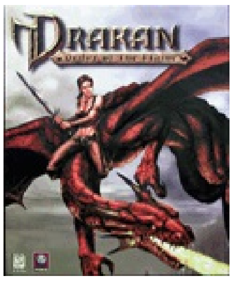
Postmortem: Surreal Software's *Drakan: Order of the Flame*

By Stuart Denman

The merging of great concepts from many different sources in order to create a new, better whole is perhaps one of the most fundamental aspects of human innovation. *Drakan: Order of the Flame* uses this notion to full advantage by combining action and adventure game concepts with sword combat, aerial battles, and simple RPG elements. It is a true hybrid of many proven gaming concepts. But this attribute made *Drakan*'s development doubly challenging because we had to create a game in which multiple elements worked well independently yet blended together seamlessly. Perhaps this is analogous to the way developers must work well as individuals and effectively as a team.

Origins of the Team

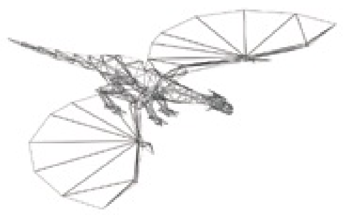
Because *Drakan* was Surreal's first product, the story of*Drakan*'s development is also the story of Surreal's development as a company. Surreal's creation is the classic game development story in which four ambitious recent college graduates decided they had nothing to lose and formed a game company. These four founders contributed four critical skills to the team: art, programming, design, and business skills. None of us had ever run a company or managed schedules, but we all loved games, and we knew what it took to make a good one.



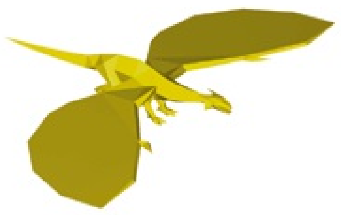
Lead designer Alan Patmore had always played games and had the business savvy to complement Nick Radovich's business experience and connections. I had been programming games and graphics since the age of ten, so even though I didn't have experience working at a game development company, I did have the skills and motivation. Mike Nichols, our creative director, came from within the industry and was the only member with any titles under his belt.

Our initial goal was to develop several game concepts and a solid technological foundation that we could pitch to game publishers. This would get us the funding we needed to pay ourselves and start hiring programmers and artists without having to involve venture capitalists.

Once we got project funding, we were able to quickly build a strong team of artists, programmers, and designers who all played games. Some of the team came from other game companies — lured by the informal atmosphere and the focus on games, not profit. Others were inexperienced with game development, but had the skills and fresh ideas we needed.



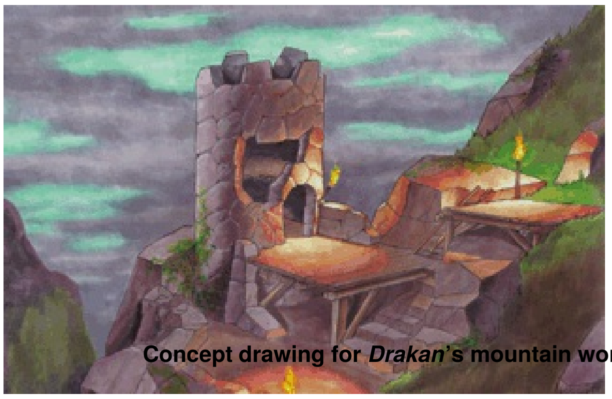
As the technology lead, I was determined to build Surreal's foundations on its technology. By retaining rights to our engine and tools, we always had something to fall back on if a game design was cancelled by the publisher. This also allowed us to develop multiple game titles from one generic technology and license the technology to other companies. Any investment in time that the programmers and I put into the engine could be quickly put to use on another project if anything went awry.



We initially moved away from the popular *Doom*-type engines toward a landscape-style rendering engine in order to set our games apart. There were many unique ideas that we could build from this: flying, underwater environments, outdoor deathmatch, and so on. But the technology was not only about rendering; the tools had to empower the designers and be general enough to support almost any game. So I designed a toolset in which every game-specific property and behavior would be provided by the game code itself, and the editor would be just a generic interface to the underlying game specifics.

Origins of the Beast

After pitching several game ideas to all the major publishers, we finally sold the first "dragon" concept to Virgin Interactive Entertainment (VIE) in the summer of 1996. The concept was very different from today's Drakan. The first concept was for a dragon RTS game in which the player's dragon could fly around taking over villages and forcing them to do their bidding. VIE wanted a more arcade-style shooter game to fill a slot in their product line, so we started developing a fast-paced, third-person dragon-flying game.



