GAMASUTRA
The Art & Business of Making Games

PRINT

Nihilistic Software's *Vampire: The Masquerade -- Redemption*

By Robert Huebner

When Nihilistic Software was founded in 1998, there were only two things we knew were certain. The first was that we wanted to form a company with a small number of very experienced game developers. The second was that we wanted to make a killer role-playing game.

Nihilistic got started without much fanfare, just a few phone calls and e-mails. After finishing work on *Jedi Knight* for LucasArts, the core team members had, for the most part, gone their separate ways and moved on to different teams or different companies. About eight months after *Jedi Knight* shipped, various people on the original team began to gravitate together again, and eventually formed Nihilistic just a few exits down Highway 101 in Marin County, Calif., from our previous home.

Having moved into our new offices and bolted together a dozen desks from Ikea, our first project was to build a 3D RPG based on White Wolf's pen-and-paper franchise, Vampire: The Masquerade. Before linking up with Activision as our publisher, Nihilistic president Ray Gresko already had a rough design and story prepared for an RPG with similar themes and a dark, gothic feel. After Activision approached us about using the White Wolf license, we adapted parts of this design to fit the World of Darkness universe presented in White Wolf's collection of source books, and this became the initial design for *Redemption*.

Because of our transition from first- and third-person action games to RPGs, we approached our first design in some unique ways. Many features that are taken for granted in action games, such as a rich, true 3D environment, 3D characters, and the ability for users to make add-ons or modifications, were reflected in our project proposal. We also adopted many conventions of the FPS genre such as free-form 3D environments, ubiquitous multiplayer support, and fast real-time pacing. To this we added the aspects of traditional role-playing games that we found most appealing: a mouse-driven point-and-click interface, character development, and a wide variety of characters, items, and environments for exploration.

Professional conceptual art, such as this rendering of Alessandro Giovanni by contractor Patrick Lambert, helped the characters evolve as the art design took shape.

Using the White Wolf license also meant that our users would have high expectations in terms of story, plot, and dialogue for the game. It's a role-playing license based heavily around dramatic storytelling, intense political struggles, and personal interaction. Fans of the license would not accept a game that was mere stat-building and gold-collecting.

In keeping with our basic philosophy, we built up a staff of 12 people over the course of the project's 24-month development cycle. The budget for the game was fairly modest by today's standards, about $1.8 million. The budget was intentionally kept low for the benefit of both Nihilistic and our publisher. We wanted our first project to be simple and manageable, rather than compounding the complexities of starting a company by doing a huge first project. Also, we were looking to maximize the potential benefits if the game proved successful. For its part,

Activision was new to the RPG market and was testing the waters with RPGs and the White Wolf license in particular, so they probably considered the venture fairly high risk as well.

Development started around April 1998. When we began, we examined several engine technologies available, such as the Unreal engine and the Quake engine, but ultimately decided against licensing our engine technology. The game we envisioned, using a mouse-driven, point-and-click interface, had a lot more in common with games such as *Starcraft* than even the best first-person engines. We decided to create a new engine focused specifically on the type of game we wanted to create, and targeted 3D-accelerated hardware specifically -- bypassing the tremendous amount of work required to support nonaccelerated PCs in a 3D engine. As an added benefit, the company would own the technology internally, allowing us to reuse the code base freely for future projects or license it to other developers.

What Went Right

1. Letting the artists and designers pick their tools.
With such a small team and tight budget, boosting the team's efficiency was our primary focus. If bad tools or art paths slowed down progress in the art or level design departments, we would have no chance of hitting our milestones. When we started to map the development project, the programmers gravitated toward using a package such as 3D Studio Max for both art and level design. Our argument was that doing everything in a single package would increase portability of assets between levels and art, and save the company money by licensing a single, relatively inexpensive tool. Thankfully, however, our leads in these areas strongly objected to this plan. They argued for allowing each department to use the tools that allowed them to do their work most efficiently. This single decision probably accounted for more time saved than any other.

