GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Treyarch's *Draconus*

By Jamie Fristrom

After finally completing *Die By the Sword* in 1998, the guys at Treyarch were pretty damn proud of themselves. Sure, it was late, and it wasn't a phenomenal seller, but the reviews were great and we learned a lot from our mistakes. We were dying to start over and write our dream game, a game that would combine the animation, sword-fighting, and creative level design of *Die By the Sword* with cooperative online play. We imagined a sort of 3D *Diablo*. Our three-day postmortem meeting for *Die By the Sword* turned into a design and planning meeting for *Die By the Sword II*, as we were calling it then. We signed a contract with Interplay and we were off. While the game designers were writing design documents, the programmers were busy with R&D into dynamic adaptive level of detail, asynchronous models of online play, and a new scripting language. All of the while we were simultaneously kicking out an add-on pack for *Die By the Sword* called *Limb From Limb.*

Screen shot from *Draconus*

The new game would have the same core management as *Die By the Sword*: Peter Akemann was lead programmer and Final Word on Everything, Chris Busse was producer, Chris Soares lead artist, and Mark Nau lead game designer. Three months later, we had finished the research, completed the design document, created an initial milestone schedule, and coded a Direct3d renderer and a nifty scripting language. But trouble was brewing; Interplay didn't want a game that wouldn't be done by next Christmas. They weren't excited about a PC game. The Dreamcast was the hot new topic and naturally they wanted us to do our game for that console. They were looking for someone to sell our contract to.

Suddenly, the game to which we had planned to devote two years needed to be ready for the Dreamcast launch. That meant September, giving us about ten months to final. To make matters worse, we had to learn a new platform and port the game to it. In addition, we had only a seven-man team and needed to staff up. We all knew it was doomed to failure and cuts would have to be made, so we eliminated network play. Some of us felt that cutting the network play wasn't enough. The trades claim that it takes about eighteen months to develop just a B+ title, while A+ titles like *Half-Life* and *Metal Gear Solid* might go into a third year. If we used that as a measuring stick, and were trying for just a B+, we would slip by about six months.

Still, the new accelerated schedule was approved. In my experience, publishers love overoptimistic schedules. They don't seem to care if a developer is smoking crack, even though it happens quite often that the publisher is the one who ends up funding the inevitable, costly slip. Interplay sold our Crave, but then something weird happened. They didn't sell the entire game to Crave, just the domestic publishing rights. They kept the international publishing rights for themselves. We ended up developing the game for two publishers: In America for Crave as *Draconus* (Interplay did not sell the *Die By the Sword* title to Crave) and internationally for Interplay as *Dragon's Blood*. We were living in interesting times.

All things considered, it's no surprise we didn't make it. We did, however, perform many small miracles. We managed to get our C++ code base ported to the Dreamcast Shinobi/Kamui libraries in time for E3, surprising Crave and creating a small sensation. The artists created all kinds of cool characters and levels and special effects. The programmers pushed various Dreamcast features further than they'd been pushed before. We built a new engine from scratch, using object-oriented techniques, design patterns, and generic programming: The programmers had all been exposed to C++ before they started work on Draconus.

In *The Mythical Man Month* Fred Brooks points out that you'll realized you're not going to make it three weeks before the scheduled end of a

project. Appropriately, three weeks from when we were to have our "launch title" done, Akemann and Busse realized we weren't going to make it. We rescheduled for a Christmas launch. Three weeks from that second due date we realized we weren't going to make it. So we rescheduled for a launch that would come out the day before Christmas. (Not too useful if you want to sell titles for Christmas. We were telling ourselves that all those people who received Dreamcasts for Christmas would want to go out and buy our game.) Three weeks from *that* due date we realized we weren't going to make it. During this period of always almost making it we were fully death marching. The bug list went over three thousand and took five months and about $400,000 more than we has budgeted to tackle it, but we suceeded. To quote Patrick Hughes, one of the coders, "Hey, it shipped." It was nine months past the date we had originally scheduled, but still, it shipped.

