Postmortem: Angel Studios' *Resident Evil 2* (N64 Version)

By Todd Meynink

In October 1997, two employees at Angel Studios posted a message to themselves on a wall: "We will release a game on September 1, 1999, that people love. We will know they love it because by January 1, 2000, it will have sold two million copies."

Shortly after, Capcom chose Angel Studios to port *Resident Evil 2* to the Nintendo 64. Project director Chris Fodor and lead programmer Jamie Briant were charged with this task, and began laying the groundwork and assembling the team. While they both had previous experience with the Nintendo 64, this ambitious and challenging project quickly illustrated that they didn't really know how the N64 worked. Sure, its got a CPU and a geometry processor, and a graphics chip, and fire exits here, here, and there, but at exactly what altitude do the oxygen masks drop down, or do they have to be triggered manually by the pilot? In the next few months the OS was rebuilt, vector units (yes, the N64 has a vector unit) emancipated, and the N64 charmed into revealing her secrets.

The original *Resident Evil 2* for the Playstation spanned two CDs. We had to get it on a single cartridge. But it's just a port, right? In our hands we had a classic game with excellent design, all the art done, and the AI tuned and in place. However, it was going to require some pretty clever programming to get it running on the N64. It was a task that the newsgroups, gaming web sites, and perhaps even the publisher had doubts could be done.

What Went Right

1. The Work Environment and Team

Given that this was going to be a very technical project, it was critical that we had a strong and cohesive programming team. The project quickly ramped up as Alex Ehrath came on board and then me shortly after in December 1998. Alex brought a wealth of experience, dedication, and hard work. Fresh out of college, I was handed the task of doing the full-motion video. (For details on how this feat was accomplished, check out my article "Mission: Compressible -- Achieving Full-Motion Video on the Nintendo 64" in the upcoming September 2000 issue of *Game Developer.*) Chris Fodor and Jamie Briant shared a joint leadership of sorts. Sometimes this led to "too many cooks in the kitchen," but more often than not they complemented each other and provided effective leadership through the project's duration. Ken Kamdar came on towards the end of the project, bringing a sense of serenity and calm (as well as his programming skills) at just the right time. By this stage we had Australia, Germany, England, Iran, and the U.S. represented. You wouldn't have picked it, but this unusual mix had a level of synergy rarely found on many programming teams.

Screenshots from *Resident Evil 2* for the Nintendo 64

We programmers had our desks arranged in a semicircle, facing outward, but it hadn't started out that way. Initially, Chris had his own office, and Alex and Jamie sat quite a ways apart, staking out their own territory. However, a month into the project we were way behind schedule. We needed to be on top of things, and to communicate a lot more.

We decided to sit together. When I, and later Ken, arrived they just enlarged the semicircle. A lot has been written about brain states and the zone, and as Jamie recalls, "Alex would sometimes interrupt me with all my balls in the air to tell me some really stupid joke, but I guarantee that the time lost was nothing compared to the gain in efficiency of having everyone right there". At the beginning of the project there are lots of questions that come up and decisions to be made. Everyone could listen in, even if they weren't initially part of the discussion, and often someone would turn around and offer a pearl of wisdom or a new insight that led to a better solution. By the middle of the project, we'd got better at not interrupting anyone deep in thought. In the end, I think we saved months: "It crashed" *[points]. [Turns head]* "Oh yeah, that's the licker bug -- don't press the B button when it's doing that *[points]* -- and I'll be checking in a fix in ten minutes."

The long, hard hours forged a strong mutual trust among the group. It wasn't long before we were all working away on our own tasks without supervision. This allowed for flexible hours without jeopardizing work throughput.

2. Nailing the First Milestone
If your company doesn't publish its own titles, then you have an external producer at your publisher. When the producers go out for a beer, they talk about how late their developers are and bitch about how to get them in line. Many producers think developers haven't got a clue about marketing deadlines, budgets, and all the other things that we really don't have a clue about. We speak a different language. But if you hit your first milestone, you're different.

