GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Raven Software's *Star Trek: Voyager -- Elite Force*

By james monroe,Michael Chang Gummelt,Brian Pelletier

In the summer of 1998, Activision had acquired licensing rights to make games using a number of *Star Trek* franchises. Their goals from the beginning were to create a broad selection of games and show the gaming community that Activision could take the *Star Trek* brand and make high-quality games with it, better than other publishers had in the past. The preliminary game slate was set with a first-person shooter as one of the initial titles.

Raven Software had been an external studio of Activision for a year, finishing up work on *Heretic 2* and diving deep into the development of *Soldier of Fortune*. *Heretic 2* was near completion, and we would soon need another project to work on. With our experience developing shooters and a reputation for making quality games, Activision handed the *Star Trek* first-person shooter project to us.

The game started out being based on an unknown *Star Trek* crew within the *Next Generation* franchise. For two months work was done on the plot and story line, with a test level of a Defiant-class ship made using the *Quake 2* engine. The main factor in designing the plot of the game was that it had to be an action game, despite the fact that *Star Trek* isn't known for action. To give meaning to the action, the idea for a Special Forces team soon emerged to drive the action for the game. Ultimately, because Activision already had two other games using the *Next Generation* license, the setting for our game changed to the *Voyager* franchise. Our excitement level was low at first, with the team feeling that *Voyager* was the least popular of all the *Star Trek* franchises. We soon realized that *Voyager's* plot allowed us not only to make our game with much more creative freedom, but also to create from something no one else had used. This inspired us to open the floodgates, continue on, and eventually realize that *Voyager* was the best setting for what we wanted to do. We quickly adapted the plot we had at that point into the *Voyager* setting. This was much easier than we thought it would be, and the Elite Force, or the Hazard Team as we called them, actually seemed to make more sense as a by-product of *Voyager's* situation. In January 1999, full production on *Elite Force* began with a small team of 15 people that would grow to about 25 core team members, with additional support from the *Soldier of Fortune* team.

***Voyager's* Hazard Team, created by Raven to help accentuate the action for the game.**

Our main focus during production was not to think about the game as a *Star Trek* product per se, but rather an action shooter that borrowed from the *Star Trek* universe. This helped us focus more on what would be fun for players. To our surprise, the Paramount approval process was much easier than we anticipated. We had heard many horror stories regarding Paramount's strictness with their licenses, things like, "You can't do anything new," and, "It's hard to get things approved because they're so protective of the license." What we experienced was the exact opposite. Paramount was more than accommodating in helping us create a fun game, and we were able to bend the rules a little along the way to help accomplish our goal. We created new Starfleet weapons, a *Voyager* SWAT team, used the Klingons, and even added "classic" *Star Trek* to the *Voyager* setting. As long as an element made sense to the story and its presence could be explained, it was no problem.

One of the biggest obstacles we had to overcome was that we would be making an action game that had to appeal to both the hardcore FPS player as well as the average *Star Trek* gamer and fan. This was no easy task, and we spent a lot of time debating over the game style being too much of an action game or more of a *Star Trek* game. Balancing these two aims was a constant battle during the course of production. We knew we had to walk a fine line blending a shooter and a *Star Trek* experience if we were going to both make a successful game and overcome people's perceptions that *Star Trek* games are not good games.

What Went Right

1. Improvements to the Quake 3 engine. Raven had worked with id Software's engines since 1992, but this was the first time we had to add a single-player game to an id engine. Normally, we had the luxury of starting with a full single-player code base and just adding things such as breakable brushes, new AI, navigation systems, and so on. But this time we had licensed a multiplayer game and had to put in many systems we took for granted. We needed AI and navigation appropriate for single-player enemies (not multiplayer bots), as well as teammate non-player characters (NPCs) and cinematics. We needed an expanded animation system for all the different animations our cinematics would require, we needed to create a load and save routine from scratch, and the list went on. One of the things that made this possible was the decision early on to separate the multiplayer and single-player executables. At this time, *Quake 3* was still about eight months from completion, so we started on single-player and would worry about multiplayer when we got the final code base. We were able to make drastic changes to the single-player game and shortcut the networking, allowing us to get away with a lot of things that would have just done very bad things to networking. With this new freedom, we revamped even more systems. In the end, we actually surpassed our initial ambitions as far as new systems and features were concerned.

