GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Raven Software's *Heretic II*

By Jake Simpson

If Will Shakespeare were around today, I'd like to think he might say "If graphic violence be the food of gaming, then code on." Looking back on *Heretic II*, I think we can safely say that we developers at Raven Software followed that credo to the letter.

We designed *Heretic II* as a third-person action game from the word go. It had actually been proposed a while back here at Raven, but at the time, the technology to do it convincingly just wasn't there. So we shelved the idea until it could be done right. After seeing what Core had done with *Tomb Raider*, we decided to see what we could do with that original idea and id's *Quake II* engine. We'd always intended for it to be an action game — we wanted cool worlds, cool monsters, and exciting ways to hand out death.

Getting it Off the Ground

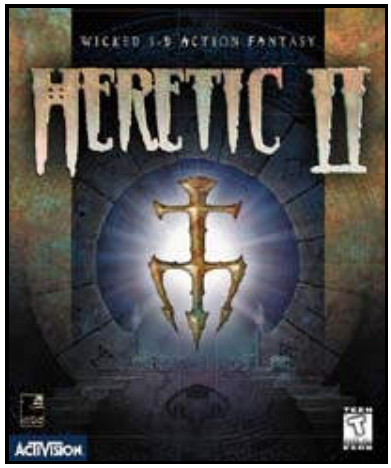
After the company decided to go ahead with the project, we quickly commissioned the fantasy artist Brom to create some conceptual art. This art helped shape the direction and flavor of the worlds and the creatures that would inhabit them. Next, the company commissioned a technology demo to ascertain whether or not it was possible to use the *Quake II* engine to build a third-person game with an intuitive camera. The camera was actually one of the easiest things to build. This all took about a month.

Once we showed our demo to Activision, the publisher gave us the green light, and we hammered out a delivery date so that we could be done by Christmas. Brian Pelletier took over as project lead, and the team drew up the project design document. Pat Lipo decided on the critical systems we would need to address, and development commenced. We had many discussions about the wisdom of attempting to implement a software renderer. At this point, the marketing department at Activision got involved and informed us that a software renderer was a requirement because Europe is traditionally about two years behind the States in hardware, so most Europeans would not have 3D acceleration. Traditionally, fantasy games have a strong following in Europe, so building a game to meet European considerations was important for us.

Development, a Thumbnail Sketch

Development progressed over the year in spurts — as development does. We recorded voice actors to portray Corvus and the tome. We commissioned a Russian group known as Creat to do our pre-rendered sequences. We hemmed and hawed over a pre-release demo until we finally decided to do one (ultimately, we were a couple of days late with it, mainly due to a malfunctioning load/save option). E3 came and went, along with the display of our demo (which brought a lot of surprised and enthusiastic reactions). After that, we hit crunch time hard to get the game done. We did have some worries about the in-game cinematics, but Rick Johnson came to our rescue with a scripting system he was developing for the fledgling *Soldier Of Fortune*. When it came to quality assurance, QA at Activision was especially tough on us because they'd just released a product that wasn't as bug free as it might have been. But this was in our favor because it forced us to tighten up and examine our code more thoroughly than we might have done otherwise.

Notes on Modeling



Our models aren't your run-of the-mill *Quake 2* .MD2s. We modified the whole modeling system so that we could have models with rudimentary backbones. A backbone was necessary to allow Corvus to look around when the camera was moved. It's a nice bit of realism, and added a lot to the feeling of character immersion. The new format also supported code reference points on the models so that we could create special effects at a specific point on the model, such as Corvus's hands or feet. The new model type was called a Flex Model, and was a product of Softimage. Softimage was used for almost all our animations, which were not — contrary to popular belief — the product of motion capture. We used 3D Studio Max to create all of our static world objects, as well as the simpler animations. The implementation of the divided animation system on the Flex models also worked really well. This allowed us to split off animations at Corvus's waist, so that we could have him running and shooting at the same time without the need for gobs of animation data. As it was, Corvus ended up with almost 1600 frames of animation for the retail version of *H2*, and that's not including the cinematic frames.

Quake II System Refinements

Some of our biggest modifications were all under the hood of the *Quake II* engine (so to speak). Obviously, the client effects system — our client-side special magical effects generator — was one of the biggest changes. With all the special magical effects that we planned, it became obvious very quickly that the existing *Quake* effects code model would not be able to handle them. Besides (the reasoning went), it would be better to offload as many of the effects onto the client system as we could. That way, the server would be freed up to do more important game-related stuff. The way it worked was that an effect would be fired off from the game code on the server, pass general information across the network (such as effect ID, location, and its owning entity), and then send effect-specific information per effect. The client effects DLL on the client would then update the effects, and remove them when culled off screen or completed. Most effects, in fact, had no extra information, which helped out quite a lot. Programmatically, this whole system turned out to be a mixed blessing — as we will discuss later — although in game play terms, it was a resounding success.