Concept drawing for *Drakan*'s mountain world.

From top to bottom: Arokh's polygon mesh, alpha-blended wings, and the final version.

It was not until early 1997 (when VIE began cutting projects just prior to closing its doors) that Surreal sold the *Drakan* concept to Psygnosis. Psygnosis saw the strength in our team and gave us complete freedom to perfect the design. We wanted more of an RPG feel, but as a dragon, the player was limited in what he or she could carry or interact with. Adding a human rider was the best solution, and a female character was the natural choice since she would be the ideal personality to offset the dragon's immense size and power. With an increased budget under Psygnosis, we hired more team members and increased the art and game-play content to a level that the press called "ambitious" at our public debut at E3 in 1998.

The production under Psygnosis allowed us to expand the technology as well. We added real-time lighting effects and expanded the simple height-field landscape engine into our seamless indoor/outdoor layer technology. Critical to this technology was Psygnosis's willingness to drop support for software rendering (a risky marketing decision at the time). This allowed us unprecedented freedom. We switched over to true-color textures, increased the polygon counts throughout the game, and built arbitrary geometry for our worlds. The downside to relying on 3D hardware was that we faced serious compatibility challenges — the game would have to run on almost every 3D card. This also meant battling Direct3D driver bugs, and the possibility that we would be inundated with technical support calls, since people would not have software rendering to fall back on if the 3D hardware failed to work correctly.

What Went Right

1. Success with graphics

There's no doubt *Drakan* had an ambitious design, so the graphics had to be top-notch in order to make the game world believable. The amount of art and animation content we would need mandated careful planning, lest our schedule slip. The solution to the problem was what I call "flexible reuse." In addition to the sharing of texture and geometry data between objects, *Drakan*'s engine (code-named the Riot engine) was programmed to allow arbitrary scaling and rotation of art content. By assigning different behaviors and combining multiple art components, we were also able to create totally new structures with minimal effort.

Because we dropped 3D software rendering, we knew all of our textures could be created in true color. This vastly improved the look of *Drakan*, so much so that we decided to switch from using palette-based textures to true-color textures, which required quite a bit of reworking on most of the textures in the first few levels. This decision is just one example of Surreal's aesthetic fussiness. Often if a few people thought that something within the game didn't look good enough, it would end up getting redone until everyone was satisfied. The benefits can be seen in the final product, but our schedule sometimes suffered as a result.

Though the artists created the objects and buildings in the game, the designers were responsible for placing the objects into the game and gave immediate aesthetic and game-play feedback to the artists. They also were responsible for building the landscapes and caves, which defined the overall level flow. This process evened out the workload between artists and designers, but it required the designers to have a good artistic sense. This can be seen in the very fantastical landscape architectures that the designers constructed and then painted with tileable textures. The textures were drawn by the artists to have many variations and transitions, which added to the organic nature of the terrain.

2. A green team with fresh ideas

Drakan had an advantage that many large game development companies sometimes overlook. It had a young team, highly motivated, bursting with ideas, and ready to take risks. The ideas were unique and motivated by the desire to set *Drakan* apart from the shooters and *Tomb Raider* clones (although this was still difficult given the tendency of the gaming press to compare games to one another).

