Postmortem: Ritual Entertainment's *Sin*

By Scott Alden

SIN is an original first-person shooter based on the Quake II engine with enhancements. Our previous effort here at Ritual Entertainment had been the highly acclaimed Quake Mission Pack #1 – Scourge of Armagon. SIN's story was developed during the completion of the Scourge of Armagon, and the main level design started in earnest immediately after the release of the mission pack.

SIN's focus was on its story and characters; we wanted to breathe new life into the first-person genre. Instead of being a mindless shooter that simply progresses from level to level, Sin has interactivity beyond that of many shooters. One of our primary design goals was to implement Action-Based Outcomes (ABOs), meaning that a player's actions on certain levels will have an effect in later levels. From the outset, we decided to license the Quake engine and get started right away, and later integrate the Quake II source code to gain the benefits that it brought.

We displayed the initial prototype of SIN at the 1997 E3 in Atlanta, Georgia. The demo showed off the Geothermal Plant level, some of the original weapons, and the original monster design. We proceeded to flesh out the game design by describing all of the locations that the SIN world would encompass. Sin's initial design featured over 40 levels, 31 of which made it into the final game.

SIN's interactivity and ABOs represented a big part of the development effort. We spent a total of about three weeks behind closed doors just brainstorming each level's interactive features — features that had essentially no effect on the game's final outcome. We would play through a single level in the morning, then after lunch we'd bombard the level designer with ideas for making the level in question more interactive. Once, while playing through the bank level over and over, someone came up with the idea of placing an ATM machine in the level where players could access certain character's accounts. This detail turned out to be one of the most notable things in the game. We also discussed the possibility of blowing up the dam level, but in the end we scrapped the idea due to the amount of extra work we would have to do to get the level to look right. We also spent about four hours hammering out how SIN was going to end. These meetings were no holds barred, and we had to be careful not to stomp all over other people's work.

SIN

We had completed most of the initial level design by December 1997. With the game's design fully realized, we began to add the major content. Then all hell broke loose — we got the Quake II source code. So, with the source in hand, we began to rewrite of the game. We didn't really intend to rewrite the game, it just sort of happened. The Quake II engine's new capabilities and features allowed us to add lots of new ideas. Our animation system with bones and nodes allowed us to attach any weapon or object to any character. This feature let us have many different grunts/soldiers by attaching different weapons to the same model. The surface system that we added let us have context-specific footsteps and ricochet sounds depending on the material types. We also added an interactive music system that changed moods during action segments of the game.

Ritual Entertainment
Dallas, Texas
http://www.ritual.com

Team size: The SIN team was made up of three programmers (Scott Alden, Mark Dochtermann, and Jim Dose), six level designers (Patrick Hook, Levelord, Tom Mustaine, Charlie Wiederhold, Matthias Worch, and Mike Wardwell), four artists (Beau Anderson, Robert Atkins, Michael Hadwin, and Joel Thomas), one project manager (Joe Selinske), one support person (Don Macaskill), one business person (Harry Miller), and one sound person (Zak Belica).
Release date: November 1998
Project budget: $2 million
Time in development: 20 months
Artist/Level Designer workstation: 200MHz Pentium Pro with 128MB RAM

Programmer workstation: 300MHz Pentium II with 128MB RAM
Critical applications: 3D Studio Max, Photoshop 3.0 and 4.0, and MSVC++ 5.0
Intended platform: Windows 95/98/NT

Finally, our scripting system let us add some internally developed features fairly late in the development cycle. A level designer might, for instance, need a special command to perform a particular type of interactive element in his level. Because it was so easy to add commands to the scripting language, the programmer would usually oblige. We added two to three commands to the scripting language daily. In the end, we had over 400 script commands total. So, while we did experience some feature creep, and though these last minute details did push our release date back even farther, we were able to add a lot of detail near the end of SIN's development cycle.

For about three months before the release, deathmatch tuning took place about two nights a week. We would play the most recently created level, and then the e-mail flood would begin. Change this, change that, and so on…. During the final weeks before the game's release, we'd spend about 12 hours a day working on single-player functionality, and about 3 hours a day on multiplayer. Once multiplayer functionality was to our liking, we froze it. We didn't change a single bit of multiplayer during the last couple of weeks. We did keep playing it to make sure that anything we did for the single-player portion didn't break any of the multiplayer stuff.

