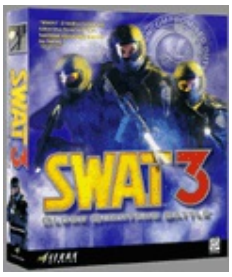
GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Sierra's *SWAT3 Close Quarters Battle*

By Jim Napier

SWAT3: Close Quarters Battle is a first-person tactical simulation where the player assumes the role of element leader in a five-man SWAT entry team. SWAT stands for “Special Weapons and Tactics”. In *SWAT3*, you are responsible for serving high-risk arrest warrants, rescuing hostages, neutralizing (killing or arresting) terrorists, and defusing bombs and other weapons.



Our goal was to recreate the SWAT experience. We didn’t want to simply create another first-person shooter. In fact, it seemed like everyone was coming out with a first-person shooter and we were concerned about getting lost in the crowd. Instead of shooting everything that moves, we wanted to create a simulation where winning a mission meant following proper SWAT tactics: using proper room clearing techniques, ordering suspects to drop their weapons, and, yes, using deadly force when necessary. But we also wanted it to be fun, and choosing between fun and realism turned out to be quite a challenge.

Officially, *SWAT3* is the sequel to *SWAT2*, but *SWAT2* is a 2D RTS and was developed by an entirely different team. Both our producer (Rod Fung) and designer (Tammy Dargan) worked on the original *SWAT*, which was based on full motion video. While *SWAT* and *SWAT2* received only marginal reviews, they sold like crazy. *SWAT* alone has sold over a million units to date. The name recognition has been great for sales, and Rod and Tammy’s *SWAT* experience and enthusiasm helped keep the team focused. While working on the SWAT and Police Quest series, Rod and Tammy developed several contacts within the LAPD SWAT community, including Police Chief Daryl Gates and a SWAT element leader named Ken, our primary consultant.

Early in the development cycle Ken put on a one-day presentation on SWAT procedures and tactics. He brought some videos of his more memorable missions, including the 1996 Hollywood bank robbery. It was like a scene from *The Matrix*: two guys (Ken calls them “knuckleheads”) with body armor and automatic rifles walking down the street shooting at everyone in sight. At one point, they even start shooting at the helicopter cameraman.

Ken’s dynamic entry missions were especially amazing: SWAT officers would blow open a suspect’s front door, throw in some flashbangs (extremely loud and bright explosive devices), and charge in yelling for compliance -- not something you’d want to be on the receiving end of. We realized that if we could recreate that experience, along with the other things we wanted to do, then we’d have a winner.

Midway during development, several titles were released that contained elements of *SWAT3*, the foremost being Red Storm’s *Rainbow Six*. That fact that *Rainbow Six* was so well received really excited us, because it confirmed what we believed all along: a lot of people were ready for a tactical simulation that went way beyond run-and-gun. This helped energize the development team and gave us someone to compete against. In the end, though, it’s unfortunate that we’re so often compared to *Rainbow Six*, because at the core they’re very different games. People who like tactical simulations should enjoy them both.

Most of the code for *SWAT3* was written from scratch. The character animation system, art import tools, client/server architecture, AI, and most of the graphics engine were all developed over the eighteen month development cycle. A couple of people on our team developed a prototype using *Quake*, but our project had so many new features that we decided we’d be better off developing our own engine. That decision really paid off: between the graphics engine and some amazing level designers, *SWAT3* has some of the most realistic environments ever seen in a game. That said, I’m not sure if writing your own engine would be a good decision today. Besides being a tremendous amount of work, it’s extremely difficult to develop gameplay elements until the engine and art process have stabilized. The programmers can only estimate the memory, texture, and polygon limitations, and artists can’t really lock down their levels until late in the project. Having a finished engine helps you focus on what’s really important: gameplay.

We started programming in May, ’97 and planned to ship for Christmas ’99. Management was, um, very insistent on meeting the Christmas deadline, so we had to build the best game we could in the available time. We originally planned to ship both single and multiplayer gameplay, but later decided to focus on making a great single player experience and shipping multiplayer at a later time. There just wasn’t enough time to build a killer single player game *and* multiplayer that would compete with *Quake3*, *Unreal Tournament*, and *Rogue Spear*, all of which were coming out at the same time. It turns out we made the right decision: *SWAT3* stands out for its compelling single player gameplay. Not

having multiplayer has hurt our reviews a bit, but we're convinced they would have been much worse otherwise.