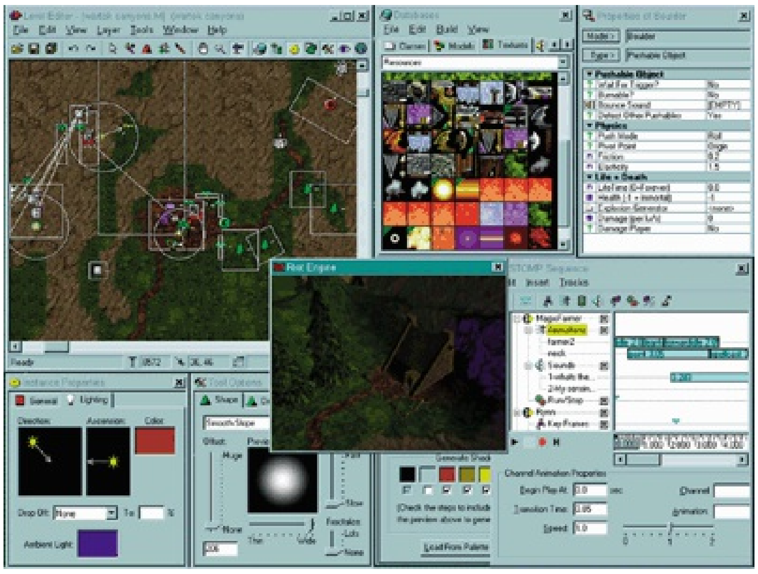
The most original idea in *Drakan* was the combination of dragon flight with sword and bow combat on the ground. This fundamental idea formed a developer's carnival for more innovative ideas and forced the player to strategize in a way not often seen in action games. The relative vulnerability of the female rider contrasted with the powerful dragon required careful thinking by the designers. Levels were created with restrictions on the dragon's ability to go places. Rynn could enter caves, but would come across areas where the dragon's flying abilities or strength would be necessary to proceed. The player (as Rynn) would then have to find a large door or other method to get the dragon inside the cave system. In this world of magic, creative ideas for special effects are very important, and these tasks were ideally suited to people who were not afraid to do things "outside the box."

Colored conceptual sketch of a Wartok grunt.

3. Engine and tools

Many recent games have shipped with engines that simply cannot handle the target platforms and the breadth of 3D hardware that they claim to support. I believe this is primarily due to a lack of planning and preparation for current and future technologies (not having scalability, for example), and a rush to focus on the game's design and art. No time is given to solidify the underlying technology, which should ideally be done before the designers and artists even start constructing content. If the designers spend half their time waiting for the game to load, or dealing with unplayable frame rates, the final game will only be half as good as it could have been.

Regarding the game's code, if ever there was an example to put the C vs. C++ argument to rest, it is *Drakan*. There simply are no performance reasons not to go with C++, as long as the programmers understand what is happening under the covers. Object-oriented code generates so many benefits, especially for an engine that you plan to build on for many years to come. In *Drakan*, the game-specific source code and engine source code were separated into different projects, so no game-specific code was allowed in the engine. The game-specific code included such features as the user interface, AI, and game entities, and contained no platform-specific code.



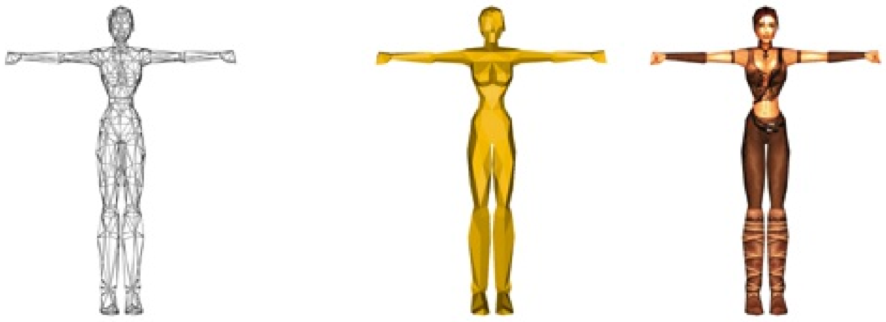
Surreal's in-house level editing tool showing the real-time 3D editing window in the center and top-down layout view in the upper-left.

The engine is broken up into many classes that handle various engine tasks and are the interfaces by which the game code accesses the engine. For instance, there is a sound class for playing sounds, a texture class for working with textures, a sequencing class for playback of scripted cutscenes, and numerous others. These also form a framework for future porting of the system-specific functions to other platforms. The stability of this system can be validated; we are currently creating additional games based on the *Drakan* engine with little or no code changes to the engine project. To further reduce the debugging time, we put coding guidelines in place to ensure the consistency of code between programmers, and created classes to catch array boundary violations and memory allocation problems.

Drakan has no scripting language. Instead, the programmers create modules that are visually connected by the designers to create scripted events in the game. Such modules include triggers, switches, timers, counters, and more complex modules such as doors, enemy creatures, and weapons. The modules have programmer-defined parameters associated with them. A parameter can be almost anything: a number, a list of options, a sound, a texture, another module, and so forth.

