GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Multitude's *Fireteam*

By Art Min

Our goal with *Fireteam* was to create a complete online game experience. The Internet gives game designers the ability to take multiplayer gaming one step further by creating a community, something that wasn't possible before online games came about. Multitude wanted to take the next step in gaming evolution by making the community a significant part of our product. In other games, such as *Diablo* or *Quake*, the players were creating communities themselves, mostly through their own web sites. Multitude, on the other hand, devoted significant development time to creating tools that would help the community. We spent as much time on *Fireteam*'s lobby and community web pages as on the game engine itself. Our goal was to create a game that would make people say, "Wow, this is what I've wanted from an Internet game."

Fireteam is an online-only gaming experience. The actual game play is a squad-based tactical combat. Players can communicate with other members of their team using Multitude's voice technology. Each player controls one character in the battlefield. The game uses an isometric, three-quarter view 2D graphics engine. There are four different *Fireteam* scenarios, and each game session is ten minutes long. The scenarios are very sports-like in their design to help promote team play. Equally important to the *Fireteam* experience is the lobby, where players can view other players' statistics, chat between games, and find squad mates and enemies for their games. The last component of *Fireteam* is the community web pages, which display players' complete statistics and provide support for *Fireteam* Companies (which are similar to *Quake* Clans). On the community web pages, players can create companies, add/kick members, and access private Company bulletin boards.

Brief History

Fireteam evolved dramatically over its first year of development. Multitude was originally founded to create the "ultimate online game," which was to be a large persistent science fiction world. We knew that there would be some competition because *Ultima Online* had already been announced, although Origin hadn't yet performed any alpha or beta testing. We spent months writing and planning for a massively multiplayer online game set in a futuristic world. The project was to have a server team of around 10 people and a game team approximately double that size. As we were designing around our original concept for the game, our desire to make a persistent-world game work as well as a single-player game presented us with many hard technical and design problems. On the good side, it was during this process that we finalized the design spec for our combat engine. The combat engine was inspired by *X-Com: UFO Defense*, emphasizing squad combat with features such as line-of-sight.

We soon realized that our new company's financing was coming along very slowly and that we needed a much more easily attainable goal (due to lack of resources, both financial and human) that would still showcase the unique voice technology that we'd developed. We looked at the combat engine specification, our voice technology, and the Internet technology that we were designing and realized that we could make a great tactical team game. So at that point, we decided to abandon the large persistent world and make team play the essence of the game.

Designing a multiplayer game is very different from designing a single-player game. I've heard that in many games, the multiplayer component was added on only because marketing had requested the feature; this approach can make the multiplayer experience less than ideal. In a single-player game, the player is the hero and the focus of the game experience. The player should be able to win 100 percent of the time (with some effort). In a multiplayer game, a player should win 50 percent of his or her games against an equivalently skilled player. The thrill of a multiplayer game shouldn't be in the winning, but more in the process and the actual competition. Team play gives players a deep gaming experience, even if they lose. Our efforts to create engaging multiplayer game play were made even more effective by our voice technology, which allowed players to hear the emotions of their fellow players.

***Fireteam* was designed to emphasize community and teamwork**

***Fireteam*'s Components**

Fireteam's network architecture is client/server-based. We chose a client/server architecture because of the benefits that it offered us in the areas of performance (especially with the voice technology), cheat prevention, and centrally located statistics. The clients all run on Windows 95/98 and the servers run on Windows NT boxes, where we use Microsoft Chat services to do the intercommunication between our server processes. We also have a Microsoft web server running the community web pages, with a Microsoft SQL server maintaining the database. Our servers are at one location, our ISP Globalcenter, in Sunnyvale, California.

Fireteam uses the Elemedia SX2.0 Voice Codec to do its voice compression and decompression. Multitude's proprietary software wraps around this voice codec and interfaces with the Windows sound system for both input and output. The game mixes multiple voices on the client side rather than the server side. Clients simply send voice packets to the server, the server then routes them on to the appropriate teammates. In the future, spectators or enemies will be able to listen in on the voice chatter. Our voice software handles both DirectSound and non-DirectSound drivers because some sound cards work with DirectSound in full duplex. Full duplex means recording from microphone and playing sound at the same time.