The level designers cited QERadiant as their tool of choice, since most of them had previously done work with id Software on *Quake* mission packs. id was generous in allowing us to license the QERadiant source code and modify it to make a tool customized to our 3D RPG environments. Because QERadiant was a finished, functional tool even before we wrote our own export module, the level designers were able to create levels for the game immediately, even before an engine existed. And since QERadiant stores its data in generic files that store brush positions, the levels were easily tweaked and re-exported as the engine began to take shape. If the level designers had spent the first six months of the project waiting for the programmers to create a level editing tool or learning how to create levels in a 3D art tool, we would not have been able to complete the more than 100 level environments in 24 months with just three designers.

Locations included both interior and exterior cityscapes, allowing dramatic situations such as this battle atop a clock tower in medieval Prague.

On the art side, lead artist Maarten Kraaijvanger lobbied hard for the adoption of Alias|Wavefront tools for 3D art. We tried to convince him that a less expensive tool would work just as well, but in the end we decided to allow the art department to use what they felt would be the most efficient tool for the job. Since Maya was just being released for Windows NT at that time, the costs of using that toolset were not as great as we feared, and it allowed the artists the produce an incredible number of 3D art assets for the project. During the 24 months of the project, an art department of four people produced nearly 1,500 textured models, a mind-boggling figure using any tool.

2. Small team, one project, one room.
When we started Nihilistic, we had a theory that a small number of highly experienced developers would be able to produce a title more efficiently than a larger team with fewer battle scars. In my experience, successfully delivering a game is less about what you do and more about what you choose *not* to do. Most games that ship late do so because the development team went down one or more "blind alleys" -- development ideas or strategies that for whatever reason didn't pan out, and the work done in that direction is lost. As a small team on a tight budget, we could not afford to lose valuable time on these diversions. Experienced team members have the wisdom to look down a particular path and recognize when it's a likely dead end.

Developers that have shipped commercial titles also know when "enough is enough," so to speak. There is a rampant problem in this industry of feature creep, when games end up trying to be all things to all people, and wind up taking four years to complete. Seasoned developers

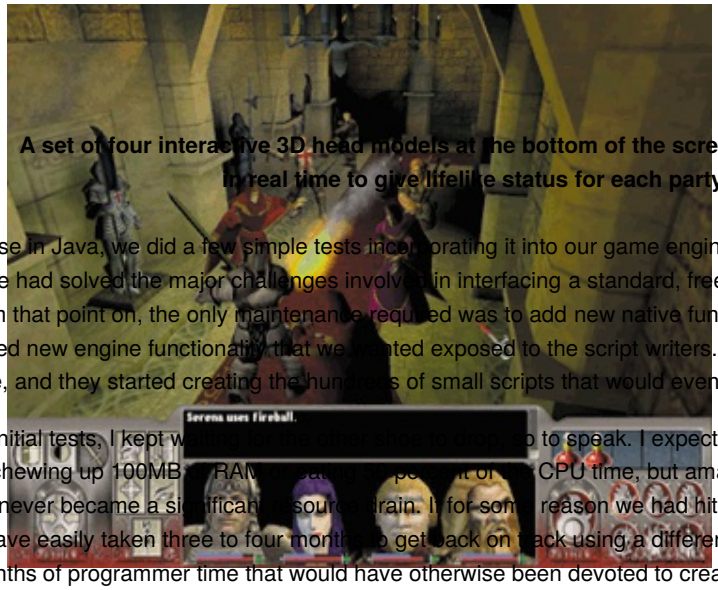
know that shipping a title is all about compromise. Any title that goes out the door could always be "just a little better" and developers, ever the perfectionists, are never fully satisfied with the box on the shelf. Creating a successful game that ships on time requires the discipline to draw that line and move on to the next challenge.