For example, our Icarus scripting system was planned from the beginning and ended up working out very well. The initial setup was finished relatively quickly and the remainder of the work was mostly just tying the commands to the game and AI. However, for the first seven or eight months, only a couple of programmers were doing any scripting, as they were still refining the commands and there was no GUI for it yet. It wasn't until the fall of 1999 that we made a GUI and the designers could finally start scripting. The system ended up contributing a huge amount to the detail, uniqueness, and complexity of the game, and without it *Elite Force* would have been a totally different game.

Your teammates fighting alongside you, with Telsia blasting Etherians.

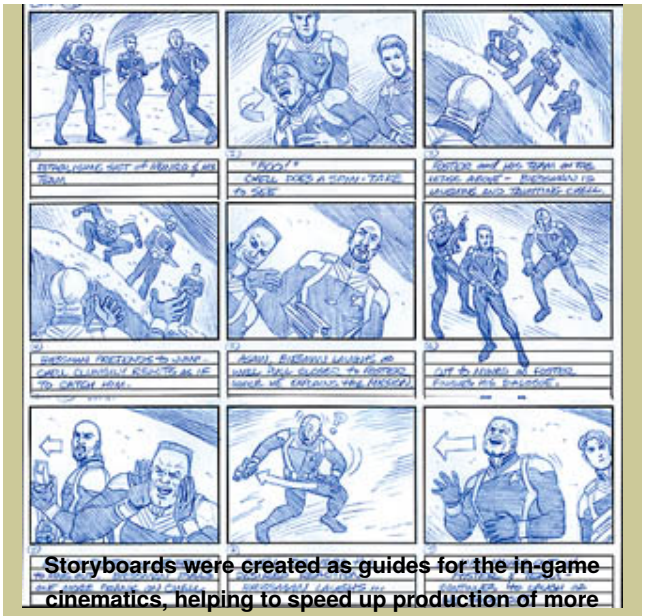
Another big technology decision we had to make was with Carcass, our new skeletal model format. It was a huge undertaking to switch over to the new format, but it really saved us in the end. At first we were using the same model format as *Quake 3*, but it quickly became apparent that we were surpassing that format's capabilities, so we looked for a solution. id had already laid the groundwork for a skeletal model, which seemed like it would work for us. Starting with that basis, we completed it and developed it into the final format we called Carcass. With it, we reduced tenfold the amount of memory a single model took up. Without the Carcass format, we would have had to cut back many animations, and we would have lost the complex detail in our cinematics.

Another technology that was successful was our lip-synch system, which really added realism to the facial animations. We did some research, looking into phoneme recognition, but finally settled on a quick volume analysis. We planned this system to make it very easy to use. Once the mouth animation art was made for each character, the system used the appropriate frame without intervention. The code automatically scanned for peak volume of sounds when loaded, and compared against that whenever a sound was played on the voice channel. Then the animation system picked up that value to choose the speaking frame. This setup required no extra effort when adding sounds, and would work automatically for any foreign languages used.

Another system we revamped was the cinematic system, which had to be powerful and flexible enough to give our cinematics that *Star Trek* "feel." The camera system itself wasn't that hard to implement, and it worked out well. First, the scripter/designer would set up the blocking of the NPCs through Icarus. Then they could go into the game and let the scripted event play out, pausing it whenever they wanted to save a camera position to a file that could be imported into the map. Using Icarus commands, they could make the camera zoom in and out, change the field of view to simulate close-ups and wide shots, move along a track, dynamically follow a subject, fade in and out, shake, and so on. This allowed us to set up our insane amount of cinematics as quickly as possible and still allow for some fine-tuning and detail (such as the "walk and talk" with Tuvok and Munro, and especially all the gestures and expressions the NPCs themselves would do to add to their characterization and the believability of a scene).

