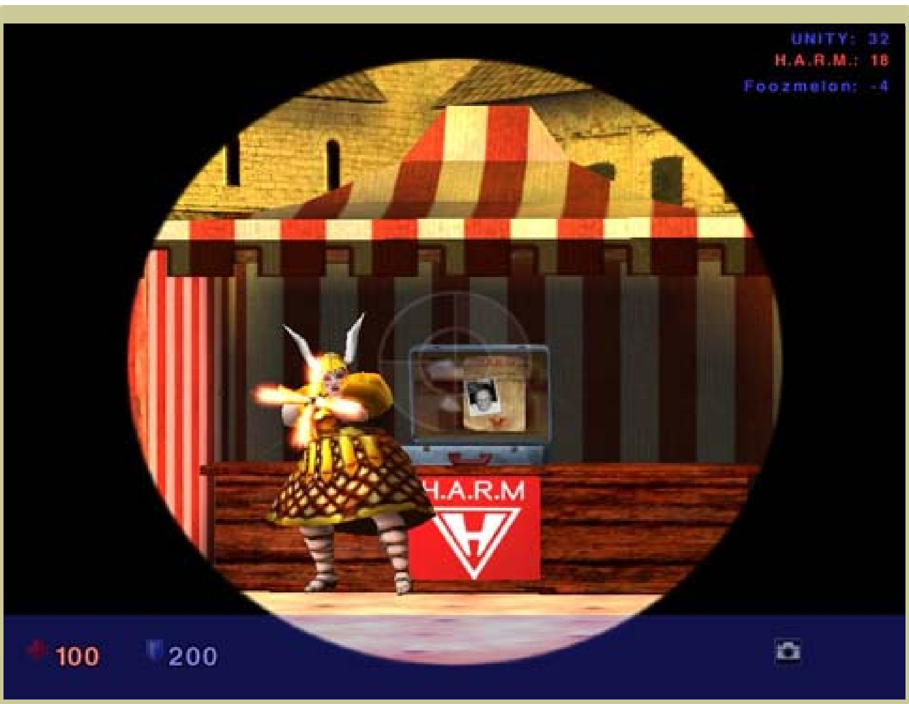
Postmortem: Monolith's *No One Lives Forever*

By Craig Hubbard

Introduction

When we started *No One Lives Forever*, the team had just come off *Shogo: Mobile Armor Division*, which (although critically successful) fell embarrassingly short of our original design goals. In fact, the only thing that saved *Shogo* from complete disaster was the realization, some six months before we were supposed to ship, that there was no way to make the game great in that amount of time. So, we concentrated on making it fun.

Ambition undermined *Shogo*. The intended scope of the project was so grand, particularly for such a tiny team, that we were overwhelmed just trying to get everything into the game. As a result, we didn't have time to polish any of it. The final product is barely more than a prototype of the game we were trying to make, even after we cut characters, settings, story elements, and whatever else we could jettison without breaking the game. It was simply too late to shore up all the deficiencies by the time we realized how many there were. I'm certainly proud of *Shogo* as an accomplishment, but as a game it is a grim reminder of the perils of wild optimism and unchecked ambition.



We felt it was better to release a comparatively humble game that got all the details right than an ambitious one that fell short in numerous area.

We were determined not to repeat those mistakes on our next project. *Half-Life* confirmed our growing conviction that presentation is more important than innovation: although that game is often hailed as having revolutionized the first person action genre, it doesn't do anything spectacularly new. What makes it so influential is that it does everything so well. The pacing is sublime, the situations inventive, the AI incredible, and the overall level of polish unprecedented. It's a game made up of unforgettable moments.

Polish, therefore, was our chief mandate. We felt it was better to release a comparatively humble game that got all the details right than an ambitious one that fell short in numerous areas. To a great degree, we succeeded, for although *No One Lives Forever* was to undergo a great deal of turmoil in the coming months, we never let it get out of control the way we had with *Shogo*. As a result, we managed to ship a product that actually surpassed the goals we set for it. That's not to say we didn't make plenty of mistakes or that the game is as polished as we had hoped, merely that it was a monumental improvement over previous efforts.