We also knew that we wanted an office environment where all the team members were in a single room without any walls, doors, or offices whatsoever. This didn't really seem like a radical decision -- many of us got our start working for teams that operated like this -- but it seems like these sorts of companies are becoming less and less common in today's industry. My first game job was working at Parallax (now Volition) software. We were eight people sitting along one wall of a narrow office space in Champaign, Ill. Even the original *Dark Forces* development team was sequestered in a one-room studio in a building separate from most of the other LucasArts teams. This type of environment doesn't just foster, but rather *forces* communication between all parts of the team. For instance, a programmer can overhear a discussion between two artists about how to proceed with something and be able to jump in with an answer that will save the project days or months of work. This sort of thing happens on a daily basis; artists correct missteps by the technology team before they are made, a level designer can immediately show a bug to a programmer, and so on. Each of these incidents represents hours or days of project time saved. In an office environment with walls and doors, most of these situations would go unnoticed or unaddressed.

3. Using Java as a scripting engine.
We knew from the start that allowing the user community to edit the game was an important part of the design. After working in the first-person action-game market, we saw the benefits of supporting the user community and wanted to carry this idea over into role-playing games, where it is not the norm. A built-in scripting system makes a game engine much more extendable by fans. In *Jedi Knight*, we created our own customized game language called COG. Creating COG took a lot of effort from the development team; several months of work went into creating the compiler, testing the generated code, and implementing the run-time kernel used to execute the scripts. The end result was worth it, but it cost a lot in terms of time and resources to pull it off (for more about COG, see my article, "Adding Languages to Game Engines," September 1997).

The ambitious design included parties of up to four 3D characters, each with interchangeable weapons and armor.

When starting *Vampire*, we looked for ways to incorporate a scripting engine more easily than creating our own from scratch yet again. There were several scripting systems we examined and tested. At about that time, another game development company, Rebel Boat Rocker software, was getting a lot of attention for its use of Java technology. After exchanging a few e-mails with lead programmer Billy Zelsnak, we decided to give Java a try. Up to this point I knew very little of Java, and had largely dismissed it as a language suitable only for making icons dance on a web page and the like.



A set of four interactive 3D head models at the bottom of the screen are skinned and animated in real time to give lifelike status for each party member.

After a crash course in Java, we did a few simple tests incorporating it into our game engine. It passed each one with flying colors. In a matter of a few weeks, we had solved the major challenges involved in interfacing a standard, freely distributable Java virtual machine to our 3D RPG engine. From that point on, the only maintenance required was to add new native functions to the scripting language, which we did whenever we added new engine functionality that we wanted exposed to the script writers. We also trained several designers in the use of the scripting language, and they started creating the hundreds of small scripts that would eventually drive the storyline of the game.

Ever since those initial tests, I kept waiting for the other shoe to drop, so to speak. I expected to come to work one day and find out that the Java thread was chewing up 100MB of RAM or that it was using 50 percent of the CPU time, but amazingly, the system was trouble-free throughout development and never became a significant resource drain. If for some reason we had hit a dead end with the Java system late in the project, it would have easily taken three to four months to get back on track using a different scripting technology. In the end, the gamble paid off. We saved months of programmer time that would have otherwise been devoted to creating a scripting environment, and the result was a system significantly more efficient and robust than any we could have created ourselves.

All of the more than 100 3D characters, such as Lucretia, a Setite priestess, were modeled and animated by hand by a team of four artists using Maya.

4. Storyteller mode.

Throughout the project, the design slowly took shape through a series of meetings that involved the entire staff. Each new design element was presented to the group and subjected to a (sometimes heated) discussion. This process of open discussion and free exchange of ideas resulted in a lot of the most interesting design aspects of the game.

It was in one of our earliest design meetings that we came up with the idea of developing the multiplayer aspect of the game not as a typical deathmatch or cooperative system, but rather to create a "storyteller" or "dungeon-master" system. The idea was inspired by the venerable text-based multi-user dungeon (MUD) games that date from a calmer time in the history of the Internet. Many of us at Nihilistic had played MUDs in college, often to the detriment of our studies. One thing that made MUDs so appealing was the ability for "wizards," high-ranking users of the MUDs, to manipulate the game environment and create virtual adventures for the players in real time. The Vampire license from White Wolf emphasizes the role of the "storyteller," or moderator, so we felt the time was right to take this style of play out of the college computer lab and into a commercial RPG.