Tools

On the PC side, we used Visual C++ 5.0 for development. After switching to Dreamcast, we ended up settling on the Metrowerks compiler and the Codescape debugger. Although one of the promises of porting to the Dreamcast was that you'd be able to use Windows CE with DirectX and just recompile, it didn't work out that way for us. We found that it was easier to rewrite our renderer to use Kamui calls and our file access routines to use Shinobi calls than to try and fix the mysterious problems brought on by WindowsCE. On the bright side, Kamui and Shinobi are nice, thin, and straightforward libraries that were really quite easy to use.

The characters were modeled in 3d Studio MAX 2.5. We didn't use Character Studio. The artists divided the characters into segments manually and the exporter would reassemble them, joint by joint, into a single mesh and then reduce the detail on them. Some of the animations, like ladder climbing and certain cut-scenes, were done with these segmented characters in MAX and then exported for the game. Other animations (the flying dragon) were hard-coded by Pete using his VSIM technology, or were done by a game designer creating sequences of moves. Textures were painted in Photoshop.

For the sounds, we used the Digidesign Pro Tools system: Pro Tools 5.0 on the software side and a d24 system on the hardware side. We also used Bias' Peak for down sampling and editing.
All the characters voice over effects and processing was done in the Pro Tools environment. Various plug-ins were used to get the desired effect on certain creatures and characters. Most notably, Marto the Troll and all his troll friends had to be pitched down in order to make them sound like hulking brutes.

The music was edited and mastered using Pro Tools as well. This had to be done in order to get the music to "fit" in the Dreamcast. We had a lot of cool music and a limited amount of space to put it in. The sound effects Sergio Bustamante and Keith Arem created all were made in the Pro Tools environment. Most, if not all, of the sound effects are originals and were created using Pro Tools to mix sound effect-elements together.

---

1. Under funded and under scheduled

Our answer to being under funded and under scheduled was to cut network play. We needed to make much bigger cuts. If we had thrown out our new engine and gone back to the *Die By the Sword* engine, cut the number of levels in half, stuck with one main character instead of two, and tried to be conservative about our technical achievements, rather than pushing the Dreamcast to its published limits, we would have had a fighting chance. Unfortunately, we were still stuck on the idea of making our dream game. Also, we were sick of the crufty (Cruft is an accretion of kludges. Kludges are poorly thought out quick fixes and workarounds to get your code out the door.) *Die By the Sword* code base. All of the components were poorly designed and needed to be completely replaced: the front end's inability to display one font at a time was a limitation built deep into its architecture, the data structure for the rendered meshes was very bloated, the scripting language wasn't a proper language -it didn't even support the use of passed parameters to subroutines, and the net play was frame-locked. Furthermore, Michael Abrash claims that the John Carmack rule is "always start over," and John Carmack must be right, right? Well, Joel Spolsky has a different opinion, and his opinion is fairly compelling. (Read "Things You Should Never Do, Part I", http://joel.editthispage.com/stories/storyReader$47. . . if Joel had written this article two years ago instead of two months ago, we could have been saved . . .) We should have replaced components from *Die By the Sword* one at a time instead of throwing the whole thing out. Then, when the schedule got cut in half, we should have just lived with the crufty components that were the least annoying. (It's general opinion that the longer a code base sits around the cruftier it gets. This isn't necessarily true. For example, I've seen Neversoft's *Tony Hawk* code base as it progressed from *Tony Hawk* to *Tony Hawk 2* and the code actually got cleaner.)