Other refinements included modifying the software renderer to handle 16-bit graphics. Originally, we were going to handle 32-bit graphics and all modes of hardware renderers (and indeed, all the code was written to that end). After viewing the slide-show speed of the game that resulted from these efforts, however, we decided to scale that back a little. MMX implementation helped a lot, but it still required some major rewriting of the very tight and very fast original id code. We will always be indebted to our chief technologist, Gil Gribb, for helping out here.

The client prediction system required a major overhaul, for a variety of reasons. Seeing Corvus perform his animations smoothly, for example, was critical to a fun multiplayer experience. Further, most of Corvus's activity code was animation driven. Marcus Whitlock had his hands full of that particular activity, with a little help from Gil Gribb.

Concept sketch from *Heretic II*
[zoom]

We added quite considerably to the whole sound system, and split sound support off into a separate DLL. The idea was to have three separate DLLs, one for the default *Quake II* sound system, one for A3D support, and the last for Creative EAX support. We didn't get to the EAX one completed before the release dates were upon us, but with a lot of help from Micah Mason and Suneil at Aureal, we did get a 1.0 implementation of A3D in there.

What Went Right

1. The fun factor.

Aside from the fact that we finished on time (and inside of a year, no less), I am really proud of the fun game play in *Heretic II*. There is no doubt about it, hacking someone's head off in a spray of blood with gibs flying everywhere is a lot of fun (Corvus and his many ways of death rock!). Taking the time to balance the weapons with the damage, and making sure that the powerup shrines were placed appropriately paid off in spades when the game was released. The story and all the supporting elements definitely helped us make the game a success, but in the end it was about good, clean, violent fun. And not the sort that can be accused for making kids run amok with an Ak47. (Not many kids can conjure up a Red Rain bow.)

2. Our camera actually worked.

There are some other well known third-person games out there (which shall remain nameless) that, with all the charity in the world, cannot be said to have a particularly workable camera. Since a game of this nature revolves (literally) around the performance of the camera, we were especially proud of our implementation. Especially gratifying was the public commentary of some gaming communities' luminaries about how it would be impossible to go back to the conventional third-person games after playing *H2*.

"Corvus"
[zoom]

3. Our multiplayer worked out of the box.

No patches required. A lot of other gaming companies seem to regard this as a second level priority, and don't worry about multiplayer until late in the project. We were aided by *Quake II's* network model, which is particularly stable. The model did evolve considerably though, after all the touching and tweaking we did to it. The addition of the client effects system, as well as an enhanced bit packing system for networked entities, increased the complexity quite a bit, too.

4. The client effects DLL.

People were amazed that *H2* could run smoothly on a Pentium 166MHz with some relatively decent hardware acceleration. The client effects system was the reason why. Moving most of the graphics special effects code from the server to the client freed up the server considerably to do more game-related stuff. The move also allowed for better scalability on the individual's machine. So your system can't handle the ten times screen overdraw of the ring of repulsion? No worries, just turn down your level of detail, and the sprites will be drawn smaller, in fewer numbers, and scaled down faster. We didn't do this at a system level, but wrote it in an effect-by-effect way because different effects required different sorts of scaling to make them look right. Also, since the camera was handled on the client too, it allowed us to write some clever code to solve camera problems.

For example, say Corvus backed into a fire. When the camera went into the middle of a fire effect, or a waterfall, with or without the Riva TNT the frame rate would slow to a crawl. We tried various ways of solving this, such as stuffing a bounding box around the fire that the camera couldn't penetrate, but that ended up looking weird and unacceptable. In the end, we wrote some code for the effect that determined distance from the camera, and just culled it totally when it was too close. No one seemed to notice this, and it helped our frame rates considerably.

Screen shot from Heretic II
[zoom]

5. Our buoy system for AI navigation.

The buoy system was a major help, and without a doubt was the reason why our AI was better than most. Mike Gummelt and Josh Weier were responsible for it. The system consisted of beacons (or buoys) that the monster AI followed around. The buoys were dropped into the maps for single play, and had information embedded within them that allowed monsters not only to navigate the world, but also to open doors, jump from ledges, run away, engage in group activities, and warn friends that you were on your way. Of course, the game was released about the same time as *Half-Life*, the undisputed king of the current AI scene, so our advances were somewhat overlooked. Even so, the team is proud of this system, and it was undeniably a factor in the critical success of *H2*.