The AI of the Borg had them adapting to your weapons like their TV show counterparts.

2. Complete plot and story right from the start. From the beginning of production, *Elite Force* had a detailed story line, and every level of the game was written out in story form. We also had standards we had to meet; after all, our game was going to be compared to the *Voyager* TV show, so there was even more emphasis on storytelling. The story had to be engaging and reminiscent of what a *Voyager* episode would be, and we had to make sure our story had a lot of depth and interest for the player, to give them the feeling that they were partaking in an episode of the show. Since one of our main goals was to have an away-team accompany the player for the missions, it was even more important that the story be solidified up front. A lot of story is conveyed during the missions, so we had to make sure the levels were paced out well and the level designers knew up front what story content their levels contained. We were able to pace the story throughout the game so that players would be continually rewarded with exposition. As they completed more missions, the overarching plot of the game would slowly form in their minds.



Storyboards were created as guides for the in-game cinematics, helping to speed up production of more than 50 cinematic sequences. The storyboards also referenced the game's dialogue script, which was more than 400 pages long -- twice the length of a movie script.

[Expand Image]

With our tight schedule, we wouldn't have time to redo parts of the game if they didn't work out, so it was crucial for all the people involved to work together on the story line toward one common goal. With a complete walkthrough of the levels written, the level designers could concentrate on the looks of the level and accommodate for where the cinematics and story segments were going to take place. Because our story line never changed during production, we were able to proceed forward uninterrupted, and never had to scrap any levels due to plot restructure. This was key, as we had a fairly small team charged with creating a lot of content in a short period of time. The majority of the dialogue was written after all the levels were finished, but this too went smoothly because the walkthroughs were updated from the finalized levels, and then the dialogue was written from them.

---

3. The dialogue. The team was excited about the concept of playing with a team of NPCs in an FPS. This led to the definition of the different characters of the Hazard Team early on. However, the actual script for the game didn't begin right away, since that had to wait for the final game flow design to be finished. Our writer (also one of our programmers) started in September 1999 and finished the first draft in March 2000.

While he was writing the script, we were making all the cinematics and needed some temporary voices. We had employees record the lines and then dropped them into the cinematics to give us a feel for the pacing of each scene. This allowed us to tweak the dialogue while saving us from having to bring the expensive *Voyager* cast back for pickup lines.

The script finally came together after many revisions and, once it was approved by Paramount, the actors were lined up very quickly and the voice recording was done in about a month. We were then able to put the final lines into each scene, replacing our temporary dialogue without having to adjust the timing or change the scripts. This was due to the fact that Icarus could pause execution of a script until a sound file finished playing, so dropping in a new line of dialogue automatically adjusted the timing of each scene. This also meant that lines would generally flow better in translation, since they wouldn't have to deliver the line too quickly or slowly to match any hard-coded timing.

We also had a system for automatically reading the dialogue script itself and turning it into .PRE files that would let the game precache all the dialogue for a level and simultaneously assign the caption text to it. Included in this was automatic localization and dialogue adjustment for the player's gender. All of these things together enabled our game to have captioning, localization, gender-specific dialogue, precaching, lip-synching, and almost no pickups.

In the end, the dialogue turned out very well and our performances were good (Paramount even let us make final casting decisions on all the *Elite Force*-specific characters). The story and dialogue added a great deal to the game and contributed heavily to the feeling of actually being in an episode of *Star Trek*.