What Went Right

1. Mission statement. Once the contract for *No One Lives Forever* was finally signed-an extremely trying process I'll elaborate on shortly-our vision of the finished product was well-defined. We drafted a clear, concise mission statement to make sure we wouldn't lose sight of our goals. This vision guided us through the entire development cycle, serving as a focus whenever the scope of the project began to blur. It was our intention that every feature, task, and ounce of effort would ultimately either support the vision or end up on the cutting room floor.

Our primary aim was to make the player feel like the hero of a 60s action/adventure/espionage movie. We came up with a list of the characteristics we felt were necessary to achieve our objective. The game must have a strong narrative, with twists and turns in the spirit of *Charade* or *Where Eagles Dare*. It must feature a fiercely competent hero and an assortment of despicable villains. The hero must have access to an impressive arsenal of weapons and gadgets worthy of *Our Man Flint*, *Danger: Diabolik*, or *Get Smart*. There must be memorable, death-defying situations, opportunities for stealth as well as all-out action, and a variety of exotic locales to explore. Finally, every aspect of the presentation must convincingly evoke the era.

Our primary aim was to make the player feel like the hero of a 60s action/adventure/espionage movie.

The mission statement was a constant reminder of our original design goals. By sticking to these goals, we were generally able to avoid squandering time on peripheral and extraneous features and content.

2. Flexible systems. In order to streamline the development process, we designed flexible game systems that allowed artists and designers to get content into the game as quickly as possible without the involvement of an engineer. In *NOLF*, adding a fully functional new weapon is a matter of updating some text files. There's no compiling or programming involved. The same applies to many aspects of the game, from adjusting player movement speeds to adding new surface types with unique footstep sounds and weapon impact special effects.

Likewise, our event and object handling systems give level designers the freedom to create relatively elaborate situations without relying on special engineering. For example, the airliner, sinking freighter, and disintegrating space station sequences in *NOLF* were primarily developed by level designers and artists. The relatively small amount of engineering support was focused on polish and bug fixes rather than facilitating things like buckling the hull of the freighter, pummeling the space station umbilicus with meteors, tearing off the tail section of the airliner, and dropping the player from 30,000 feet with enemy paratroopers in pursuit.

Our world editor not only lets us create, texture, and light geometry, but also includes a keyframing system, an interface for setting surface types and rendering properties on textures, and a hierarchical node tree that lets us arrange and manage every component of a world the same way you organize files in Windows. Our proprietary model tool lets us adjust animation speeds, add and position model attachments such as a hat on a character or a silencer on a weapon, and even send messages from a specific keyframe in an animation. Footstep sounds are perfectly synched to animations instead of triggered arbitrarily in code. Likewise, events and camera moves during in-game cinematics are frequently controlled by the animations themselves, which both helps to keep everything synchronized and also allows for precise, deliberate pacing.

There were certainly disadvantages to this degree of flexibility, particularly in terms of the complexity of certain systems and the sometimes unwieldy interface for interacting with them, but the payoff more than justified the expense.

3. Team cohesion. Although we were well into the project before we were fully staffed, we ended up with an amazingly cohesive team, which kept morale stable and productivity high even during the darkest moments of the project. In the course of the interviewing process, we learned to focus as much on how someone would fit the team as we did on the person's skill level or experience. We discovered the hard way how vital it is that everybody have the ability to communicate effectively and weather criticism without taking it personally. Strong technical aptitude is meaningless if you can't cooperate with your peers.

The *No One Lives Forever* team.