The system meant designers could tweak parameters and combine modules in ways that the programmers never intended. One particularly nice example was an effect that was originally created for the "ice sword." The effect was made up of a number of particles (originally snowflakes) that would collect for a certain amount of time on the mesh of an affected object. After a time, the particles would fall to the ground and stick for a bit. All these properties, from the timings to the particle texture, are configurable. With this feature at their disposal, the designers created glowing auras around ghosts by increasing the particle size and making the stick-time infinite. They created snow that landed on invisible platforms to guide players across them. The snow effect was attached to arrows to drop ice behind them as they flew. All this from a small bit of programming.

The engine also has an efficient caching system, so it's able to handle hundreds of megabytes of data on our minimum system requirement of 32MB RAM. The two main characters, Rynn and Arokh, total more than 20MB of animations, plus 12MB of sounds (including in-game cutscenes). To pull this off, the system keeps the most recently used sounds or animations in the cache and can flush memory that it hasn't used in a long time. Further reduction of memory usage is achieved by sharing animations between characters with the same skeleton, even if they have completely different skins. The system only loads the data that it needs, as it needs it. This is important during development, as artists and designers are prone to leave unused textures, sounds, and models in a database. The result is good engine performance during development, which is also representative of the final product.


Rynn’s highest polygon count was only 538. Single-skinned characters such as Rynn generally look much better with fewer polygons compared to the segmented characters traditionally used in most game animation systems.

We tried to ensure that the engine and tools always displayed to the artists and designers something that was representative of the final game (WYSIWYG). The best example of this was our real-time 3D editing system. The engine was integrated into the editor, so any geometry, texture mapping, or lighting changes made by the designer would be immediately reflected in the 3D view. The importance of this aspect of the tools should be emphasized because it gave the designers the ability to tune levels and game play very quickly and with a minimum of guesswork.

4. Compelling design

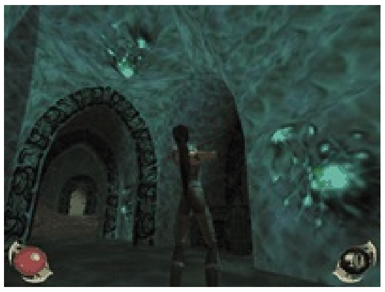
A good design will not only sell a game — it can also help smooth the development process. The *Drakan* world has immense possibilities, so new ideas were born easily within its scope. This kept the team highly motivated, as there were always innovative things to do with the genre. The varied environments gave a wealth of new things to work on for the art and design team, and were an ideal canvas for programmer invention.

The design also kept Psygnosis very interested. *Drakan* became its top PC product, and it was comforting to us as developers to know that our publisher was behind the product. Psygnosis saw the marketing potential in a beautiful female character combined with a fearsome, fire-breathing dragon and the press latched on to the concept with excitement. They could market it to *Tomb Raider* fans, AD&D fanatics, and even 3D shooter addicts.

Even the most brilliant design would be difficult to implement lacking a proper design document. The 175-page Drakan design document contained outlines for the entire game, including all AI behaviors, weapons, and level flows. It served its initial purpose well, and was a blueprint for our lead designer’s vision. The document was vital to the development team, especially when it came to scheduling, creating tasks, and communicating with the publisher. But as you will read in the following “What Went Wrong” section, feature creep overtook the project halfway through, and the document never kept up with the changes. A design document should always be maintained throughout development to preserve it as a useful resource for the team. Fortunately, the team could always rely on Alan to explain anything or to fill in any holes in the design document.

Concept sketch of the Dark Union sailing ship.

5. Indoor/outdoor environments


Morphing walls were created by assigning special behaviors to the world geometry.

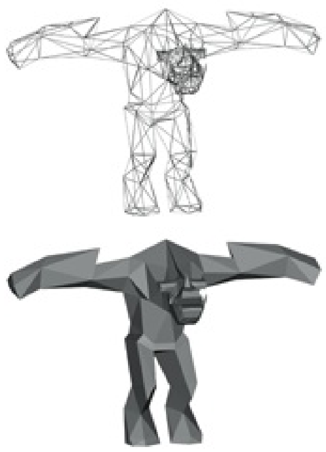
One of the major technologies that set the Riot engine apart from the other landscape engines was its ability to render both indoor and outdoor environments using the same engine. The benefits to game play were huge because we could do arbitrary cave systems, arches, overhangs, and other structures that were perfect for a fantasy game. The “layer” system that the landscape was created with was ideal for massive outdoor environments and allowed the designers to create very organic-looking worlds. Rectilinear structures such as buildings and objects were created using arbitrary models imported from external 3D modeling programs. Although these models were not included in the visibility calculations, the layers were included and were nicely suited for use in the visibility culling of large environments. Because the layers were small height fields that made up the ceilings and floors of the surroundings, they took up very little space in memory. This meant that the levels could be vast, and it helped give the player a sense that there was a living world around them.