What Went Right

1. Focus on Core Gameplay

Conceptual sketches for *SWAT3*

Development was basically broken into two phases. The first eight months were spent developing the graphics engine, application architecture, and assorted tools. We then took a step back and put together a rough project schedule that covered the remaining ten months. At that point we confirmed what we suspected all along: our design was too ambitious, and we were going to miss our deadline by several months. With our set-in-concrete-change-it-and-die ship date, we had no choice but to reduce the feature set and focus on core gameplay. This resulted in some difficult late-night sessions between our team and upper management, as we negotiated which features could be cut and still have a viable game. We ended up breaking features into three categories: required, optional, and later version, with the optional features prioritized by importance to gameplay. While the team was disappointed to let go of so many features, in the end we implemented most of the optional ones, and many others we hadn't considered at the time. Our tight schedule simply didn't allow for much feature creep, and our focus on core gameplay helped us ship on time.

Conceptual sketches for *SWAT3* hostiles .

2. Using Worldcraft

Early on, one of our level designers convinced us to use Worldcraft, the level editor used to create *Half-Life*. At first we planned to use 3D Studio Max, but it wasn't really designed as a level editor. Worldcraft was a much better choice: not only was it great for building and texturing levels, it also had a customizable entity system that allowed us to create our own objects (lights, doors, characters, etc.), each with their own unique set of properties. Our level designer was already familiar with it and could get started immediately. The file format was a bit weird (polygons are defined implicitly by a set of boundary planes), but it was text based and easy to figure out. Worldcraft's batch commands made it easy to save the current level, copy it to a new directory, run the import tool, and launch the game, all by clicking a single button. Level designers could create multiple commands for different tasks: quick build with no lighting, full build with lighting, etc. This made it easy for them to preview levels without exiting the editor. Worldcraft had a few annoying bugs and limitations, but we developed workarounds. This saved us a ton of time, especially when you consider that many teams write their own level editor.

3. Character Animation System

In *Gabriel Knight3* (another 3D game by Sierra), we used a vertex-based character animation system. This allowed for all kinds of effects, but had a serious drawback: changing a character mesh meant that all of the associated animations had to be re-exported. For *SWAT3*, we developed a skeletal animation system. All animations are applied to the skeleton, and the mesh (skin) is deformed to fit the skeleton at runtime. This allowed the character artist to work independently from the character animator, and we saved a lot of time and frustration as a result.

We used 3D Studio Max and Character Studio to create our character meshes and skeletal animations. We wrote two custom exporters: one for exporting meshes, and another for exporting animations. The animation exporter allowed the character animator and sound designer to trigger events in the game when certain frames are reached. This made it easy to add frame-based sound effects and other effects, such as footprints.

The animations themselves were motion captured, which had mixed results. All of the animations required a lot of hand tweaking, and it's not clear how much time we saved over simply creating the animations from scratch. Simply giving the character animator a movie of the desired

motion might have been just as effective, and would have cost a lot less.

Another important feature of the animation system was its ability to smoothly blend between animations. This eliminated most of the transition animations that would have otherwise been required. When a character is injured, for example, the animation system will smoothly blend from whatever their pose is to the first frame of the injury animation, play the animation, and then smoothly blend back to their previous state. We set up a simple transition table indicating how much time should be spent blending from one animation to another, and the results, while not always perfect, were much better than most games.

Character animation was achieved through motion capture and a skeletal animation system developed for *SWAT3.*

Note: I haven't evaluated it, but RAD Game Tools has recently released a character animation library called Granny that looks very similar to what we developed. It's definitely worth checking out.

4. Dedicated People

Many of the people on the team were experienced professionals who were dedicated to shipping the game. Our producer and designer were extremely enthusiastic and passionate about the game and its potential for success. Our designer had a strong vision but was also flexible about making design changes to fit the limitations of time and technology. Our producer supported our milestones and programming schedules, and backed up our decision to postpone multiplayer.

The art and programming staff got along great, which helped the development process go pretty smoothly. This isn't to say that everyone always got along, but the two groups as a whole worked really well together. This was especially important given the engine was being developed concurrent with level design. When an engine change required modifications to existing levels, the artists were supportive and would help test the new features. When an artist requested a change, one of the programmers would do their best to accommodate them.

The art director was especially helpful. He worked closely with the programming staff to define art formats, procedures, and naming conventions.He did an excellent job designing the user interface, and built a working prototype to demonstrate how the menu system should work. When the art started falling behind schedule, he took on the additional responsibility of building several levels, working a tremendous amount of overtime in the process.

Despite not really coming together until the last month of development, the AI in *SWAT3* has been rated as one of the best in the industry.