Who Worked on *Fireteam*

Ned Lerner and I started Multitude and began working on the original project in April 1996. The development team grew gradually over the course of the project. Jim Morris was brought on during the summer of 1996 to be the chief technical officer, and his first project was to develop the voice technology. Alan Murphy was brought on to provide art for the prototype and eventually was named art director. Conroy Lee, Harvey Smith, and Harry Schaffer were brought on in early 1997 to help take *Fireteam* from a prototype to the real game that we showed off at E3 1996. Bill Money, James Poelke, and David Reese came on in late 1997. The team has a very diverse group of products to its collective credit. Lerner and Morris were two of the first people to work on 3D in the game industry. Murphy's art credits include *Galaxian, Pac-Man, Defender, Taz,* and *X-Men*. The others have worked on games such as*System Shock, Terra Nova, Magic School Bus, Ultima VIII*, and *Front Page Sports: Baseball.*

What Went Right

1. Combining team play and voice together. *Fireteam*'s design focus was on team play. Just as we've seen in team-oriented sports, the cooperative nature of playing as a member of a team has proven to be a very addicting and powerful gaming design. *Fireteam*'s cooperative nature was a symbiosis of our voice technology and team play design. We needed to give people a reason to talk to strangers on the Internet. Team play was that reason; it gives people the ability to say "Watch out behind you!" or "Good job!" Teammates can share the joys of victory or the agonies of defeat. Because there is no button to push to transmit your voice (it transmits automatically when you talk), players can hear the spontaneity of teammates yelling and laughing. Emotion comes across very clearly with voice and is definitely preferable to typing in ALL CAPS or emoticons. The ease of vocal interaction brings the team together.

In a fast-paced tactical game such as *Fireteam*, players don't have time to coordinate movements with the keyboard. Without voice, you limit team communication to select macro keys (or players who can type very fast). In *Fireteam*'s Basetag scenario, for example, teammates protecting the base can give instant information on where the enemy is making its attack. Over the course of their lives, people have already learned how to talk; it's an interface they understand. Vocal communication doesn't require a key card list for communication hotkeys, just a microphone to talk into.

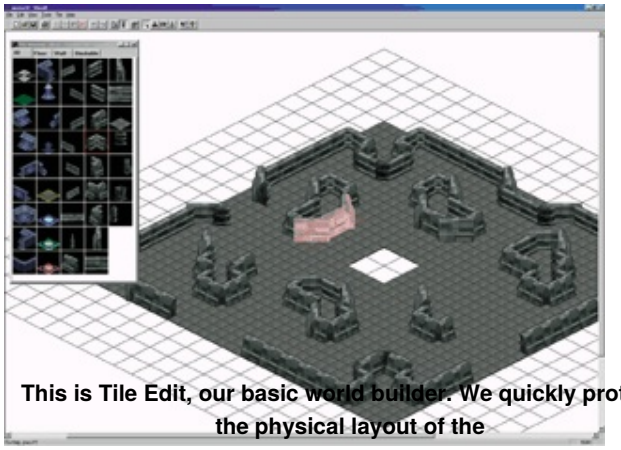
One of *Fireteam*'s successes was its integration of voice communication

2. Designing the project around constraints. Multitude was founded to do a game for the Internet. The Internet offers many problems that we had to solve in order to make a fun game. The biggest technical problem was Internet latency. Fundamentally, latency causes each player to have something different on his or her screen, so there is always a delay between a player performing an action and the other players seeing that action carried out.

For example, we decided early on that we wanted the game to respond as quickly as possible. When you shoot at something during a *Fireteam* game, you'll see instantly whether or not you hit your target. The actual damage will take a small amount of time to be applied to the target. So it's possible that you'll see someone get shot, walk a bit, and then die. The game cannot provide a perfect view of the world to each player; that's not possible given the limitations of the Internet. So we decided not to show players their opponents' health. If you can see that your opponent has only a sliver of health and that one shot could kill him or her, then you would expect that player to die instantly with the next shot you made. By hiding opponents health from our players, we hide the perception of lag.

A project's constraints can also be exploited to the project's benefits. Because the constraint was that we needed to be on the Internet, we created the community web pages to give players the ability to look at their statistics and create *Fireteam* Companies. We wanted a strong community for *Fireteam*, and the community web pages were an easy way for people to have an identity in this community and to join a group of other players.

3. Spending sufficient time to develop tools. We used several proprietary tools to create *Fireteam*'s game environments. Early on, we spent a lot of time building easy-to-use tools that allowed us to create content rapidly. Our internal testers used these tools to create new arenas for *Fireteam*. And because *Fireteam* is an online game, we were able to test new maps very quickly with our beta testers.



This is Tile Edit, our basic world builder. We quickly prototype the physical layout of the maps with this tool. It's very easy to move walls around to achieve the right game balance.

With an online game, game balance is crucial. Players will find any competitive edge and map imbalance they can and exploit it. Especially in an online game with a lobby, word of cheats or advantages spreads very fast. Your tools must allow you to tweak your maps, so you can quickly fix any small problems. Many of our maps changed during the course of testing as our testers would point out weaknesses that they found.