What Went Wrong

1. Staying on schedule

Drakan was originally slated for release in February 1999, but ended up being released six months later. Even with careful scheduling and task planning, we failed to meet the final deadlines. Part of the problem was that we didn't account for the time the team would spend creating versions of the game for E3 and for magazine and Internet demos. Each demo pulled nearly two weeks of time away from our normally scheduled tasks. The majority of the scheduling problems were due to feature creep and other improvements that were considered necessary during development.

In March 1998, the design team was faced with a mountain of work ahead in order to complete the 14 original levels as designed. After careful consideration, the designers decided to spend their efforts on enlarging and improving upon the ten best level designs. They also ended up cutting many features that did not show much game-play promise. The dart gun and the boomerang weapons were among those eliminated from the game. Even though these tasks had already been mostly completed (in terms of code) for several months, they had not yet been put into the game. By the end of the project, the designers did not have adequate time left to work with programmers to play-balance those features, and the art staff had not done any work on them either. So they got cut. The decision allowed us to focus on improving the weapons that worked well, such as the bows and arrows. We now know that it's critical that programming tasks get put into the game and tested almost immediately so that their effectiveness can be realized early on. This lack of coordination between designers, artists, and programmers often caused problems during development. Some of this was because our design document wasn't updated when weapons, levels, or AI were redesigned.



The initial AI programming followed the original design document, but didn't work well when put into the game. It wasn't until our designers worked with the AI programmers to figure out exactly what they wanted for our combat system that the AI really came together. This kind of collaboration should have occurred at the beginning of the AI programming process and the lack of it caused moderate delays.

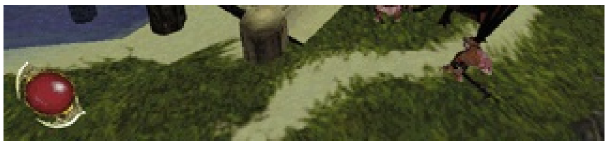
Drakan's art team often rebuilt geometry and model textures, sometimes up to three times before they were satisfactory. This may have been partially due to Surreal's high aesthetic standards, but a lack of consistent artistic vision is also to blame. *Drakan* had lots of character conceptual art, but no "art bible" to document all the models and environments for the game. This meant that if our art lead was not satisfied with work of another artist, he would often rebuild it himself. At various points during development his time was spread thin across many different tasks. In addition, the art team went through communication problems and power struggles that hampered the coordination of the team.

2. Inadequate testing

Although we tracked bugs internally before and during the alpha and beta releases, Psygnosis was responsible for the bulk of the testing after alpha. For a game as vast and ambitious as *Drakan*, the time that we allocated for testing was inadequate. Multiplayer and collision detection issues, in particular, were not given enough testing time. When the final shipping date approached, we reluctantly agreed to allow some noncritical bugs to slip through to the gold master in the interest of meeting the deadline. A patch was inevitable.

The war giant towered above Rynn and had multiple high-resolution texture maps.

Other testing complications added to the problems. As Psygnosis was being reorganized by its parent company, Sony Computer Entertainment Europe (SCEE), half the testing department was let go and merged with SCEE's U.K. testing group. This caused minor hiccups in tester allocations to *Drakan*. Since the testing team was located in Europe, communication was difficult, and often messages were delayed by a day or two. Bug reports were sent to us via Microsoft Excel worksheets, which were converted from an Oracle database that sat isolated on their LAN in the U.K. Often the Excel worksheets would come to us corrupted or would have incomplete bug descriptions. Bug responses from Surreal's programmers had to be tracked carefully and entered back into the Oracle database by hand.