Implementing the storyteller system turned out to be fairly simple from a technology standpoint. Most of the basic functionality for a storyteller game is identical to what would be required in a traditional client/server multiplayer game. The added cost was mostly in the area of design and the user interface. It took a bit of experimentation and redesign to arrive at an interface that was powerful enough to run games as a storyteller without being overly confusing to the novice player. The UI work included new interface panels with lists of objects, actors, and other resources, and a few buttons to manipulate the selected resources. Our overall design goal for the user interface was to ensure that important functionality was accessible using only the mouse, and all keyboard functionality represented only "advanced" controls such as hotkeys and shortcuts. Even though the storyteller system is something used primarily by advanced players, we wanted to preserve this design goal, which meant quite a bit of extra UI work to make a mouse-driven interface powerful enough to drive a storyteller game.

In the end, the storyteller feature ended up being one of the gems of the game design, and resonated with both the press and gamers alike. Activision made good use of the feature in their PR and marketing campaigns, and we hope the expandability and storyteller aspects of the game will give the game an increased shelf life.

5. Using experienced contractors.
One problem with our strategy of using a small core team is that we couldn't possibly cover all the aspects of designing a commercial game with just 12 people. Instead, we relied heavily on external contractors for certain key aspects of the game.

Sound was one area where we made use of external talent. Our colleagues from LucasArts referred Nick Peck to us, based on his excellent work on Tim Schafer's *Grim Fandango*. Nick ended up not only supplying us with sound effects, but also working on some of the additional voice recording and ambient loops. For our music, we teamed up with Kevin Manthei who scored the Dark Ages portion of the game, and with Youth Engine, a local duo, for the modern-day tracks.

Even in the conceptual stages, we used external artists to help us sketch and visualize the game. Peter Chan was the lead conceptual artist for *Jedi Knight* and had subsequently become an independent contractor. His work in the first months of the project was key in establishing the look of the game's environments. We also worked with Patrick Lambert for character concepts and he delivered incredibly detailed full-color drawings that really brought the characters to life for the modelers and animators.

Perhaps the most critical external relationship was with Oholoko, a small startup spun off from Cyclone Studios. We hired them to do our cinematic sequences that introduce the story and provide the endings. While starting the project, we met with several firms specializing in computer animation, but pretty much across the board their rates were well beyond our budgets for that part of the game. It seems that the high demand for computer animation from movies and television has driven the larger firms' prices beyond the reach of typical game budgets. By working with a smaller, less established company, we were able to get more bang for our buck in our cinematics, and the results proved to be of the highest quality.

What Went Wrong

1. Overly ambitious design.
In retrospect, we were in some ways our own worst enemy. Many of the team members had wanted for some time to do a really huge, ambitious role-playing game. When we actually started the project and had a budget and schedule, we probably weren't realistic about how long RPGs typically take to develop, especially one that travels to four different cities across an 800-year timeframe. We were very reluctant to make big cuts in the design, such as cutting one of the two time periods or removing the multiplayer aspect. Because of this, we eventually had to make the decision to miss our first scheduled release date of March 2000. We also cut back on our plans to release an interactive demo some months before the game and scaled back the scope of the multiplayer beta.

Fortunately, by expanding the schedule a few months (from March to June), we were able to preserve almost all the elements from the initial design. But to accomplish this, the art and design departments really had to work above and beyond the call of duty for an extended period of time.

We did cut back a bit in the area of multiplayer by removing the ability to play through the entire single-player scenario cooperatively as a team, and instead replaced that with two smaller, custom-made multiplayer scenarios using levels and art from the single-player game. Part of this was because we did not plan properly for multiplayer when making some of the Java scripts that drive the single-player game. If the multiplayer game had been functional earlier in the schedule, the single-player game scripts might have been written from the start to be "multiplayer friendly" and we could have shipped more multiplayer content in the box.

Characters were created with a budget of between 1,000 and 3,000 triangles. Boss characters, such as Ahzra the Tarmisce Elder, were generally the most complex.