What Went Wrong

1. No Linux code at launch.

One acknowledged misstep we made was not commissioning a Linux Server version of *Heretic II* to be released at the same time as retail. *Half-Life* demonstrated the wisdom of having Linux code available at launch time; witness the number of servers around on launch day. *Heretic II* never caught up on this, not even to this day. We have a Linux version in the works now, but it's been delayed by the Expansion Pack and the Expansion Pack bug fixes, but it may be a case of "too little, too late".

2. Don't rebuild your physics system unless you really have to.

We did this, and it turned out to be a much larger albatross than we had imagined. It didn't help that our resident physics expert left for greener pastures after implementing it, either. The original idea was to integrate the new physics systems with the AI more tightly, fix one or two legacy bugs with physics from the *Quake* engine, and add some cool features like proper friction and inertia. As it turned out, we introduced more bugs than we fixed, and the AI integration was never used.

Screenshots from Heretic I [zoom left] [zoom right]

3. Too many cooks can spoil the stew.

Since almost every programmer at Raven had been into the code base at one point or another, there was a fair amount of redundancy in routines, and so on. It's a credit to lead programmer Pat Lipo that things went as smoothly as they did. However, inevitably, coding conflicts arise, as do competing subsystems. I think it would be safe to say that we found the point at which too many clever people working together becomes more of a liability than an asset. It is worth pointing out that the reason we had so much programming power on this project had little to do with the quality of the core team, and everything to do with the sheer quantity of work, and looming deadlines.

Concept sketch from Heretic II [zoom]

A good example of this is what happened with the client effects DLL. Although this was an extremely helpful system, it also turned into a large pile of messy code. We ran into several problems both with removing some constantly occurring effects (such as the sphere of Annihilation). If the packet with the removal code got lost, then we were in trouble. Mostly, this was taken care of by the *Quake* system, which automatically removes entities when it loses track of them. However, there were circumstances where client effects with no owner were set off into the world, and then left to the effects system to be removed. We sometimes had some problems getting effects to go away if we had a large packet drop. Another aspect that gave us problems was the sheer complexity of the system. To begin with, it started out as a very cool subsystem, with some simple culling and a clear idea of its scope. However, with up to 11 programmers all attempting to add effects and code, it soon started burgeoning into a larger subsystem. At that point, there was no one person with a complete comprehension of exactly how it all worked, with predictable results — a big, complicated mess.

4. Release timing.

We released a few days off from *Half-Life*. You can't get much more unfortunate than that, unless you release the same day as *Quake 3*. With all the anticipation that had been built regarding *Half Life*, the release time was unfortunate. On the other hand, our release date had been set by Activision when we were first green-lighted, whereas the Valve guys tended to be more floating with theirs, so it would have been difficult to plan around it.

5. A late release for the Enhancement Pack.

Once the game was released, the decision was made to go ahead with a glorified patch, referred to as the "Enhancement Pack". The idea behind this was that *H2* was pretty much free of major bugs, and as such didn't really require a patch to fix anything. However, there were some bugs in the code. The first was a joystick mistake we made. This was fixed almost immediately, and released on the net. After that, most of the team was disbanded into other projects, leaving Pat Lipo, Marcus Whitlock and myself to complete the Enhancement Pack. We started out fixing the small bugs we knew were out there, and then moved into expanding the feature set of *H2* to what we would have liked it to have been when we shipped. Most notably, we added a female model, a custom skin selection, autodownloading of skins, models, maps, A3D 2.0 support, EAX support, some new deathmatch levels, correct prediction of Corvus-related effects and sounds, a couple of new spells and a plethora of other smaller helpful additions. Once we were comfortable with all the additions we made, we decided to try a small public beta test of the Enhancement Pack. This was an extremely successful thing to do, it turned out, as much helpful commentary came back from those involved. In fact, we ran into feature creep quite a lot at the end of this burst of activity, which mainly came from the extremely helpful and interesting suggestions we got from the beta testers. We did suffer a bit of criticism from the existing *H2* community over the Enhancement Pack, because we originally predicted that we'd be done just after Christmas. That was a bit optimistic as it turned out, but our mistake was in announcing an approximate release date for the Enhancement Pack. The *Heretic II* community was not pleased about this, and made no bones about announcing it. Part of the reason for the lateness was due to the construction of the female model. With 1600 frames to build, and the fact that it was all built from scratch due to the limitations of the model format, it took longer to get it right than we originally estimated.

Lessons We Learned

Create a Linux version of your game (assuming that it's a client/server architecture game) as you develop. This is an important factor, and is now one that is almost assumed for all games. In the same way that a game suffers if there is no multiplayer option, a game will suffer from no Linux version. You don't have to go all the way — just developing a dedicated server is enough. We messed up in this regard, and simply didn't recognize the demand there would be for Linux. We won't make this mistake again.