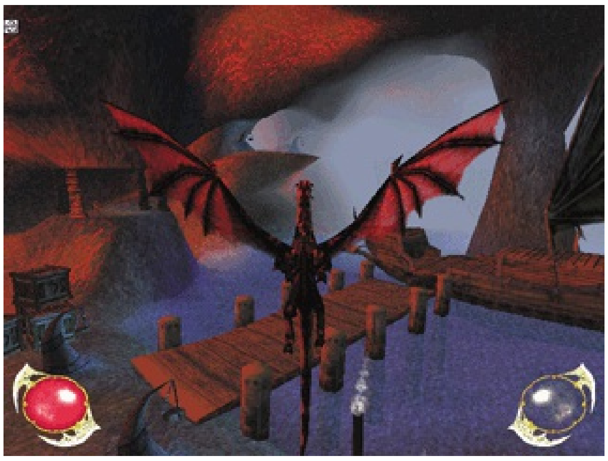


The island level showing the organic qualities of the terrain and textures.

We kept our own internal database at Surreal using Outlook forms in special public folders on our Exchange Server. We ended up generating almost 1,000 internal design, art, and programming bugs during the entire project. This rivaled the number of bugs generated by the testing team during alpha and beta. The internal system worked very well, but it could have been more useful if the Psygnosis testers had access to the system as well. We tried getting on-site testers, and some of the U.K. team did come to Surreal for about a week. But it was too late in the project and for too short a time to be effective.

3. Collision detection and response

One of the biggest chores for the testing team was to make sure that all of our hundreds of 3D models could not be penetrated by missiles, NPCs, or Rynn. Each model had to be properly bounded by the artist during the model's construction, a process which took about 20 percent of their modeling time to construct. Bounding was generated in our custom modeling tool and approximated the polygons of the model using a hierarchy (tree) of bounding spheres or oriented bounding boxes (OBBs). This made the collision detection system very fast and accurate, but it also meant that if an artist made a mistake in the bounding tree, collision detection might not work. To say this created a testing challenge would be an understatement.



Lighting effects created varying moods. This bridge model was reused from the islands level.

Even though the engine was capable of rendering arbitrary meshes, the collision detection system was not designed to handle some of the detailed meshes that the artists produced. Some of our AI used the bounding information at the lowest level, while Rynn's collision response system used a polygon-accurate analysis, which didn't work perfectly for some complex models. Frame-rate variations across machines also caused differing results, making it hard for programmers to reproduce the bugs and correct the problems. Finally, our indoor/outdoor landscape system created some challenging collision-detection problems that we hadn't anticipated when it was originally designed.

4. Multiplayer

Considered by some the Achilles' heel of Drakan, its multiplayer suffered from developmental neglect. For the game's multiplayer to have succeeded, the design, art, and programming teams would have had to spend at least twice as much time on it than they did. The two multiplayer designers did most of their level and weapon work during the alpha and beta periods. The same designers also created most of the artwork for the multiplayer effects and weapons. Any game-related bugs that came up were fixed by our single network programmer, who already had his hands full optimizing the underlying network engine. Most of these game-related problems arose because the same weapons were used in both single and multiplayer games, but the original programmers were not careful to make them "network aware."

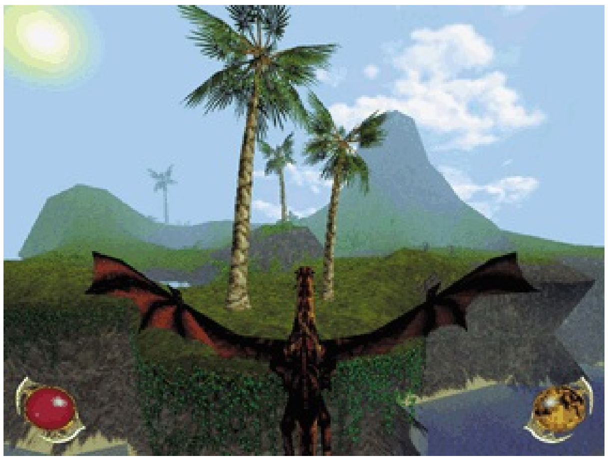
A multiplayer ground deathmatch level.

Originally, we thought that DirectPlay was the easiest networking solution for us. But as the design got more complex, we found that DirectPlay just did not work well for us. DirectPlay was a debugging nightmare. The network programmer's machine crashed several times per day when debugging the networking code and we couldn't determine what was causing that to happen. It wasn't until we switched to Winsock that we discovered that the crashes were caused by DirectPlay. DirectPlay also caused a serious problem for us while we debugged the game under the first release of Windows 98. DirectPlay actually caused the system clock to slow down. This caused the game to run slower and sucked up tons of CPU cycles, forcing a reboot. When put to the test, DirectPlay also had issues with firewalls, which we were not able to resolve. Under certain circumstances, the way DirectPlay handled the message queues sometimes caused messages to pile up until the application hung. Perhaps Microsoft will be addressing these issues in future releases.