4. Re-creating the look of the Star Trek universe. Raven has always tried to push graphics boundaries and painstakingly create beautiful settings for our game worlds. *Star Trek* was no exception, and challenged us not only to create a beautiful gaming environment but also to create it in a likeness that is known worldwide. It's one thing to make arbitrary-looking levels in a never-before-seen world, but when trying to re-create the look of *Voyager* we came across many difficulties that we hadn't expected. For starters, the *Quake 3* level editor is made to create levels at a fairly big scale. When we built the bridge of *Voyager*, we were all astounded by the detail that we achieved, but when we put a normal-sized character on the bridge, he was incredibly tiny. The bridge was huge, yet it was built like you would build any normal *Quake 3* level. We realized that we had to build the levels on a much smaller scale than what we were used to. It took seven attempts at rebuilding *Voyager's* bridge until we attained the proper proportions between the characters and the level. We tweaked the scale until it looked perfect and the player and other characters could move around with ease. Once the scale of the levels was set as a standard, we continued forward with the other *Voyager* rooms with little rework needed.

Extensive photographic reference was used to get as close to the exact likeness as possible for *Voyager's* interior. Here, the photo reference of the bridge of *Voyager* is on the left and the game screenshot is on the right.

The artists spent much time working on the textures for the environments, getting their reference from many sources to make sure everything was exact. Having access to Paramount's *Star Trek* reference library was key in getting reference for carpets, chairs, upholstery, computer panels, and more. We sometimes scanned in the photo references themselves for the textures. Watching episodes of the show on tape was

also instrumental in determining what things looked like. We used a total of 1,033 textures to create the look of *Voyager's* rooms and hallways. Working together, the level designers and artists created the best-looking *Star Trek* environments in any game to date.

Just like the challenges we faced building the environments, creating *Voyager's* characters and crew presented the challenge of re-creating something that already existed. We used many references from the *Star Trek* reference library to help us re-create these real people. Each actor had a series of photos of him or her as their character, which we used to help get just the right shape of the head, and we even used the photos themselves (with Photoshop touch-ups) as textures on the polygon heads to achieve the likenesses. There are limitations to any technology, and working around the limitations is where we succeeded. The heads could only have 150 polygonal faces and the textures for the skins could only be 5123512 pixels. The fine craftsmanship required to work with so little and still achieve the right look for the characters is a testament to our artists' skills. For example, designing the Hazard Suit in the *Voyager* style took many attempts, but we finally got something everyone was happy with and looked natural to *Star Trek*.

5. Creating smart NPCs. To create a *Voyager* game that resembles what you would see in a TV episode, we had to create a working spaceship filled with its busy crew, and make it believable enough so players would feel like they are in a real place. We also had to create an away-team to fight alongside the player. After all, what is *Star Trek* without an away-team?

Exploding architecture helped in strategizing your attacks.

We created our NPCs using a few different things. The Icarus scripting system allowed us to have precise control over specific actions. The NPC Stats system allowed us to create many characters with various looks. The Squadmate system gave us the tools to enable the teammates to work with the player, and we used a waypoint or pathfinding system to make our NPCs navigate through a complex environment. All of these systems together created the artificial intelligence for our NPCs. We used scripting with the pathfinding to make the crew of *Voyager* come to life, and we could tell them exactly where to go and what to do without too many unknowns to cause problems. They did exactly what we as designers wanted them to do.

The teammates, on the other hand, have to act according to what is happening with the player. Since players can do anything they want while playing the game, this means a lot of unknowns for how the teammates should react. The teammates could have easily been a hindrance to the gameplay, and now we know why no other company has ever tried to make an FPS game with up to five teammates working alongside you throughout entire levels. In the final stages of developing the teammates, we weren't sure if they were going to work out; they had so many problems, and every time we would fix something another problem would crop up.

Just getting them to follow you was no easy task and was something we kept tweaking right up to the final days. Sure, we could get them to follow the player, but the game took place in tight hallways and small rooms so players would bump into them. They wouldn't get out of the way and would constantly get stuck on each other. Also, having them follow the player everywhere made them seem less like intelligent characters, so sometimes we had them stand their ground or take up a position while the player went exploring. Then we had the constant problem of the team not following players at all, even though they might need them later on. When we did get it working, someone found a new way to break a level with a teammate.

