful toolset for general application development.

We're big fans of the STL, and for the most part we can't imagine working on a project that doesn't use it. Unfortunately, depending on what containers you decide to use, the STL is notorious for making many small memory allocations. Our STL container usage was paralleled by our use of an uncomfortably large number of ANSI string objects. The ANSI string class is a great little class that makes dealing with character strings much easier than it used to be when we were all writing code in C. Like most STL containers, however, excessive use of the string class also leads to large numbers of small memory allocations. By the time we decided to port to the PS2, most of the damage had already been done.

As I mentioned earlier, our global memory manager's original focus had been memory-leak tracking, but now we needed it to help with our STL problem. We accomplished this by introducing the concept of bins, which were really just a hierarchy of fixed-length memory allocators. When the memory manager received a small memory request, it could very quickly and efficiently satisfy the allocation if the size of the request fell into the range serviced by our bins. We ultimately relied on the bins for both rapid memory allocation services and fragmentation management.

I should also note that we had a pretty rough time with memory fragmentation. Going into the PS2 port we suspected that fragmentation was going to be a problem. On the PC we had made an effort to generally clean up after ourselves in ways that would help reduce fragmentation, but we never made a concentrated effort to eradicate it completely, because we knew that in a pinch we could always rely on the PC's virtual memory system. One of my jobs during the last six weeks of the project was to build debugging systems that would give us detailed memory maps and then track down each fragmenting memory allocation one at a time. It was every bit as unpleasant as it sounds, and I urge those PC developers making the switch to consoles to take this lesson to heart.



A schematic for the Naboo Starfighter—one of the only elements in the game that was present in both the original design concept and the final product.

4. Not enough attention paid to performance. There is little question that in the rush to implement features and ship the game on time, performance suffered. Part of this was due to having an inexperienced staff, and part of this was due to the fact that we had ported a PC code base to the PS2, but in truth most of us were so preoccupied with one issue or another that we had little time to revisit code with an eye toward optimization.

There was a pervasive attitude among many of us that we could safely ignore code problems until they showed up as hotspots on a profiling run. There is some merit to this strategy, since premature optimization efforts can be more wasteful than not fixing the code at all. But since profiling can turn up hidden problems in areas of the code that the team had previously thought complete or issue-free, it's important to start profiling much earlier than we did. For example, we had severe performance problems in our collision detection systems that we would have identified immediately if we had profiled sooner. As it happened, by the time we realized that collision detection was working poorly, the best we could do was apply spot fixes instead of the large-scale reworking that the problem actually demanded.

Even after we started a fairly regular regimen of profiling late in the development cycle, we still didn't do enough of it. In the end, only one

programmer did all of our profiling, and he was responsible for making the rounds and pointing out problems to other members of the programming staff. This was a real shame, because the Metrowerks PS2 profiler is a very nice tool, and most members of the team had uninstalled licenses. I should have made our developers responsible for profiling their own code and doing so at a much earlier stage.

5. Space-to-planet. If there was anything about the original *Star Wars Starfighter* pitch that met with widespread enthusiasm, it was the idea of seamlessly transitioning from planet-side environments to the depths of space and back again. Dog fighting close to the planet surface certainly has its own appeal, but there is something about the promise of being able to pull back on the stick and blast off all the way into space that is simply very, very cool. This high concept was so exciting to the team that the original game pitch featured this idea predominantly. In fact, in many ways this single feature was to define the game.

Well, it's a bit of a trick to actually pull off. First, there were the technical considerations. A planet is big. I mean really, really big. Even a small planet would require dynamically creating thousands of terrain tiles. Although most of these tiles could be procedurally generated, they would still need to be created and discarded on the fly; depending on the player's location, custom mission-area tiles would have to be streamed in from the hard disk, all while maintaining frame rate. Of course, since we wanted to allow the player to fly absolutely anywhere on the planet, ordering this data on the disk in a streaming-friendly format was problematic. We exacerbated the situation by requiring even our lowest-resolution terrain height maps to be much higher resolution than they really needed to be. This in turn made higher theoretical demands on the streaming and resource systems.

This single feature had introduced a tremendous amount of technical risk to the project, and yet we had blindly charged ahead anyway because of the idea's inherent coolness factor. The technical issues, however, did not describe the full extent of our problems with this feature. Quite quickly we also came to realize that there were plenty of game design issues implied by the space-to-planet concept. For example, there was the constant issue of player craft speed. We felt pretty sure that our ships should have a top speed of about 450 miles per hour, because dog fighting and bombing ground targets becomes extremely difficult if you move much faster. However, at that speed it would take the player 20 minutes to achieve a low-planet orbit. To circumnavigate a small planet the size of the moon could take as long as 16 hours. Although we were able to brainstorm several fanciful solutions to this problem, most were time- or cost-prohibitive, and all of our solutions threatened to shatter the illusion that you were in a small fighter craft, engaged in small, intimate battles.

Back to Earth

Star Wars Starfighter finally shipped in February 2001. While it was a little bit later than we had initially hoped, we burned our first set of master disks in mid-January, within three days of the "realistic" schedule projection that Daron had made a year earlier. While it certainly has its flaws, *Star Wars Starfighter* represents the culmination of an effort that involved almost 50 people, and it is a product that we are all very proud of. The lessons leaned over the last few years, both positive and negative, are already starting to be used by other Lucas Arts teams, ensuring that the project's legacy will be with us long after the last copy of the game has been sold.

Game Data

Project Europa—Lucas Arts' *Star Wars Starfighter*

Publisher: Lucas Arts

Number of full-time developers: Approximately 40 at the height of production

Length of development: 30 months

Release date: February 2001

Platform: Playstation 2

Hardware Used: 700MHz Pentium III's with 256MB RAM, GeForce 256, PS2 tools

Software Used: Windows 2000, Microsoft Visual C++, Metrowerks for PS2, 3D Studio Max, Softimage, Photoshop, Bryce, Visual SourceSafe, Perl, AfterEffects, Premiere

Notable Technologies: Eve level design tool, Miles Sound System, ObjectSpace STL,Macromedia/Secret Level Flash, Planet Blue's Tulip for prerendered cutscene lip-synching

Lines of code: 301,000 including tools