I can recall a particular controversy over whether Gunball (a *Fireteam* scenario similar to combat football) was balanced enough. Many of the advanced players were complaining that Gunball's offense was too hard. Using our Tile Edit tool, we quickly created a few maps with two endzones for each team (Gunball maps normally only have one endzone). Through testing the new maps, we discovered some of the problems with Gunball were unrelated to the maps themselves, but that the offense simply had a disadvantage when trying to score. So instead of redoing all of our map designs, we tuned the Gunball game by giving the Gunball carrier a protective drone.

We take the output from 3D Studio Max (with our plug-ins) and, with our proprietary ZHMPView tool, convert the files to a format that *Fireteam* can read.

An added bonus of easy-to-use tools is that you can make them available to the public and let your players customize the game and create their own content. We haven't yet taken that last step, because we're not sure how we want to store the maps on our servers and present them to the community.

4.Managing risk: voice technology. The biggest risk in developing *Fireteam* has been the voice technology. Many smart people initially said it was impossible, but we knew that the game's design objective was a cooperative team game, and voice was very important to accomplishing the goal. So *Fireteam*'s first technical project was to determine whether or not voice on the Internet was even possible. Once we had the technology running over the Internet, we still faced the possibility that it wouldn't work with the wide spectrum of sound cards in the market.

We tried to minimize the problems that voice would cause by providing users with a tool that would configure the sound card and microphone during installation. Multitude was very aware (almost scared) of the fact that *Fireteam* would represent most users' first use of voice technology on their computers. So we had to make sure that it worked on as many sound cards as possible and that it was very easy to use. We eventually released *Fireteam* as two executables — one for systems with DirectSound and one for systems without it.

One feature that we explicitly did not put into the game was the ability for players to talk to their opponents. We wanted players' first experience with voice to be a positive one. We didn't think a 12-year old telling you where to put your gun in his shrieking voice would convince people that voice is a wonderful addition to gaming. Similarly, people asked for the ability to eavesdrop or steal another team's radio

and listen to the other team. This feature would impel team members not to talk if they believed they were being monitored. These types of behavior would weaken the voice feature.

5. Promoting community. If you're going to design an online game, you cannot ignore the community. Any online game, from *Fireteam* to Poker on AOL to *Ultima Online*, will have a community because the players will be able to communicate with each other. Online game developers should take advantage the fact that their product inherently has a community. Most online games go through alpha and beta online tests mostly to test the software, but few deliberately create or test the community aspects of a product. Players are not only a source of revenue for a project, but they are a feature of your game. In an online environment, the players' game experiences are dictated by their teammates and the opponents against whom they play. You want the players to follow guidelines and really care about the game and the community. If your population is full of a bunch of player killers, then that's the experience that the players will get.

Multitude succeeded in developing community-enabling tools. We spent significant time and discussion on our lobby and community web pages. Given *Fireteam*'s team nature, we wanted players to feel a sense of belonging so that they would want to save each other's lives. The *Fireteam* product is not just the game itself. The game is an important piece of the *Fireteam* experience, but it's only a piece. The community plays a large part of the whole experience.

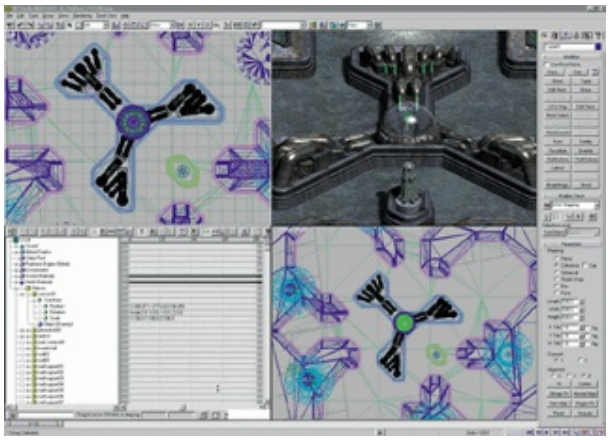
What Went Wrong

1. Misjudging market conditions. When Multitude was founded in April 1996, there was a lot of buzz in the online game space. Mpath and Ten had big plans. *Ultima Online* was about to go through testing. We believed that an online-only game, sold directly to customers via the Internet, would be acceptable to the market when we eventually shipped. What we've discovered is that the online game market has not matured to the level that we expected. Very few online-only games have been released, with *Ultima Online* being the only clear success. We made two decisions early on that should have been reexamined when it became clear that customer acceptance of an online-only game was not a forgone conclusion.