2. Prototyping with a proprietary API.

When we started developing the 3D engine, which we named [illegible]D API landscape was quite a bit different from how it is now. We decided to use Glide as an initial [illegible] API with the belief that it would be a more stable platform and avoid the complexities of supporting multiple hardware through a more [illegible] API until we had [illegible] engine a bit. However, once we had a basic, functional engine running under Glide, the programming [illegible] turned toward [illegible] functionality rather than switching the graphics engine to a more general API such as Direct3D or [illegible].

Because of this "if it ain't broke" mindset, we [illegible] our support be[illegible] late in development. At the first public showing of the game at E3 in 1999, we were still basically [illegible] only game, which meant we [illegible] couldn't demonstrate the game in 32-bit modes or support some features not present in Glide at the time.

The extensive use of Glide also gave us some [illegible] performance estimates for other hardware. Since Glide allows low-level access to things like texture-memory management, we spent significant time writing our own optimized texture manager. When we switched to Direct3D, most of this work had to be discarded. Since Glide allows more flexible vertex formats than Direct3D, some of our underlying data structures needed to be changed, which meant re-exporting hundreds of levels and models. We were making low-level architectural engine changes at a stage when the engine should have been pretty much locked down. Also, because we switched late in our development schedule, we probably didn't spend as much time as we should have on compatibility testing with a wide variety of hardware. In retrospect, we should have switched to Direct3D or OpenGL several months earlier in the development schedule.

3. Pathfinding difficulties.

One problem we identified early in the development process was the problem of pathfinding. Navigation of variably-sized characters through a completely free-form 3D environment is one of the most difficult problems I've had to tackle as a game programmer. Unit navigation is hard enough when you have a flat 2D plane or restricted 3D environment, but in an environment where the level designers are free to make stairs, ramps, or any other 3D construct you can imagine, the problem becomes exponentially more difficult. My natural tendency when presented with such a sticky problem is, unfortunately, to make it good enough for the early milestone and demo builds, and then just "deal with it later." Unfortunately, "later" quickly became "now," and "now" turned into "yesterday."

Real-time continuous level of detail allowed models to appear highly detailed in close-ups without sacrificing speed in longer shots.

We should have tackled this problem much earlier, before the levels were near completion. We should have worked with the level designers to come up with a set of restrictions for their levels, or some additional tagging in the editor to specify to the engine where characters should and should not move. Instead, the only hints from the level-design tool were "walkable" floor flags, but little or no special marking of walls, cliffs, and other pathing hazards. Since we waited too long to address the problem, better solutions such as walk boxes or walk zones would have taken too long to retrofit into the more than 100 levels already in the can. Instead, we spent weeks making small iterative fixes to the system to hide the most extreme errors and turn what was an "A" bug into a "B" or "C" level problem.

4. Feature and data timing.

This is a fairly common problem in games I've worked on, and *Vampire* was no different. The technology team typically looks at the development schedule and schedules that entire block of time to achieve a certain feature set. Often, however, new engine features get added too late in the schedule to be utilized fully by the designers and artists. This happened several times during *Vampire*. Some of the

more interesting special effects, for example, were added only a few weeks before the data was to be locked down for final testing. Other features that we added couldn't even be implemented extensively. For example, we added a more flexible shader language so late that only one to two percent of the surfaces in the game were able to take advantage of it. Some features that we had originally planned for the engine, like bump mapping and specular lighting, were cut completely from the initial release because there was insufficient time both to complete the feature and to create art to drive it. We softened the blow somewhat by moving some of these features to a planned patch, which would add them later if the game proved successful.

Unfortunately, there are very few programming tasks that don't require some sort of artist or designer input to find their way into the finished product, so unless programmers spend the last six months of the project doing nothing but fixing bugs, some of this is inevitable. We can justify it to a degree by looking toward the likely sequel or add-on projects as a way to take advantage of some of the engine work that was underutilized in the original title.