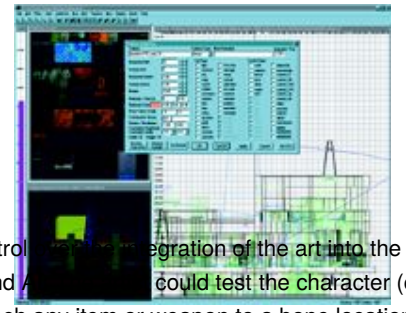
Things That Went Right

1. Power in numbers. When we started designing SIN, one of the first things we did was to form a tribal team approach. This is the way the company is run as well. There are two main camps of project management. In the classical Roman Empire approach, decisions are trickled down through a rigid pyramid-shaped line of designers, managers, and finally the implementers. Conversely, in the Attila the Hun approach, the entire group is given equal input and has equal weight in most decisions.

We opted for the tribal approach because it offers the greatest pool of ideas from which to dip and it adds a certain synergy to the whole of the game's design. Often a project can become myopic and narrow-minded in the hands of only one or two grand designers. We've noticed that the number three is magical. The greatest game designs have resulted from one person's initial idea somehow supplemented by a second person and then finally finessed by a third. It's that third derivative that usually ends as an ultimate idea. For example, the oil rig level originally started out as a cinematic boat ride up to the rig. Someone came up with the idea of letting players circle around the rig until they sniped off all the guards walking around. This version was pretty good, but players could ride around the rig 20 times before they managed to get all the guards. So we decided to have the boat wait on each side of the rig until the player had killed the two guards walking around that side.

2. Licensing the Quake engine. Licensing the Quake engine gave us a very stable base from which to begin. We were able to add new features and effects and then try them out pretty quickly. We also rewrote entire parts of the engine and heavily modified other parts. We wanted a higher level of interactivity than was available in the Quake game code, so we completely rewrote the game event system and AI code. We also built a character rendering system from scratch. The character system we used is a single-mesh hierarchical system. Finally, we added a bone-definition system so we could attach things such as guns or spears or spew out bubbles from any particular point. Even though we wrote a lot of the game code from scratch, the Quake II code was useful as an educational tool for game programming.

3. Animation system. The animation system was tied into our .DEF file system. The .DEF file is the extension we ended up using for all of our model text files. A .DEF file defines each character's animations and event triggers for specific frames of animation. The .DEF file is in plain text, so an artist can make updates to the file when he changes an animation. For example, if an artist changes the rocket launcher's firing animation, he can redefine in which frame the rocket would be fired in the game. Later, we discovered that we could develop a lot of special effects with the .DEF file system. We were able to create muzzle flashes, smoke, rocket trails, and various particle effects on the client without having to send over temporary entities with the networking system. We made the networking architecture so streamlined that firing a bullet only sent over one byte of information in the network packet.



SINED, SIN's level editor

[zoom]

4. Artist control. The artists had total control over the integration of the art into the game. Any artist had the capability to place a character in the game with a set of basic animations and A[illegible] could test the character (examine its skin, invoke its animation, and so on) within in the game very easily. We were able to attach any item or weapon to a bone location with one command. This flexibility allowed for a lot of tweaking to take place at the artist level. Artists tend to be much more critical of details, so giving them the ability to fix minor glitches without bugging a programmer was welcome. Furthermore, most of our artists were able to cross over into other disciplines, whether building models, skins, animations, or textures, or any number of other related tasks. This sharing of duties was important because of the large volume of art we had to complete.

5. Our scripting system increased level designers' control. We added a flexible scripting system so the level designers could create interactivity on their own. Previously, implementing most of the more interesting characteristics of levels was in the hands of the programmers. With our extensive scripting system in place, however, the programmers could focus on other areas rather than spending time writing specific pieces of game code for every level designer.

As I mentioned previously, the final version of our scripting language comprised over 400 commands. Level designers had intricate control over every aspect of level geometry, character animations, paths, and player interactions with the characters. Level designers could go far beyond the simple whizzing gizmos and script entire scenes of characters and machinery and gunfights. And, because we integrated the AI with the scripting system, the level designers were able to create a lot of specialized content. In the SIN world, bums will chat with you and give you clues, and civilians will cower in fear or run away.

SIN's scripting system was actually a full-blown multithreaded language — the level designers became programmers on their levels as well as architects. Besides, making these modifications was the most fun and rewarding aspect of level design. Level designers gained a lot of freedom, but only at the expense of time and effort on their part. To borrow terms from the movie industry, level designers have become the set designers, casting directors, directors, lighting engineers, gaffers… they have control over it all.

Things That Went Wrong

1. Young and naïve. Ritual is a new company and it went through a lot of growing pains during SIN's development. Many new faces came into the company, and many left. Actually, it is only now that the dust has settled that any real sense of a bona fide team can be seen. Our newly formed tribe felt little sense of cohesion, as most members were basically strangers to each other. A real team requires lot of faith and trust in order to act as a unit while performing something as creative as game design. This fact was especially evident in our decision-making processes. A clear vision was often muddled by too many inputs; settling on specifics often became impossible.