First, your producer has a different story to tell to his boss and his peers. Second, you've established a basis for communication -- you've demonstrated that you understand what the word "deadline" means. Having learned the first word in a producer's vocabulary, you'll find that they are then open to discussing more complicated ideas, even being flexible on future deadlines. Finally, you have credibility with the producer and your publisher. If you miss your first milestone without a care, your producer will permanently put you in the "baby sit/baby talk" bin.

3. No Religious Attachment
It's never a good idea to become too attached to something you've done, whether it's a business process, an algorithm, or just an implementation. If something doesn't work, don't do it. If something did work, and it doesn't anymore, drop it and find something new.

We applied this principle to our group communications. We began by using Microsoft Team Manager 97. That didn't even last a month. Then we moved our desks into the semicircle, with everyone close at hand and in the same room. That worked very well, and we kept it.

Next, Jamie decided to see what would happen if everyone was forced to use the same editor, with the same keyboard choices. The result was that you learn the new editor in a day, you're fluent in a week, and everyone can sit down at your machine and use it like it was their own.

We used Outlook's tasks system. Outlook's seemingly simple task list can use e-mail to keep everyone's tasks in sync. This worked very well for several months, but after a while we used it less. This was because at the end of the project, there was simply a whole list of very well known things that needed to be done. We tried a very visual display, by making pieces of paper for every task item and pinning them on the wall under everyone's name. And there they stayed, pretty much untouched, until the day we published the game. They were replaced with Excel spreadsheets.

When it came to programming, this attitude worked wonders for the development of our FMV system. Relentlessly trying everything, often multiple times (as an improvement in quality or speed made a previously rejected approach viable again), brought us a great result and an industry first -- high-quality video on a cartridge-based console.

We also learned the value of writing code for the task at hand, not what the task might be in the future. I hope most programmers find this obvious, but many of us (including me at one time) have been lost in C++ land. There are many benefits to C++, but a lot of them just aren't viable for performance and size reasons on an aging console. It's very easy when writing object-oriented code to fall into the trap of trying to design even a reasonable system. Don't bother. Be prepared to rewrite everything. You might be able to rewrite some section of code three times faster than it would take you to design a perfect class hierarchy. The bottom line is that you shouldn't be attached to what you have. Be prepared to throw it away and try something else.

4. Using a Detailed Schedule and Plan
From the outset, *RE2* had a clear and detailed plan of exactly what would be required, broken down into very fine detail. With this information our capable producer, Stewart Spilkin, was able to schedule the project's tasks and resource allocations accurately (Stewart was instrumental in dealing with external difficulties, which allowed us to concentrate on development, always pushing the project closer to completion). I can't overstate the importance of a detailed plan. It forces you to examine and often discover what really needs to be done and allows you to plan for it. You can't have too much detail. It was an ambitious schedule to be sure, but one that was attainable -- which made it both hard and rewarding.

We prioritized features. As a deadline rolled up, we ruthlessly worked on essential, strategic features only. Occasionally there would be arguments over what features were and weren't necessary, but this attitude ensured that we stuck to the plan and got the project done. We wanted our game to be perfect, but it had to ship. Do what's needed to get it out the door, then make all the touch-ups you have time for.

We dealt with the publisher's requests to add new features, especially towards the end of the project, with aplomb. Rather than an internal attack of "feature creep," these came from the outside. Each time a new feature was proposed, we examined what it would take to implement it and presented an honest account of what it would take, in terms of resources, to implement it. For example, we estimated that with an additional full-time programmer we could definitely achieve task A, probably task B (80 percent) and maybe task C (20 percent). The client then had all the information they needed to make a choice and most times they chose "no."

Dealing with these added pressures can be stressful, and each time it takes you away from your work. However, you can save your project's schedule and budget by rationally examining what needs to be done and what implementing that new feature will require.

5. Maximizing Reuse
With the vast amount of assets provided by the PSX version, it made much more sense to analyze each type of data (2D sprites, collision data, and so on) and implement a pipeline that would then convert the PSX-specific assets into something we could read in and use on the N64. Since this was a port, we could be sure that we would end up with all the pieces we needed if we simply batch-converted them in their

entirety, rather than individually touching up each and every one of them by hand, and save enormous amounts of time this way. It took a few days to write code that converted all of the 2D sprites into a format we needed, but it would have taken an artist a very long time to touch up thousands of sprites by hand.