Our AI programmer took on the daunting task of modeling the behavior of SWAT officers and other characters, coming up with a true 3D pathing algorithm in the process. He was self directed and worked with very little supervision. While the AI didn't really come together until the last month of development, many reviewers rated it as one of the best in the industry.

Near the end of the project, our QA lead jumped in and helped us place character start points and AI path nodes. He came up to speed quickly and became an integral part of the development team. This was pretty typical of most people on the team – they were dedicated to the project and put in the huge amount of time and effort required to ship the game.

5. Frequent Milestones

At the start of an eighteen month project, it seems like you have all of the time in the world. But the days go by quickly -- all of a sudden the two weeks you estimated for that easy feature are gone, and you wonder if two additional weeks will be enough to finish it. Then two weeks later... you get the idea.Milestones are a great way to keep things on track, and we used them frequently.

The programming team scheduled milestones at roughly eight to twelve week intervals. Here's how it usually worked. First, we'd update the engineering document describing the current programming tasks. Then I'd meet with each of the programmers and we'd come up with a list of tasks that could be completed for the milestone. We'd consult with the art director and sound designer about the needed resources, and finally we'd present the milestone to the producer and designer for approval. Once the milestone was approved, we'd all get to work. A week or so before the milestone was due, we'd wrap up any last-minute integration tasks and put together a demo.

This process worked extremely well. We met most of our milestones, and on several occasions delivered much more than expected. The milestones were close enough to be tangible, but not so close that the overhead of putting together the demo was a burden. It was fun to bring in management and show them our progress, and seeing everyone's work integrated in the game helped keep the team excited. The demos were often packaged and put on a CD, so people outside the team could see the game. We lost very little time preparing a demo for E3 – it was simply an embellished version of our latest milestone.

What Went Wrong

1. Over-Ambitious Design

The original design was extremely ambitious. Most of the individual features seemed doable, but added up they represented a tremendous amount of work. All of us had extremely high expectations for the game, but the total feature set turned out to be unrealistic given our small development staff and fixed schedule. Looking at it another way, the expectation was to create a graphics engine similar to *Unreal*, with a better character animation system than *Half-Life*, with better AI than both games put together, and multiplayer, and compelling SWAT officer gameplay. Right. No problem. Some of us suspected the design was over ambitious. In design meetings, we would say something like, "um, this is getting a bit complicated here," but without a concrete schedule, our concerns were difficult to justify.

The levels for *SWAT3* were designed using Worldcraft

As mentioned previously, eight months into development we had most of our core technologies implemented and were able to develop a schedule for the remaining ten months. As suspected, we simply didn't have enough time to finish the game before Christmas, and had to cut back in several areas. This was painful for everyone because we felt the game wouldn't be nearly as fun without the missing features. As every game developer knows, for every item on a task list there are usually several items that are every bit as important but are not recognized until during development. These "hidden" items are the hardest part of the schedule to justify, and in our attempt to come up with a schedule that would result in the fewest cut features, much of our padding was sacrificed.

The tight schedule, combined with our ambitious design, high personal expectations, and staffing problems, resulted in some of us putting in significant amounts of overtime during the last ten months of development. This left very little time for integration and play testing, and only through the team's dedication did the product turn out as well as it did.

2. Staffing Problems

We had a difficult time finding the right people for both programming and art positions. Several people were transferred to our project after theirs had been cancelled or had shipped way past its deadline. Needless to say, this is a bad way to fill out a team. Some of them adapted well, but others were totally fried from the tremendous amount of overtime they had just put in, and had a difficult time getting motivated for a new project.

We had an especially hard time finding good programmers. We interviewed a lot of people, but our only external hire was the AI programmer, who turned out to be exceptional. The programmer responsible for the character animation export tools had just come from two cancelled projects and only lasted a few months. Another programmer was difficult to work with, refused to work overtime, and quit a couple of months before our ship date.

The art department also had difficulty finding the right people. Our lead level designer left during the middle of the project. That was quite a blow at the time, but his replacement was awesome and ended up creating some of the best levels in the game. One artist, who had just come from a project that was over a year late, refused to work overtime and had trouble getting along with the art director.

So, what could we have done better? We should have only hired and kept motivated people. Even those with less experience will contribute more in the long run than those who don't want to be there. We should have been much more aggressive about hiring the right people early in the development cycle. This would have helped us finish the project sooner and given us more time for integration and play testing.

3. Concurrent Engine/Art Development

Due to our tight schedule, we couldn't wait for the game engine to be completed before starting on the art. This was difficult for the artists, who were developing content for an engine that didn't exist yet. They used Worldcraft and *Quake* to prototype some of our early levels, but since our engine used a different rendering algorithm (cells & portals), they could only do rough work.