Elevators and teleporters added to the risk of teammates being left behind. We were getting worried that we wouldn't even be able to get them to walk through an entire level, and we would have to resort to something drastic. Luckily, we did get them to work in the levels. They may have run funny or jumped down long elevator shafts to catch up with the player, but at least they stayed in formation through the whole level no matter what the player was doing.

Of course, once there were enemies we had to worry about friendly fire. We wanted the team to react intelligently to being shot, but we didn't want to punish the player for shooting them accidentally. After a lot of trial and error, we decided that teammates would retaliate against a player only if the player had shot them repeatedly outside combat. In combat, they'd still react to friendly fire, but couldn't be killed, and would never turn on the player.

Save your teammate from the Borg, one of the many multiple outcome events.

Then came the problem of trying to balance the teammates' involvement in combat. Once we put enemies in the levels, we found that the teammates were so good that they killed most of them, leaving little for the player to do. To balance this, we had the teammates shoot less often, but then they got attacked constantly by enemies, turning the gameplay into "shoot the aliens attacking your teammates." Eventually we made the enemies attack the player more, so the player would feel threatened by them, and the teammates helped but didn't do all the work. It's funny now to hear people say that the teammates were stupid because they hardly killed any enemies, or that the enemies were dumb because they attacked the player more than the teammates. If they only knew how tiresome the gameplay would have been had we not balanced it the way we did.

What Went Wrong

1. Not enough programmers. For the first half of development, there was mainly only one programmer working on scripting, enemy AI, teammate AI, pathfinding, and the animation system. This programmer was also writing all the dialogue, and it became necessary for him to relinquish other programming duties in order to finish writing. Unfortunately, we didn't have any extra programmers to help.

Eventually we got a programmer from a different project to start working on game code. He completely rewrote the navigation system, which took time away from creating the AI for all the enemies, which didn't end up being completed until the game itself was done. This created a real lack of cooperation between level design and enemy AI, and forced the designers to rewrite their scripts constantly to match the changes in the underlying game systems. With more programmers working on AI and navigation early on in the development, these kinds of last-minute changes and back-end design could have (hopefully) been avoided.

The Klingons' AI allowed them to crouch and run for cover.

2. From Ghoul models to regular models to skeletal models. It was a big decision to switch to the new skeletal models. In the beginning, we were using what was to become *Soldier of Fortune's* Ghoul system (see "Postmortem:Raven Software's *Soldier of Fortune*" for more on Ghoul). When we received the *Quake 3* code, we tried to integrate Ghoul into the new code, but found it to be too different. It just didn't fit in with the new optimized rendering pipeline *Quake 3* provided. So we switched over to regular *Quake 3* models.

There was a lot of learning going on at this time. When we get new code, it doesn't come with operating instructions, and it's often not complete. We went through a lot of growing pains adopting the new format and figuring out its requirements. When the option to go skeletal came up, we had to weigh the benefits of the new system versus the risks and time it would take to switch over. While we did the right thing and embraced the new technology, we had to write a new set of tools to handle the new formats and learn new procedures to get our animations out of Softimage 3D and into 3D Studio Max. We had a lot of squashed creatures and bizarrely stretched limbs along the way, but

it was well worth the trip. Unfortunately, by the time we got it working right, we were past our alpha date, and because of all the different changes the models had gone through by that time, we didn't have enough time to fully implement a good AI system for the enemies, and had to settle with what we could do in the time we had.