The worst thing about under scheduling a project is that it's easy to trick yourself into thinking you can still make it. If you don't keep a running bug list of the problems people are reporting, you can lull yourself into the sense that all your features are 'nearly finished' even though they are rife with problems that will take countless days to solve. It's not until you're three weeks away from when you were supposed to be feature complete that you suddenly realize it's hopeless. In hindsight, even our two Christmases away schedule for our original *Die By the Sword II* vision wasn't reasonable, seeing as how we took almost that long anyway to do the stripped-down no-network-play Draconus. The sad truth is that the game you really want to make will probably require more time and money than any publisher in their right mind is going to want to give you. Most of us just aren't going to get to make a *Metal Gear Solid* or a *Half-Life*: we need to make the best game we can within the limitations placed upon us, and that means we need extreme pessimism.

2. Dual publisher situation

Fortunately, this is extremely rare. What seems to happen when two different companies are going to sell your game in two different places is that neither one of them wants to stand behind your product. They both point at the other guy. Need more development stations? "That's the other publisher's responsibility." Need more money? "That's the other publisher's responsibility." Need *anything*? You get the picture. Of

course, when it's time to get your milestone accepted, it goes through both publishers.

3. Poor planning and communication

Once you're under funded, everything else goes to hell. Our planning probably would have been better if we had more time. We had a fairly nice design document for *Die By the Sword 2*, but that was thrown out when we went to *Draconus*. Mark had a vision for *Draconus*, it was going to be something like what *Gauntlet Legends* turned out to be: a real bad-ass hero beset by dozens of foes. This vision was promptly ignored by an overly optimistic programmer (myself) who told the artists to go nuts and pump so many polys into the foes that we ended up being able only to have three of them on the screen at any one time.

Screen shot from *Draconus*

Another example of poor planning was that magic was an afterthought. We always knew we would have spells in the game, special abilities with nice special effects that your character would acquire as the game progressed, but they were left until late in development because there were always more pressing issues. Of course, once we did implement spells, they unbalanced the play of the game horrendously, and much more unplanned work was needed to make magic function reasonably. Aspects that are fundamental to playability need to be prototyped early, even if the pretty special effects aren't ready yet, so that the play balance can be measured and tested.

The game designers and artists didn't really communicate as well as they should have on the levels they made. A given level was supposed to be a collaboration between four people: an artist modeling it in MAX and a designer scripting it, with the whole process being overseen by the lead game designer (Nau) and the art director (Soares). As discussed in the bit on art, below, Nau provided a very rough sketch of how he wanted the level to play out to the level modeler. Rather than the scripter and the modeler working together, however, the modeler would just finish the level and pass it off to the scripter who would populate it with creatures at whim, not having Nau's original sketch. Then the level was often passed off to James Chao to add special effects to it, again with no communication as to what exactly the special effects were supposed to be. Nau also wrote some intro text to describe the goals for a given level, but his text was not forwarded to the scripters. Eventually the inconsistencies in level design were discovered and fixed as bugs, but it cost a lot more than doing the levels right in the first place. This absence of planning went unnoticed because nobody was actually testing the game; they were all too busy trying to finish their tasks.

4. Bad work flow

It took about five minutes to load a single level on a developer's station. Therefore, it took about five minutes to test the smallest change. We knew that would be unacceptable on the shipped version, but our solution for optimizing the load of the shipped version was to save a snapshot of the Dreamcast's heap, global variables, and texture memory, and have that be what the final version of the game loaded. This technology gave us *very fast loads*. One reviewer even commented on how fast our loads were. But the downside was crippling, implementing the technology took over a month of programmer time. Our developer station loads never got any faster, so working on the project was tedious at best. The process for making a release build was insane. Because we had two different characters that could wear five different costumes, we couldn't just load each level and then save the image file. We didn't get some of the costumes until later in the game, but we still had to save the images for about fifty different levels. To get it done, Akemann or Busse (or both) would stay at work all night, handling the process. Later, a bug cropped up where the image files would only load correctly if you went and hand edited them with a hex editor.