Fireteam would have been more willingly accepted by the market if it contained some artificial intelligence (AI). With such an implementation, players could practice using the interface by themselves and, more importantly, players could practice as a team against AIs. With computer-controlled opponents in place, players could play offline or possibly on a LAN against computers opponents. Many users are intimidated by having to learn a new game while playing against other more experienced human players. AIs would have helped ease players into the online-only part of the game, providing a feature that many players expect to find in games today.

Fireteam also should have had a demo available on day one. As an online game, providing a demo presented an interesting problem because of the server issues. With a traditional game, developers can hand out a million demo disks and never think about the problems their users might experience. If we gave out a million demo disks, then we would need to have enough servers to support all those people that actually play the demo. We didn't create an infrastructure to support a demo mode of *Fireteam*. *Fireteam* is a new type of game — people aren't yet accustomed to online tactical team games with voice technology. We should have made some extra promotional effort to get potential users to make that initial leap and try out the game.

2. Managing contractors. *Fireteam* was a large and extremely challenging project. We had to look outside of our own company for help with certain parts of the project. Our mistake was in assuming that these experts held the same priorities as the rest of the development team. These groups have their own objectives and aren't expected to understand the big picture or know how to create fun games. We realized that when working with contractors, we needed to give those contractors a very precise and clear specification. Because a good game design evolves over the course of a project, a project manager must constantly make certain that the contractors are following the latest version of the specification.



A 3D Studio Max layout of one of the maps. Our artists take the game-tested layout from Tile Edit to create backgrounds for each map.

As we were developing *Fireteam*, our attention was also focused on growing our new company. We found that it was easy to forget what the

contractors were doing and how they fit into the project. We made the naïve assumption that they would be willing to work with an evolving specification. However, when a project is fix-bid, an external developer will only do so much tuning and reworking of code before he or she starts charging you for it. If you don't manage this relationship closely, these costs add up very quickly. For example, the cost for developing the community web pages doubled from the original quote because the design evolved. The final version of the community web pages was great, but a more thoughtful initial design specification and better management of the process would have saved Multitude significant money and time.

3. Internet technical issues. The Internet poses significant problems for developers. Although we did our best in designing the game around the limitations of the Internet, we did have some technical problems. We originally designed *Fireteam* around TCP/IP because it's a reliable transport protocol for network traffic. However, the reliability comes at a very high cost: retransmission times. If a packet is lost on the Internet (which happens a lot), it takes some time for the machines on both ends to realize this and resend the data. TCP/IP guarantees that all packets are in order; therefore, all of the packets after the lost packet will be delayed until the lost packet is sent again. In a fast-paced game such as *Fireteam*, lost packets can really cause problems. As soon as we started doing real Internet tests, we realized that we needed to start sending some packets unreliably via UDP. These packets could get lost, be out of order, or even duplicated, but they wouldn't be delayed by other packets. We learned that different packets require different sets of reliability and timeliness, and that developers should use all the tools available to them, both TCP/IP and UDP. We initially labored under the idea that only one protocol should be used for the sake of simplicity, but it's best to use the appropriate tool for each job.

Packet loss and high ping times are simply part of the reality of dealing with the Internet. You do your best to deal with these issues, but they'll still cause you endless headaches as routers over which you have no control go down throughout the country. Many online games come with a little utility that does a trace on the route the packets take between a player's machine and the servers. The information that the utility returns can help the player and his or her ISP determine where the bad connection is along that route. Developers should be aware that while they cannot fix the Internet infrastructure, it's important to understand its limitations and deal with them as best they can.

4. Server spaghetti. *Fireteam* is a very complicated project with many processes running on both the client and the server. Add in the complication of the Internet, and you can get one confusing mess (Figure 1 shows just how complicated the *Fireteam* architecture is). We tried to break our server components down into smaller, more manageable pieces, each with its own function. We hired some experts in various disciplines to help us better understand parts of the server technology that were new to us. Our mistake was in thinking that these experts could just come in and solve our problems. As we busied ourselves with other parts of the project, it was easy for us to say to ourselves, "They know what they're doing." In the end, however, the development team needs to understand the whole picture and how the pieces really fit together. One of *Fireteam*'s unique properties is that its server-side components run remotely at an ISP's facilities. In order to debug something as complicated as our server architecture remotely, our key programmers — not just the client/server experts —needed to understand the whole system.