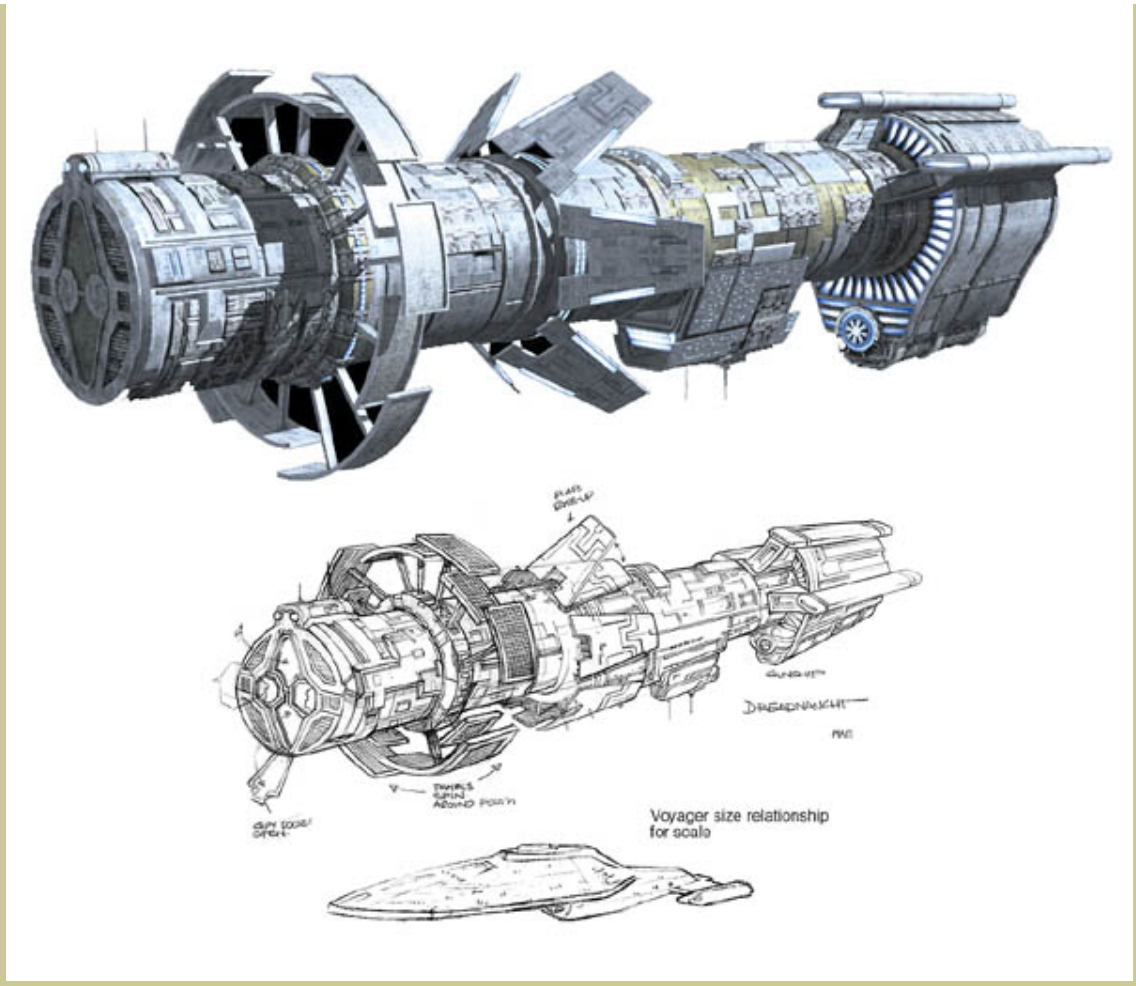
Other models were designed by Raven and made by the company making the prerendered movies.

3. Underestimating the amount of scripting work needed. As we mentioned previously, our Icarus scripting system was a huge plus. But we also encountered a lot of problems with scripting. Not only had no one ever used this system before, none of the programmers behind it had ever written a scripting system before. The designers didn't even get their hands on the scripting system until about eight months before the game was done. They did pick it up quickly, but not without a lot of effort and time involved.

One of the major problems designers had when scripting was the constant changes to the underlying systems that the Icarus commands relied upon. They'd script an NPC one way and it would work fine. Then the following week, something in the navigation, AI, or animation systems would change, and the script would be broken. This was a source of major frustration among the designers and definitely impeded their productivity. Ideally, those systems would have been finalized before the designers had to start scripting.