In other words, the act of just submitting a revision to QA took a whole day of someone's time. It sounds like premature optimization to make sure you're saving your game data, the meshes, levels, and what-have-you in the same binary form that it's going to be read into the game. You may have to fix up pointers and shrink memory allocations but that's fast so I think it's worth it. The mesh data for *Draconus* was a chunk-oriented text file. We had plans for how it could be converted to a binary file, but when we did the conversion it was still too slow to parse all those chunks. We may never get away from the need to just store our data in its final useful format, because even though our processors will get faster and be able to process the data faster, the data itself will get bigger and more complex. Side note: another advantage to text files is that you can edit them after you've exported them from MAX. When you think about it, this is actually a disadvantage, because it encourages people to make changes directly to the text file when they should be editing the MAX file.

5. Technical Failures

Progressive, adaptive level of detail was the hot topic at the GDC a couple of years ago. It sounded wonderful to me. I read all of Hugues Hoppe's papers on his website and developed a progressive mesh rendering system for use in Draconus. I was quite pleased with the results; watching the system remove the least useful vertices from a model was cool. Unfortunately, it wasn't much faster. Due to various kinds of overhead and the Dreamcast processor's small cache (although I'm making excuses here; what I really needed, as usual, was more time to optimize the thing), rendering the lowest-detail model still took about half the time as it took to render the highest-detail model. We would have been better off if we stuck to the old school of having the artists turn out half a dozen models at different levels of detail. (A lot of people don't like that system because of the way models pop from one state to the next, but that popping is much more visually pleasing than some of the artifacts you can get with progressive meshes. Again, with more time, I could have done better.) Furthermore, the renderer was too slow in general. Later on we figured out some tricks to get more juice out of the Dreamcast but it was too late to implement them in *Draconus*: it would have required a dramatic change to the renderer's data structures and we were already feature-locked. Much of our optimization took the form of making artists (mainly Alex Bortoluzzi) go back and simplify the levels, making the game designers go back and make sure not too many foes were visible at once, and -our most unpleasant hack- just turning characters invisible when they went beyond a certain range.

C++ served us extremely well on *Die By the Sword*, and we continued to use it extensively, including some of the more esoteric features such as STL. But with *Draconus*, C++ was quite the handicap. The first compilers available for the Dreamcast were the Hitachi C compiler and the WinCE Visual C++ compiler, which we assumed we'd be using. Unfortunately, the first version of WinCE for the Dreamcast was very poor. However, while we were struggling with it and looking for other solutions to port C++ code to a C compiler, two C++ compilers became available: Metrowerks and GNU. We couldn't get our Visual C++ STL usage to compile on GNU, so Metrowerks it was. We lost many weeks in our battle to find a compiler that could actually compile our code.

---

1. Quality people

When I wrote the post-mortem for *Die By the Sword* [ed. Note: ref GD Mag] I said that the number one thing that went right was the people, and I said that maybe once our team grew larger than eight we'd have to learn to work with people who weren't superstars. Fortunately that didn't happen, everyone on *Draconus* was an A person in that there were talented, intelligent, and productive. It's part of the Treyarch philosophy to only hire the best people. If you do that, you can probably survive anything. Leads are always involved in the hiring process, and we generally only hire people with experience unless they seem to have enough raw talent (shown by their sample work) to justify taking a risk on them. Potential artists models are scrutinized for economy of polygons and potential coders are given simple tests involving linked lists. The employees were loyal, and despite the continual death marching nobody quit. Maybe that's because the management was in the trenches with us. Nobody worked harder than Akemann on this project. It wasn't the situation of "Here, make this game for me while I go skiing." When the project was fully staffed there was one producer, five programmers, five game designers or 'scripters', one character modeler (he also did at least one of the levels), three or four level modelers, one full time texture painter, one full time special effect designer, one full time audio director, and one full time art director.

2. Maintaining a PC build

Due to the low budget, we only had six Dreamcast development stations to go around. Every programmer had one, as did the producer, while the artists and game designers had to go without. Obviously, we needed the artists and game designers to be able to test their changes as well. We accomplished that by keeping the PC build of the program up and running. Maintaining this alternative platform also had the advantage of enforcing higher code quality in certain areas: it stayed portable, and operating system differences were abstracted out. We always had C versions of our assembly routines, for example, and the interface layer between our game and the operating system was the same no matter which platform we were using. Although a PC version of Draconus will probably never happen, it is a simple matter to port games using the engine to the PC.