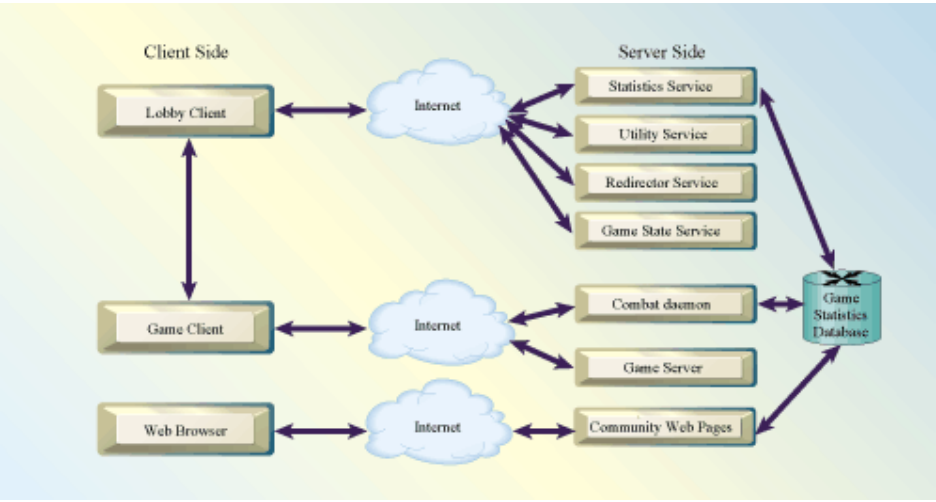


Figure 1. *Fireteam*' s technical architecture

One or two weeks spent planning and discussing the entire project with everyone involved will save you months down the road. The process of actually finishing and shipping a game is the hardest part of the development cycle; not many people actually know how to ship a game. During the final stage of the project, it's essential that the entire team understand all the pieces of the puzzle.

5. Coping with the community. As I mentioned previously, when you create an online game, you need to embrace the community. At the same time, a direct connection with a community of testers who aren't 100 percent aware of your objectives is something that needs to be managed very carefully. The testers will always want something different. When is the last time you played a game and said, "This is perfect"? I've often said that even my favorite game would be better if it had feature X. Most beta testers are young people who have a lot of time on their hands; that's great for finding bugs, but it can also be a problem because some of them lack perspective. All players have an equal voice in the *Fireteam* lobby, so we had to watch over the lobby constantly because a few testers could ruin the fun for others, even to

the point of instigating a mini online riot.

The home page for the *Fireteam* community web pages. Players can access a wealth of information about their statistics or other players' statistics.

From what I can tell, some online game companies simply ignore their testers' constant demands. After the experience of developing *Fireteam*, I must admit that this is a possible solution, though not an optimal one. Many of us on the development team spent many hours justifying our design decisions in order to educate the testers on why we were doing things a certain way. While this education does make them better testers, it takes up a lot of time. And it's a dangerous black hole that you can be sucked into if you're not careful. I believe that the true balance is to pay attention to your community, but sometimes to sacrifice the battle in order to win the war. You should involve your intended community in the evolution of your game, but don't let it take over your design process or time.

Evolving Right Along

In building *Fireteam*, we as developers accomplished our goal of providing a complete online gaming experience with true team play, innovative voice technology, and extensive community building tools. The Internet offers brand new gaming experiences; game players can compete in ladders such as Battle.net or tournaments such as the PGL. Also, an online game lets players meet new friends with whom they can share true social gaming experiences.

However, the Internet introduces a lot of negatives to the gaming experience. Instead of lightning-fast LAN connections, players must now tolerate latency. Instead of a small group of friends, a player's opponents may be complete strangers who aren't polite and may even be cheaters. Because to the newness of this market, *Fireteam* may be ahead of its time. Or it may not have exactly hit the sweet spot that online multiplayer gaming should be. But, *Fireteam* has helped online game evolution along by demonstrating that voice technology does work and that team play and community are compelling elements that don't have to be accidental.

Art Min is *Fireteam*'s project leader and the cofounder of Multitude. He lives in the San Francisco Bay Area. Art relaxes by hitting a little black object on a cold surface and attempting to learn how to paint, though not at the same time (yet). He can be reached at minman@multitude.com or minman@alum.mit.edu.

Fireteam

Multitude Inc.
Burlingame, California
(650) 685-2001

http://www.multitude.com or http://www.*Fireteam*.com

Team Size: 14 full-time developers. Some number of contractors.

Release Date: December 1998

Target Platform: Windows 95/98

Budget: Approximately $2.5 million

Time in Development: Two and a half years

Tools: Microsoft Developer Studio 5.0, Microsoft SQL Server, Microsoft IIS, 3D Studio Max, Microsoft Interdev 6.0, Microsoft Chat Service, and Windows NT