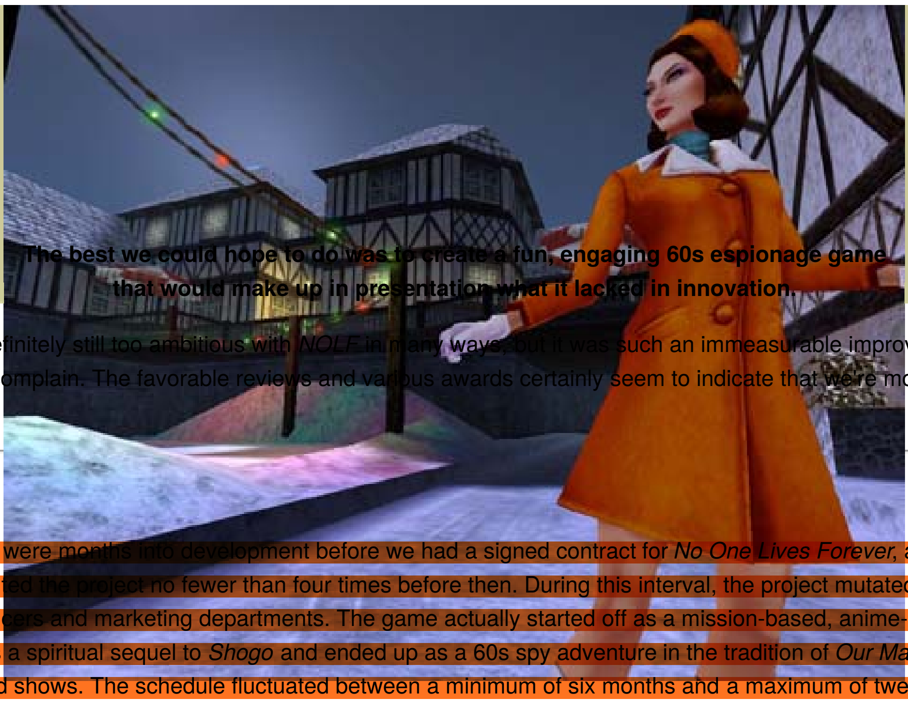
When you're planning a project, it's easy to underestimate the importance of good teamwork. Perhaps it's because many people think in terms of what needs to be done without considering how it's going to get done and by whom. Needless to say, there are numerous ways to solve a given problem, and some solutions are guaranteed to create new problems for someone else on the team. If your team isn't united in its purpose and naturally inclined to collaborate, such misunderstandings may happen frequently and produce unnecessary tension.

Frankly, the strength of our team was more a matter of luck than anything else, but it was a huge factor both in the success of *NOLF* from a development perspective as well as the overall quality of the game.

4. Good scheduling. It's a common myth that game development is more an art than a science and therefore can't be scheduled effectively. Numerous developers adopt a "when it's done" attitude or advocate a strictly organic design process. Sadly, very few companies have such luxuries. For most of us, going a month over schedule means digging into our own pockets to fund additional development time, which can rapidly put a company out of business. Just read the gaming news headlines for lists of casualties.

Therefore, unless you have obscene funding (yes, I'm jealous) or a corporate death wish, effective scheduling is vitally important. Monolith learned these lessons the hard way on Blood, *Shogo*, and various other titles. Certainly any long-term project can run into unforeseen complications, but the development of *NOLF* proved to us that it's both feasible and desirable to build a realistic schedule and stick to it. The scheduling wasn't perfect, due to some oversights in planning for demos and so on, but it was tremendously beneficial in steering us away from the sorts of catastrophic mistakes we had made in the past.

5. Realistic expectations. We didn't have any illusions about what*NOLF* would be. Given our budget, team size, and development cycle, the best we could hope to do was to create a fun, engaging 60s espionage game that would make up in presentation what it lacked in innovation. It's simply naïve to think you can compete directly with a production like *Terminator 2* when you have the budget of *Reservoir Dogs*. The key is to focus on your strengths and not overextend yourself.



The best we could hope to do was to create a fun, engaging 60s espionage game that would make up in presentation what it lacked in innovation.

In retrospect, we were definitely still too ambitious with *NOLF* in many ways, but it was such an immeasurable improvement over previous projects that it's hard to complain. The favorable reviews and various awards certainly seem to indicate that we're moving in the right direction.