3. Art

The game art turned out beautifully. The artists think this is despite circumstances rather than because of them, but there are a few things we did right. The game has a consistent dark-fantasy look to the art style. Chris Soares, the art director, would discuss the atmosphere he wanted to see with the artists before they did any work. He pointed them to Frank Frazetta, Brom, and Phil Hale. For a given level, after Mark laid out a bare-minimum sketch of how he wanted the level to play (consisting of lines and circles. The art team has abandoned this technique since *Draconus*) Soares would give the level designers sketches and resource materials that he mostly culled from the internet and photo-journals. The level designers would use placeholder textures that just suggested where details should go, and then Chris Erdman, who textured every level, would do two drafts of the textures. ("Erdman's rough textures were as good as most games final textures," Soares said.) Communication was key. Splitting a job that is ordinarily handled by one person between two required that Chris Erdman work especially closely with the level artists, which he did.

Screen shot from *Draconus*

Artists were held responsible for the framerate of their levels. As they were modeling, they would export the level and try it out in the game. Unfortunately, because of the dearth of Dreamcast dev kits, they could only run the levels in the Windows version. We picked an arbitrary framerate that the levels were expected to meet on the Windows build. The game would print statistics about how much time it was taking to render the level and how many polygons were being rendered at any one time. (There were times when the statistics code broke and nobody fixed it: Soares pointed out that bad statistics are worse than no statistics at all). Once I failed to optimize the game as much as I had hoped, we made the artists go back and lower the detail of the levels and characters. This was a huge time waste but it did guarantee that we were pushing our code to the limit. (And past the limit in many areas, resulting in a drop in frame rate or increase in fog that we got nailed for in the reviews.)

The characters were sketched by Soares, then modeled by Arnold Agraviador, and then textured by Soares, Agraviador, and Erdman. A new character was finished about every two weeks, and we used placeholder models in-game while we waited for the actual character models to show up. Finally, James Chao was the sole artist in charge of special effects. This high division of labor is what helped us produce as much quality art as we did in the time we had available.

A lot of the reviews of *Die By The Sword* said it was too short, so we wanted *Draconus* to be much bigger. Here, we succeeded: it took me about six hours to play through *DBTS* when it was done, and it took me fifteen to get through *Draconus*.

4. Technical Successes

The scripting language for *Draconus* was much more powerful than one used in *Die By the Sword*. Charles Tolman had a B. S. in computer science from UC Santa Barbara, where he took a two-semester class on compiler design. In under a month he had written a full-on, object oriented, C++ style interpreted scripting language. The game designers wrote in this language, which was then parsed into a token stream. The game had a virtual machine that ran the token stream. The language could be used to manipulate any of the entities in the game. Since all of the entities, be they doors, goblins, or particle systems, are derived from a single entity base class, the same set of functions could be used to manipulate any of them. For example, when Mark needed code to manipulate the camera for a cut-scene, the code was already available: the same code he could use to, say, slide a door shut could be used to move the camera.

The scripting language turned out to be doubly useful when the artist who was designing the magical effects found out about it. The game had particle systems that worked a lot like the particle systems in MAX in that you could create a particle sprayer that had characteristics such as density, direction, and aperture while the particles themselves could be sprites or meshes. James Chao did some pretty neat things with these particle generators, but when he used the scripting language to start moving and orbiting several particle generators for one big special effect it was *fantastic*.

We didn't just create a scripting language for the game, there was another for the front end shell. We had an HTML-like language to allow the artists and producers maximum control over the front end screens. Although this language turned out to be too complicated for the artists to use, our producer could do most of the work, and we didn't have to waste programmer time when the layout or text of the front-end screens needed to change.