Even after the engine was up and running, it was constantly evolving. Our design called for levels with an incredible amount of detail, and the engine was constantly being optimized to increase the frame rate. These optimizations would sometimes require the level designers to change the way they worked, and existing levels had to be modified. Engine changes would often cause rendering artifacts, and it was difficult for them to tell whether the problem was with their art or the most recent engine change.

Due to time constraints the engine and art for *SWAT3* were developed concurrently.

The AI was also constantly evolving and frequently required existing levels to be changed. Things like valid door angles, minimum hall widths, and 3D path node placement were not finalized until late in the project.

From the start, we realized that concurrent engine & art development was going to be difficult, and it was. Fortunately our programmers and artists got along well, so this wasn't as much of a problem as it could have been.

4. Failure to Track Design Changes

Our design document was basically a framework for the game, not a detailed specification of how each feature worked or how they would be implemented. That was fine, except that as we filled in the details, we failed to put the resulting documents into a centralized location. We had

many impromptu meetings with the designer where we'd discuss how a feature would work and how it would be integrated into the game. Most of the time, the decisions made in these meetings were sent to the team via email. This worked okay initially, but as the project grew more complex and the number of design decisions increased, these "mini design documents" became increasingly difficult to keep track of. The current design was spread out among many separate documents and emails, and it was difficult to tell if the one you were using was the latest.

Design changes were frequently made due to programming or art limitations, and the reasons for each change were rarely documented. This was especially true at the beginning of the project. If the programming or art limitation was later overcome, then we were often hard-pressed to remember which features might be affected.

In the future, we should keep all design documents in a central location and keep them up to date. Features cut or modified due to art or programming limitations should be recorded as well. Emails are fine for discussing things, but once a decision has been reached, it should be recorded in the corresponding design document.

5. Limited Integration Time

In an effort to pack in as many features as possible, we simply didn't give ourselves enough time for integration and play testing. For example, most of our character dialogue didn't roll in until a couple of weeks before we were scheduled to begin QA. Integrating the dialogue took longer than expected, and only at the last minute did we discover that some of it could never even be heard.

The AI didn't really come together until the last month of development. There were some minor problems in the character animation system that caused the AI programmer quite a bit of grief, and the pathing system went through several last minute changes. Having more time to integrate those modules would have made things much easier.

Our user's manual was another casualty of our tight schedule. We were so focused on finishing the game, we simply didn't give it the priority it deserved. When someone says the manual isn't important, don't believe them. People really do read the manual, and it should be play tested just like the game. We won't make that mistake again.

Conclusion

SWAT3 was a large project. We overcame some difficult obstacles and shipped a quality game in a relatively short amount of time. The US version shipped in November, and the foreign versions a couple of weeks after that. The game had only a few minor bugs, and no show-stoppers. We've recently released a patch that fixes those bugs and includes a comprehensive tutorial and other information to supplement the manual. The game has received some excellent reviews, and we're excited that so many people enjoy playing it.

We made a good decision holding off on multiplayer. We're working on it now and are planning to add many more gameplay options than we originally intended. It's coming along very nicely, and everyone's really excited about it.

SWAT 3: Close Quarter Battle

Sierra Studios
Bellevue, WA
(425) 649-9800
http://www.sierra.com

Release date: November, 1999

Intended platform: Windows 95/98/2000

Project budget: $2.2 million

Project length: 18 months

Team size: 20

Rod Fung, producer; Tammy Dargan, designer; Cyrus Kanga, art director; Jim Napier, lead programmer; John D. Anderson, programmer; Gary Spinrad, sound designer; Mark Sigel, artist; Jeff Lane, artist; John P. Anderson, artist; William Todd Bryan, artist; Mark Nicolino, artist; Robert Munsil, artist; Travis Brady, artist; Max Braun, artist; Michael Chaves, artist; Brian Johnston, programmer; Michael Stahl, programmer; Jim Edwards, programmer; Cade Myers, QA lead, Jimmy Kowalski, artist; Paul Balon, programmer

Development hardware: Pentium III 500, 3D accelerator, 128MB RAM

Critical software: Visual C++ 6.0, 3D Studio Max, Character Studio, SourceSafe, Photoshop 5.0, Worldcraft, DirectX 6.1

Jim Napier was *SWAT3*'s lead programmer and wrote the game's graphics engine and networking code. He's now at Microsoft leading the development of a new XBox title. He spends his spare time playing *Total Annihilation* and trying to convince his wife why it's good for his career. While he's waiting for his Buzzsaw to finish building, he checks his email at jimn@highstream.net.

Return to the full version of this article