Keep in touch with your publisher's marketing department. It is paramount that the folks marketing your game understand all of its nuances, and understand what the game means to you, the developer. Communication is of the essence here, and it will pay off big when launch time comes around.

Limit the number of programmers involved in the project. Throwing bodies at a project with a looming deadline is not necessarily the way to

get it done on time. If you use contractual programmers, or programmers from another project, be sure you have clearly defined the code that they need to write, and that you have broken the coding work into modules. This may sound obvious, but it often isn't.

Screen shot from Heretic II
[zoom]

Have a good project lead. Often the difference between an amazing game and an okay one is the ability of the project lead to describe the vision in his head to you, the developer. We were lucky that we had a damn good one. The concept art helped a lot too.

Don't hand out demo dates. Another mistake we and Activision made was in announcing a date for the demo. We were two days late, but to the frothing masses out there, two days can be an eternity. When the demo for *Hexen II* was late, Brian Raffel received over 400 mails in an hour demanding to know why it was late. Some even included death threats. Scary, eh?

Don't go out of your way to create extra code for upcoming demos. E3 is important, but it's not the be-all and end-all of whipping up interest in an upcoming game, and quite often creating a demo for E3 can be a drag on real development. The only exception to this is if you have new technology to show and you are looking for a producer — if you don't already have a delivery date, then it doesn't much matter.

It *is* possible to get games done in a realistic time frame. Having a "when it's done" mentality may work for id, but then they are, well, id. Most of us aren't. Having a deadline and sticking to it shows discipline, and helps focus you when something is due a month away. One big plus to actually making a delivery date is that all your marketing will be right on track.

A private Beta test can shake your code tree in ways you never imagined. This is not to attack any QA group, but it can act as a great precursor to the actual quality assurance process. QA-ing almost always costs you, the developer, money. Private beta tests out on the wilds of the internet cost nothing, and you end up with a legion of loyal fans that will beat your game better than anyone that's paid for it. It was a great experience of "real world" testing that we will most certainly be repeating.

"Caurthorian"
[zoom]

Supporting and being available to your community can bring rewards that are indefinable. We made a point of making sure we perused the message boards daily, answering all questions and mail as much as we could. This makes the community feel that you care, and that they can approach you. In some ways this backfire too, because there are always some fans that will cause you trouble, and by being responsive you open yourself up to abuse by them. But by and large, communicating with your fan base is almost always a positive thing to do. You will be exposed to suggestions that you would never have heard before.

Info Box:

Heretic II
Raven Software
Madison, Wisconsin
(608) 833-5791
http://www.ravensoft.com

Team Size: Producer at Activision: Steve Stringer. Project Lead: Brian Pelletier. Programming Lead: Pat Lipo. Assistant Programming Lead: Jake Simpson. Programmers: John Scott, Marcus Whitlock, Bob Love, Josh Weier, Mike Gummelt, Gil Gribb. Additional Programming: James Monroe, Rick Johnson, Josh Heitzman. Level Design Lead: Jon Zuk. Level Designers: Michael Ramond-Judy, Matt Pinkston, Mike Renner, Tom Odell, Jeremy Statz, Tim Jervis, Brian Raffel. Models & Animation: Jeff Lampo, Jeff DeWitt, John Payne, Eric Turman, Brian Shubat. Artists: Brian Pelletier, Jeff Butler, Rob Gee, Kim Lathrop, Gina Garren. Sound Design: Kevin Schilder.

Release Date: November 1998
Estimated budget: $1 to 1.5 million

Time in Development: 11 months

Typical workstation: Started out on Pentium Pro 200's, and graduated to P II 400's half way through the project

Critical Applications: Visual C++ 6.0 dev studio, Visual Sourcesafe 6.0, 3D Studio Max 2.0, Photoshop 4.0, SoftImage 3.8, QERadiant level editor

Intended Platform: Windows 95/98/NT

Jake Simpson worked on C64, Amstrad, and Sinclair Spectrum games back in the early eighties, then ducked out of gaming computing into the corporate world (after being on the receiving end of a bad situation between a UK publisher that shall remain nameless, and a software house that shall also remain nameless). After trying his luck out in the world of Big American cars and McDonalds, he hit it lucky and ended up working at Midway for six years, helping out such arcade projects as* NBA JAM *(Nani Edition),* WWF Wrestlemania*,* Revolution X*, and even doing a bit of work on* Mortal Kombat 3*. His most recent post is at Raven Software, developing first- and third-person shooters using id 's* Quake *technology. If you feel the need for feedback, he can be contacted at jsimpson@mail.ravensoft.com.

Return to the full version of this article