We also had success with multi-legged creatures and the dragon. Since most of the animation in the game is done by VSIM - to that point a biped-only technology -- Pete had to expand it to handle the movement of four legged beasts and the flying dragon that you encounter at the end of the game. Unfortunately, the quadruped motion was never deemed lifelike enough (although it looked fine to me) to animate the wolf characters, which were removed from the game. (The hell-hounds and insects were fine, though, since nobody really knows how hell-hounds and alien insects are supposed to move.) Pete also spent weeks algorithmically animating the final boss, the dragon X'Calith, so it could catch you in its mighty jaws, beat its mighty wings, and drop you from a mighty height.

5. Game Design

Draconus' design is another thing that turned out well despite circumstances rather than because of them. Mark's original Draconus plan for one-bad-ass-against-many is a recipe for a surefire hit, but when the technical limitations hit us, we had to come up with a new concept of fun. Through a lot of trial and error, we came up with somethings that worked. Although *Die By The Sword* drew a cult following for its highly interactive sword-fighting system, the complexity alienated the mass market. With *Draconus* we wanted a more fighting-like game, where button mashing would get you through the first few levels but you'd eventually need to learn the combos and blocks. For the longest time, you could just hold down the block button and be invulnerable. I'd be able to hold a conversation with Busse in the midst of a fight just by holding down the button. This problem was solved by making your back vulnerable when you are blocking, and by making you lower your guard after someone hit you. Suddenly you had to pay attention in a fight. Season this with magic spells and you have a fun game.

The game play features in the levels were created by game designers using the scripting language to add characters, obstacles, power ups, and in-game cut scenes to the levels. The power of the scripting language allowed them to do some interesting things; boat rides, moving level geometry, water levels lowering, specialized AI, and even bosses that summon other creatures to protect them as you fight. We needed more game designers than originally anticipated, largely because scripting was a painstaking process that we needed to improve. Again, great people accomplished great things despite obstacles.

There is a good deal of dialog in *Draconus*, almost all of which was written by Mark Nau, the game designer. After we counted up the pages, we discovered there was as much dialog as in several Hollywood screenplays. The dialog is campy and tongue-in-cheek, and a lot of it is surprisingly funny. After the voice acting was done you'd frequently find people quoting the game to each other at work: "We fight now? Yeah, we fight now," or, "Prove nothing. Of course club more good on human." The dialog gives the game characters an Army of Darkness -like incongruity between the dark fantasy artwork and the sarcastic one-liners.

Overall

It wasn't pleasant working on *Draconus*, and we didn't end up with the game we had envisioned at the beginning. We must have done something right, though, because in just two years we built a game with a brand new engine, created fifteen beautiful levels that you could play through as two qualitatively different characters, and included some cool bosses. (My favorite is the lich queen, although the dragon is certainly awesome.) And now we have an engine that is well written enough to be used again, this time for Max Steel. (Sure, large swathes of it are being replaced, such as the animation system, but large swathes are also remaining just as they are.)

Hindsight is always twenty-twenty, but it seems like a few software development fundamentals could have gotten us to a game of which we would have been just as proud, only a lot more quickly. If we'd made deep feature cuts in the very beginning, if we hadn't tried to model our levels and characters to the promised limits of the Dreamcast, and if we'd had one or two testers or production assistants through the life of the project making sure that the game was balanced, consistent and bug-free, I think we would have ended up with a smaller game of higher quality that was done on time with a minimum of death marching. Of course, I have no way to prove that. One of these days I'd like to read the postmortem for a successful original title that shipped on time. This is not that postmortem. We can keep learning from mistakes, ours and others', and still not learn the things we need to know to succeed, to actually schedule a long-term project successfully and ship it when we say we will.

Jamie has been programming games in Los Angeles and San Diego for nine years. He still makes a lot of mistakes. Check out his website at http://jfristrom.home.mindspring.com/ or e-mail him at jfristrom@iname.com.

Return to the full version of this article