Within Icarus itself, there was one major flaw that should be addressed. Icarus can start a command and wait for completion, but it does not have a built-in system for letting the command (or "task") time-out and continue or take another route. There was no failsafe if, somehow, a command never completed. Given the sheer complexity of our scripting, these kinds of showstopper problems showed up constantly and would completely stop the game in its tracks. Up until the last minute, we were frantically trying to find every case in which a script would just stop execution. In the end, we did manage to catch them all, except for two cases that were caused by people turning their detail levels up too high and causing the game to drop to very low framerates, which could in turn mess up the scripted events. It turned out pretty well in the end, but all the effort that went into constantly revising the scripts could have been put to other, more productive uses.

4. Adjusting to new Quake 3 technology. The biggest level design headache in working with technology that is still being developed is that it's constantly changing. We started building our levels way before the *Quake 3* engine was near completion, and this caused scheduling problems every time we got a new code build. We built our levels one way with the tools and knowledge we had at the time, and then when a big change was made to the *Quake 3* code, the level designers had to spend a few days altering each of their levels to keep up with the changes in the code and how it handled surfaces, lights, and architecture. This happened numerous times during development, and we often went months without new code drops. The level designers would continue to work on levels in order to make progress, and then when we finally received new code, we had to go back to all the levels that had been done and spend a month getting them up to date. This month was not accounted for in the schedule, and therefore a month of designer time was simply lost. This happened more than once and was a big factor in keeping us behind our original schedule.



It was important to make highly detailed sketches, since another company was making the models from them.

Another part of adjusting to the *Quake 3* technology was realizing that our levels couldn't be as big as *Quake 2* levels. The wonderful thing about the *Quake 3* engine is that you can have many more polygons in the view at one time, allowing for more detailed levels and rooms, and of course curves. Without this engine we wouldn't have been able to create such accurate-looking *Voyager* rooms and locations.

When we started building levels, we made them as we did using *Quake 1* and *2* technology. We had expansive levels that looked awesome and showed off what the *Quake 3* engine could do. Then, somewhere near the middle of our development, we realized that the file size of most of our levels was huge, running 11 to 15MB each, when they should have been about 6MB. This was a problem, since we'd planned for the file sizes to be 10MB or less out of a total memory budget of 64MB. The levels were the normal game-world size of a *Quake 2* level, so what was the problem?

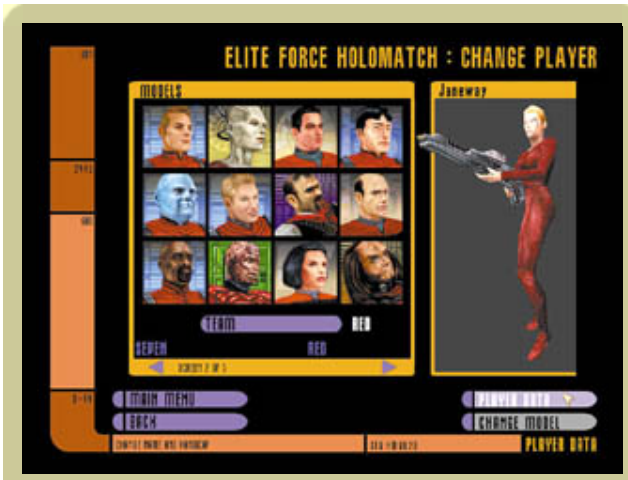
It turned out to be the high polygon (or triangle) count used to create a much more intricate and detailed environment. We realized that although *Quake 3* can handle more polygons in the view at one time, the file size for the level had not increased much from a *Quake 2* level. We had a dilemma; either we could bring the file size down by taking out all of the detail that made the *Quake 3* engine superior and keep the physical size, or we could cut the size of the level down, making it smaller yet highly detailed. Since we were making a world that could readily be compared to a TV show, we opted to keep the detailed environments of the *Star Trek* universe and cut the level size down.

Fighting aliens in the stasis ship, which was created with 90 percent curved surfaces.