It was clear even before alpha that the networking code would need to be rewritten. In the final design, we only used the TCP/IP portion of DirectPlay, and we used a Winsock front end to handle communication with the master server. We proposed to Psygnosis that they give us more time to convert the system over to Winsock, to which they replied yes — but only as a downloadable patch, since the additional work would have delayed the game's release. The Winsock conversion was not finished until a month after *Drakan*'s release, and greatly stabilized the multiplayer experience. This, combined with the release of the level editor and mods, has created a resurgence in multiplayer support, but it will never be as good as it could have been.

5. Badly executed story

Although the overall story concept of *Drakan* was a great, the script and execution of the idea were lacking. We hired a movie scriptwriter to do the initial work on the script, but he was not familiar with the fantasy genre and did not have a firm grasp on Alan's vision for the design. From there, the script was edited and rewritten by several more people: members of Surreal, members of Psygnosis, even one of the voice actors. Under pressure to finish the script, it was completed with cheesy one-liners and other badly written dialog. Once it had been recorded by the voice actors, it was very difficult to re-record lines that were badly written or acted. Some were re-recorded, but that was a luxury we could only afford for the completely failed lines. The voice acting was difficult to get right, because just as some of the writers had almost no vision of the game, the voice actors likewise had little understanding of the characters they portrayed.



Dragon's-eye view of a fantastical island formation.

Another problem with the execution of the story was that most of the construction of the cutscenes was left until the last minute, since all the levels had to be "geometry complete" before cutscenes could be created. This meant that the scenes at the end of the game were hastily done, and some even had to be cut from the game.

Onward to the Next Projects

Drakan's development was a bumpy ride, but it went suitably well considering it was the first game developed by an inexperienced team. Even with some of the schedule slips that occurred, the great design, art, and programming kept the project going strong. *Drakan* has been a great learning experience for the team, and the careful evaluation of our past mistakes has helped us in the development of our current projects. *Drakan* was recently named PC Game of the Year by several popular magazines, and it has sold very well. If *Drakan* showcases what this team is capable of in our first project, it will be very exciting to see what we are capable of in the future.

Stuart Denman was the lead programmer on *Drakan* and is a co-founder of Surreal Software. This was his first "real" job following four years as a student of computer engineering at the University of Washington. After scouting around Europe on a post-*Drakan* vacation, he is currently back at work developing Surreal's next 3D engine technology. He can be reached at stu (at) surreal (dot) com.

The Drakan team: Front row, from left to right: Satish Bhatti (network programmer), Tim Ebling (programmer), Todd Andersen (designer), Susan Jessup (artist), Louise Smith (artist/animator), Andre Maguire (designer), Mel Guymon (lead animator), Tom Vykruta (programmer).

Middle row: Shaun Leach (programmer), Armen Levonian (programmer), John Whitmore (designer), Greg Alt (programmer), Heron Prior (animator), Tom Byrne (artist).

Back row: Stuart Denman (lead programmer), Scott Cummings (animator), Boyd Post (sound engineer), Alan Patmore (lead designer), Hugh Jamieson (character artist), Mike Nichols (lead artist), Hans Piwenitzky (artist), John McWilliams (designer), Nick Radovich (business/sound).

Not pictured: Joe Olson (artist), Duncan (designer), Isaac Barry (designer), Ben Olson (artist).

Draken: Order of the Flame

Surreal Software Inc.
Seattle, Wash.
(206) 587-0505
http://www.surreal.com

Release date: August 1999

Intended platform: Windows 95/98

Project budget: $2.5 million

Project length: 28 months

Team size: 23 full-time developers, 2 sound and music contractors

Critical development hardware: Pentium II and AMD K6-2 (3DNow!), 200 to 450MHz, 128MB RAM with Nvidia Riva 128 and TNT, 3dfx Voodoo 2 3D hardware. Artist workstations: Wacom tablets.

Critical development software: Windows software, Programming software: Microsoft Visual C++ 5.0 and 6.0, Visual SourceSafe 5.0, Intel VTune 2.5, InstallShield International 5.0. Art and animation software: Softimage, 3D Studio Max, Adobe Photoshop, In-house modeling and texturing tools. Sound and music: Sonic Foundry Soundforge, Emagic Logic Audio

Return to the full version of this article