5. Self-restraint.

As the project was drawing to a close, we found that we ended up with a bit "too much game," as someone put it. From the start, we decided to author our data for a high-end platform, so we'd have a good-looking game at the end of the 24-month schedule, and also because it's much easier to scale art down than up. Unfortunately, we never really started to rein in our art and design teams when we should have near the middle of the project. Instead, we continued to add more and more resources to the project, resulting in a minimum installation footprint of about 1GB.

We authored all our textures in 32-bit color and then scaled them down at load time for 16-bit cards. Our models were also extremely detailed (1,000 to 2,000 triangles each, on average) and relied on automatic level-of-detail algorithms to scale them down for slower machines. We lit our levels with relatively high light-map resolutions. All of this made the game look great on high-end systems, but it meant the game was fairly taxing on low- to mid-range systems. In the end, the game just barely fit on two CD-ROMs.

The game's story unfolds mainly via in-game cutscenes. The excellent models allowed for the creation of intricate details such as individual fingers, which helped make these scenes feel like pre-rendered cinematics.

We had originally planned to include both 16-bit and 32-bit versions of the game textures and allow players to choose which version to install, but after all the art was completed there was no room on the CD for more than one version. Likewise for sounds: we wanted to include multiple quality levels but space prevented this. We actually compressed most of the voice samples with MP3 and had to remove several sounds from the game in order to fit it on two CDs.

In the end, our game looked gorgeous but had difficulty running on machines with less than 128MB of RAM -- and even then, it used a fair amount of space on a swap drive. This glut of resources will also make it more difficult if we choose to port the game to a more limited console environment.

At Last, Redemption

For the first project from a new development startup, I can't imagine how things could have gone much better than they did, except perhaps if we could have avoided shipping it the same year as *Diablo 2*. As a company, we managed to accomplish the three most important things in this business: not running out of money, not losing any team members, and actually shipping the product. Our publishers remained committed to the project throughout its life cycle, and even increased their support as the project continued to take shape.

The course of development was amazingly smooth, with very few surprises or conflicts along the way. In this industry, you can almost bet that at some point in a two-year development cycle, something traumatic will happen to either the development team or its publisher, but for

us the waters were remarkably calm. About the most exciting thing to happen during development was when we lost our entire RAID server while attempting to add drivers to it, resulting in the loss of a few months' worth of archived e-mails.

Our good fortune allowed the team to focus strictly on the game and prevented distractions from outside the company. Also, keeping our company focused on just one title and resisting the frequent temptation to take on more work and more staff allowed everyone to be on the same team with little or no secondary distractions.

Hopefully, by avoiding feature creep and a four-year "death march" kind of ending to this saga, we can avoid a lot of the burnout that we have seen and often experienced on other teams. By maintaining links with both the fan community through our web board, and with the developer community at large by attending shows like GDC, E3, and Siggraph, our team was able to keep a positive attitude and high energy level throughout the schedule. We remain convinced that small development teams with a single-title focus are the best way to ship quality titles consistently, so our plans moving forward are to staff up gradually from 12 to perhaps 16 people over the next few months and embark on our next two-year ordeal a little older, a little wiser, and just a tiny bit larger.

Game Data

Publisher	Activision
Full Time developers	12
Contractors	8
Budget	$1.8 million
Length of development	24 months
Release date	June 2000
Platforms	Hardware-accelerated PC
Hardware used	Intel and AMD PCs, Nvidia and 3dfx 3D accelerators
Software used	Alias|Wavefront Maya, Photoshop, QERadiant, Visual C++
Technologies	3D skinned characters, continuous level-of-detail, custom-built 3D engine, MP3 audio compression, lip synching Lines of code: 300,000 for game, 66,000 lines of Java for scripts.
Lines of code	300,000 for game, 66,000 lines of Java for scripts.

:

Robert Huebner is a co-founder and lead programmer at Nihilistic Software, an independent game developer located in Marin County, Calif. Prior to working on Vampire: The Masquerade, he contributed to several other game projects, including Jedi Knight: Dark Forces 2, Descent, and Starcraft. Robert is on the advisory board for the Game Developers Conference and has presented a number of sessions there, as well as at Siggraph and E3.

Return to the full version of this article