We were able to cut most of the levels in half and make two separate levels out of them, but then all the level designers had twice as many levels to work on, and this could have caused some major scheduling problems. Unfortunately, to keep up with the schedule, large parts of the levels were deleted and redesigned, which resulted in much smaller levels that could be traversed quicker, and ultimately made for a shorter game.

5. Mission stats never got finished. The only major thing that didn't get into the game that we originally planned for was our end-mission statistics. The feature made it into the game in the form of basic stats when you died, but it was planned to be much more, and would have really added to the game. The end-mission stats would have improved the replayability of the single-player game and given more emphasis to the interactive and multiple-outcome events. The main goal with the stats was to grade players' performance when they completed a mission; for example, giving a score of 100 percent for a perfect mission. A number of medals would also be awarded to players based on what they did during the mission. At the end of the game, all the scores and medals would be added up for one final game score.

The stats would have made a significant gameplay feature, letting players know at the end of a mission whether they could have saved someone or done something differently to get a better score. This would have made the interactive events mean something more, emphasizing the fact that they are interactive; events that players didn't even know could be changed would have been presented to them after the mission, making the game world seem that much more alive. With this feature not in the game, the multiple outcome events didn't mean anything more than just a "cool" factor, instead of being intrinsic to the gameplay and adding to replayability.



Menu screen for choosing your player character in Holomatch gameplay.

Final Thoughts

We started out with a lot of great ideas, and almost all of those ideas were implemented in the game with the exception of a couple. That's certainly an achievement in this industry.
We made the game we set out to make and are very happy with the end result, so it was hard to think of the things that went wrong. Even though we had many problems, we worked around them and ultimately finished the game, which makes us feel like we did everything right. Then we heard comments from fans and game reviewers who didn't like certain things about the game, and all the memories of working through all the obstacles came back, and we thought, "If they only knew how many problems we had to work through to get the game to its final state." It's working through those development obstacles that makes a game successful. It's also gratifying to hear from fans and reviewers that the game was successful both as a fun game and as a *Star Trek* game. For us, that means many more things went right than wrong, and the team was talented enough to work through the things that went wrong and make a game that is being enjoyed by thousands of people.

Elite Force was a very difficult project cycle with a really long crunch mode, but what game is any different? Yet we had a lot of extra obstacles to overcome, including the perception that all *Star Trek* games suck and that meant ours would too. It seemed like we were destined to fail before we even started. Working with one of the two biggest science-fiction franchises in the world added to the pressure. But Activision supported us all the way from upper management to a top-notch marketing and PR staff. Paramount was surprisingly helpful, and proved to us that they care about the quality of the games made and would do everything possible to ensure that quality. With a dedicated and very talented team of individuals, we met our challenges and succeeded in making what is being called the best *Star Trek* game ever made.

The *Voyager* 3D model used in prerendered cinematics is the actual one used for the TV show.

Game Data

Raven Software
Star Trek: Voyager—Elite Force

Project length: six months preproduction, 18 months production

Number of full-time developers: 20

Number of contractors: 13, including two prerendered animations studios, additional musician, voice director, casting director, and eight main *Voyager* actors

Project Size: Single-player and Holomatch: 919,749 lines of code; 1,679 files. The single-player game was largel controlled by scripting, totalling 112,056 lines of code and 2,236 files.

Budget: Multi-million dollar budget.

Release date: September 20, 2000

Intended Platforms: Windows 95/98/NT/2000, Macintosh, Linux, Playstation 2

Development hardware used: Average system: Dell Pentium II 550 with 128MB RAM, 18GB hard drive, GeForce 3D acceleration card, and 21-inch monitor.

Development software used: Microsoft Visual C++ 6.0, Microsoft Visual SourceSafe 6.0, Borland JBuilder 3.5, 3D Studio Max 2, Softimage 3D, Photoshop

Notable technologies: Licensed the *Quake 3: Arena* engine from id Software (using OpenGL); Icarus scripting system, BehaveEd scripting tool, Carcass skeletal system, Bink, and motion-capture data from House of Moves

Return to the full version of this article