Whenever possible, we emulated PSX-specific routines and hardware functions to achieve similar results on the N64, maximizing our reuse of the existing source code.

What Went Wrong

1. Submission Process to Nintendo
It's Murphy's law: for a project that had run so smoothly, our biggest problem came in the final hour, after the game was "complete" and we were awaiting approval from Nintendo. We were all ready to kick back and congratulate each other on a job well done, but our title still needed approval from Nintendo, both in Japan and in the U.S. The ensuing delays caused us to miss our ideal release date, which in the U.S. was Halloween.

While it would be easy simply to complain about how terrible big publishers are, there are things you can do to mitigate the bureaucracy. The thing to keep in mind is that at some point there are decision makers involved with approving your game. Anyone who has shipped a game, especially for Nintendo, knows that even the most minor misbehavior can hold up the process. So what to do now that your game is being sent into the hands of who knows how many random individuals? Simple: realize that there is a process.

Someone at your company will hand the game to someone at your publisher. Someone at your publisher will hand the game to your console maker for approval. That person will subsequently hand the game (and associated materials) to someone else at his company, possibly overseas. At some point the process will reverse itself with either a "yes," or a "no" and a reason. You must involve yourself in this chain so you can respond to mandated changes rapidly. Glue a cell phone to your ear, and send a copy of your contact information to as many persons in the approval process as possible. Get to know them personally if you can. Ultimately you'll find that these people would actually like to publish your game and it is equally frustrating to them to have to knock it back over a minor detail.

If you can talk to the people directly you can fix things much faster and literally save days. You will also know where your game is at any given moment. Instead of imagining it simmering inside the convoluted bowels of some demonic company, you will know that Sam said he passed it to Bob and you will be able to call Sam and please ask him to go over to Bob and request that he do something with your game. This is often overlooked in a small team that has been very internally focused for an extended period of time. Suddenly having to rely on someone else, or a great many someone elses does not come naturally. Force yourself to ride the phones and know your status at any given time -- you deserve it. It was a joyous moment for Chris when he got a call from someone at Nintendo whom he'd never met before, changed a single bit in a few moments' time, and sent the game back on its merry way in 30 minutes. He'd simply heard incorrectly through the grapevine what the actual status bit was supposed to be.

Screenshots from *Resident Evil 2* for Nintendo 64.

2. Insufficient Testing
Never underestimate your testing requirements and, more importantly, never let your publisher underestimate your testing requirements. Almost obvious to anyone who has developed a game should be the testing phase, especially for projects with shorter deadlines. Make sure that the level of testing support you expect is written into the contract with your publisher.

As an aside, programmers and artists are not testers. Such a practice reduces their effectiveness when they are called upon and at the time when the utmost care must be taken to ensure their changes do not have ripple effects elsewhere in the code. Having them on call 24 hours a day is fine, but have them be on call, not sleeping on the floor.

Exact definitions of what developers expect from the publisher and what the publisher expects from the developer are crucial to ensuring expectations are met on both sides. Our testing operation suffered because we expected the publisher to test the game, while they expected us to test it. A testing plan was not mentioned in the contract or the porting plan -- an oversight by both the team and Angel Studios. In the future, we will be sure to have a testing plan explicitly stated in both the contract and the development schedule.

3. Compression

While we achieved a tour-de-force of compression overall, we didn't anticipate all the assets that would need to be compressed. Audio and video received most of the attention and were pushed and squeezed until they fit. Our focus on these assets, however, caused us to neglect other areas such as the animation data. We did manage to cage these beasts, too, but on our final burn we had to shave another megabyte off the video. This last megabyte took us under the critical point where the video suddenly went from very pretty to just a bit too obviously compressed. Not all the game's movies were affected, but it was disappointing that we had to sacrifice some quality because we assumed the other assets would fit, rather than adhere to the apothegm "assume nothing." Next time, we won't leave anything to afterthought. We'll examine everything, to the point where we can make informed estimates (in this case, an asset's uncompressed size and its expected compression ratio derived from some sample tests) for everything.