We've also identified an almost crippling realization that comes to all game developers during their first full game: We call it the Making Games Isn't a Totally Cool Job Syndrome. Crunch time sets in far too early, and a game's development cycle soon becomes a 24-hour-a-day, 7-day-a-week ordeal that lasts a year or more. Only hardened veterans know that the effort will prove worthwhile in the end, and that the end can be a long way off from the seemingly endless now.

2. Integration of the Quake 2 source code late in the project. Sin was originally scheduled for a Spring 1998 release, and we didn't get the Quake II source until late December 1997. We started the porting in early January, and it took a lot longer than we had anticipated. Because the Quake II source was drastically different from the original Quake engine that we started with, major systems were rewritten instead of ported. We rewrote the scripting system three times during the course of development. This was a major setback — finding issues with levels after they had been completed forced the level designers to rework each level, and in most cases, they just started over from scratch. We definitely learned our lesson in regard to trying to implement a major technology change midstream in a project.

We also had problems using Quake II's new tools (qbsp3, qvis3, and qrad3) with our maps. A lot of changes went into the tools before any of the old levels could be compiled and used in the game. We had to write a complete texture grabber utility because our texture and surface properties format was so different than Quake's. Major additions were made to Quake's level editor (QE4), which we then renamed SINED.

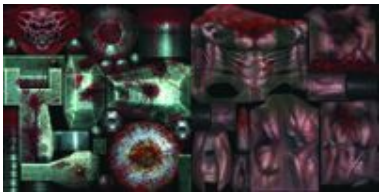
3. Overcomplicated systems. Developers at Ritual are very sensitive to the needs of the first-person shooter gaming community, and one the reasons that Quake has been such a success is that it is easily changed and modified by the end user. Nonprofessionals in this community made hundreds of modifications and total conversions, and we wanted SIN to be just as flexible.

Most of the major systems in SIN can be changed without rewriting any source code. Just make a simple change to a plain text file, and voilà, you have new behavior and effects. However, writing generalized source code is a lot harder than writing special case functions just to perform one task. Accommodating our goal of an easily modifiable game added a lot of extra development time to the new systems that we'd created.

An example of this is the console system, which we developed from scratch. This system had major ramifications in the game, and took about three to four man-months of programming time. We had not planned for consoles during SIN's design stage, and each level designer was responsible for the console content. It took a lot of planning and work to make a console, because the text messages were presented using a full layout language (like HTML). Something as seemingly simple as an ATM machine in a bank required tweaking over the course of several months. Moreover, the level designers already had a lot of information to absorb in creating a SIN level. Add 100 more console commands to the 400 script and AI commands, and you are just asking for trouble. Because consoles took so much work and effort, when crunch mode hit, consoles were underutilized and in some cases dropped out from the level and replaced with a push button.

4. Inconsistency and too many assets. We lacked a standardized method of texturing models from the outset. We started out using the Quake method of planar projecting the front and back of a character. Planar projection worked well, but about halfway thorough the project, it became apparent that this method wasted a lot of texture space and wasn't the right way to map, especially with less organic models. When we got per-face mapping and 3D Studio Max 2, we took more care to unwrap the models carefully and use the texture space better. Most of the original models were redone, but a few slipped through due to time constraints.

Modeling ThrallMaster in 3D Studio Max.
[zoom]



ThrallMaster's skin.
[zoom]

Because it was our first major project as a team, we spent quite a bit of time just seeing what worked and looked good in the engine. We created over 3,000 textures, but many simply didn't look good in the game or were too general; we only used about a third of the original texture library. We also built over 400 in-game models, including around 35 characters. This huge model library made for some very diverse and interesting environments, but the overwhelming number of models represented a lot of work; sometimes, the quality suffered because of the sheer quantity of content that had to get done.

5. Too many cool things isn't too cool. The more that's in a game, the more there is to go wrong. Although creating the gadgets and gizmos and character AI and environment interactions and such are some of the best aspects of level designing, level designing in particular has become far too complicated for any single individual to control. Level designing isn't simply making square rooms with connecting hallways anymore, and gone are the days of simply placing characters on a map and relying on their inherent AI to control them. The demand for realism has requires almost unattainable detailing to successful levels. Each and every character must be placed with paths and directions to escape points and attacking advantages, and most all their behavior must be scripted. Environmental interaction (entities) is also growing exponentially as level designers now have control over just about everything in their levels. A normal level can now take more than two months to complete, and play testing all of these added enhancements can be a nightmare. Lighting and sound have also become complicated and burdening. Level designing will soon become a multidisciplined department due to these increasing demands. We actually tried this organizational approach, but being a totally new concept, its actual implementation may have hindered more than helped SIN.

Return to the full version of this article