4. Lack of Follow-up by Leads

Everyone has a certain set of things that they hold dear. For some of us our work integrity is very important to us, for others it is not. As I mentioned before (and will mention again), when you show that you understand the word "milestone" you set yourself apart in the game industry. This is truer in a game where your current progress can be held up against a completed yardstick. Not everyone, and certainly not everyone on your team, will necessarily share this understanding.

In a perfect world, everyone works hard, everything works together, and no one needs supervision. However, it is the lead's responsibility to identify those tasks and people that do and to get a commitment from them stipulating what they will do in writing. Remind them of these commitments as often as necessary. It is equally important that the individual and the team as a whole understands that "done" means "complete, meets quality standards, requires no future adjustments or tuning, and is ready to ship." When you do this you set up both the team and that individual for success.

5. Not Making Audio a High Enough Priority

At the beginning of the project, this was the most underestimated and underbudgeted task. We quickly realized the scope: at least 200 pieces of music, many short, but there nonetheless with each MIDI piece having its own unique samples -- a nightmare for conversion to cartridge. Angel Studios simply did not have the know-how to execute this area of the project.

The tool we had developed internally required a ten-minute compilation process between the alteration of a sequence or sample and hearing that sequence. Obviously, when you are going to be attempting to refine more than a thousand individual samples not only for correctness but also to make them as small as possible, this system simply would not work. Fortunately, we had a connection on our team in the German development community that used to play videogames in his basement with Chris Huelsbeck from Factor 5.

Factor 5 is developing Nintendo's next-generation sound tools. They have a system that reproduces on a PC development system exactly what you would hear on the Nintendo. Any change could be heard in an instant and samples could be continuously refined until they met size and quality constraints. They jump-started the conversion process for us and trained our people on their tools, while converting half of the MIDI files and all of the instrument samples. Notes that sounded exactly the same that were scattered across multiple MIDI files became collated and used repeatedly in the sound bank. Each looping sample (such as a reverberating piano note) was shaved and shaved again by Factor 5's Rudi Stember, while Chris Huelsbeck was involved in the MIDI conversion process. The result was better than anyone expected. Finally, using their sound driver on the N64 allowed us to use Dolby Surround Sound to place sounds in true 3D space, making the in-game sound far superior to its PSX predecessor.

Miscellanea

Now I'm just going to take a minute to vent a little steam. Whoever invented the "big-endian" convention should be shot. We saw this coming, so I can't say that it totally went wrong, but it was a lot of hard work due to the myriad magic numbers and hard-coded values in the *RE2* code. We literally had to understand and reparse every single bit of data in the *RE2* library. And we did, much to the credit of every programmer on the team. This created quite a few bugs but we sacked them all.

Though the original *RE2* code was written in C, it resembled Assembly language more closely than it did structured code. Given that most of us are flat out reading English, the Japanese comments weren't particularly helpful, either.

We could have done a better job through the project of anticipating dependencies and eliminating them ahead of time. This simple ounce of prevention would invariably have saved a pound of cure. I had no complaints about the programming talent, but I think there is always room for better software practice. Case in point: our failure to utilize code reviews.

Despite these minor rants, *RE2* on the N64 was a great success. While we didn't hit our lofty goal of 2 million copies by January 2000, we did deliver a faithful rendition of a great game to the Nintendo 64, and even managed a few industry firsts and a couple of extras along the way. We managed to do this on time (almost) and under budget in spite of the large technical challenges thrown upon us, thanks largely to a great team, a detailed plan and schedule, and a lot of hard work!

Game Stats

Publisher:	Capcom
Number of full-time developers:	9

Number of contractors:	1
Budget:	$1,000,000
Length of development:	12 months
Release date:	November 16, 1999
Development hardware used:	SN Systems' SN64
Development software used:	Visual SlickEdit, gcc, Debabalizer, Photoshop, Softimage
Notable technologies:	Our proprietary N64 OS and FMV compression and playback system
Total lines of code:	approximately 200,000

Acknowledgements
Thanks to Jamie Briant, Chris Fodor, and Alex Ehrath for their input in this article.

When there's no surf to be found, Todd's busy pretending to be a software engineer at Angel Studios. Drop him a line at todd@angelstudios.com.

Return to the full version of this article