What Went Wrong

1. Signing the deal. We were months into development before we had a signed contract for *No One Lives Forever,* although various publishers had green lighted the project no fewer than four times before then. During this interval, the project mutated constantly in order to please prospective producers and marketing departments. The game actually started off as a mission-based, anime-inspired, paramilitary action thriller intended as a spiritual sequel to *Shogo* and ended up as a 60s spy adventure in the tradition of *Our Man Flint* and countless other 60s spy movies and shows. The schedule fluctuated between a minimum of six months and a maximum of twenty-four, the intended feature set scaling accordingly.

Needless to say, it took us a while to get a handle on exactly what we were making, which tended to be a bit disquieting. Really, though, you can't take it personally. You just have to adapt and survive, which is precisely what we did. Each time the schedule changed, we came up with an entirely new proposal instead of trying to cram a 20 month game into 11 months. It just so happened that when the roulette wheel finally clicked to a stop, we were planning to make a 60s spy adventure game. As I mentioned, we were then able to draft a mission statement and stick to it, but we bled away many precious weeks prior to that point.

The schedule fluctuated between a minimum of six months and a maximum of twenty-four, the intended feature set scaling accordingly.

2. Fleshing out the team. It wasn't until some time after we had a signed contract until we had a stable team. Almost every aspect of the project was affected. We lost a product manager, two engineers (although, thankfully, one came back to the fold), and three 3D artists. Level designers came and went.

Part of the problem owed to our publisher woes. On two separate occasions, what was effectively the cancellation of the project resulted in our not being able to hire people we were planning to bring aboard. Without funding for the game lined up, it made no sense to bring on new employees.

Another source of instability for the team was the desire to provide stability for Monolith as a whole. In the aftermath of *Shogo* and *Blood 2*-which led to the restructuring of the company, the liquidation of our publishing efforts, and the resignations of a number of people either burned out by grueling development cycles or disillusioned by the problems that led to them-we had three projects entering development but only a handful of people experienced with LithTech, so we redistributed this modest wealth of experience to maximize our efficiency. In the full course of time, both Monolith and the *NOLF* team were better off, but it was a demoralizing, debilitating process, from which the team took months to fully recover.

One additional complication was the difficulty of hiring level designers who were not only talented enough for the job, but also capable of becoming valuable members of the team. In some cases, we hired amateur designers who were used to making levels for finished games and therefore found it difficult to put up with unfinished tools, missing features, placeholder content, and all the other compromises professional level designers have to contend with on a daily basis. In other cases, we hired professionals whose attitudes, productivity, and quality were anything but.

3. Inefficient pre-production. Given the prolonged uncertainty at the beginning of the project, it's not especially surprising that pre-production was neither as thorough nor as productive as it could have been. We had certainly done a tremendous amount of research, but not enough of it was focused on things that would benefit the project directly.

Most notably, planning for the game environments was too broad and unfocused to provide the foundation we had hoped for. As a result, level designers often started working on a setting with plenty of reference material but without the proper direction. A folder packed with miscellaneous images isn't especially useful compared to a handful of carefully selected references organized and agreed upon by everyone who will be involved in creating the area.

Another complication was that every time the schedule expanded or contracted, which it did on a regular basis those first few months, the feature list changed along with it. Unfortunately, most of the expansion happened late in the process-when the project jumped from 11 months to 18-which meant we had to add features at the last minute in order to fill out the development cycle. Under the circumstances, it was impossible to spec everything out fully before we signed the contract, effectively committing us to features we hadn't fully investigated.

It's easy to dive headlong into development the moment a contract is signed. The excitement of starting something new and the urgency you feel when you look at the schedule encourage you to get started right away. It's only months later when you're slaving to finish up all the partially implemented features and redo unacceptable levels for the third time that you realize that effort spent in pre-production pays for itself many times over during the final weeks of the project.

Cate Archer, the central character in *NOLF*.

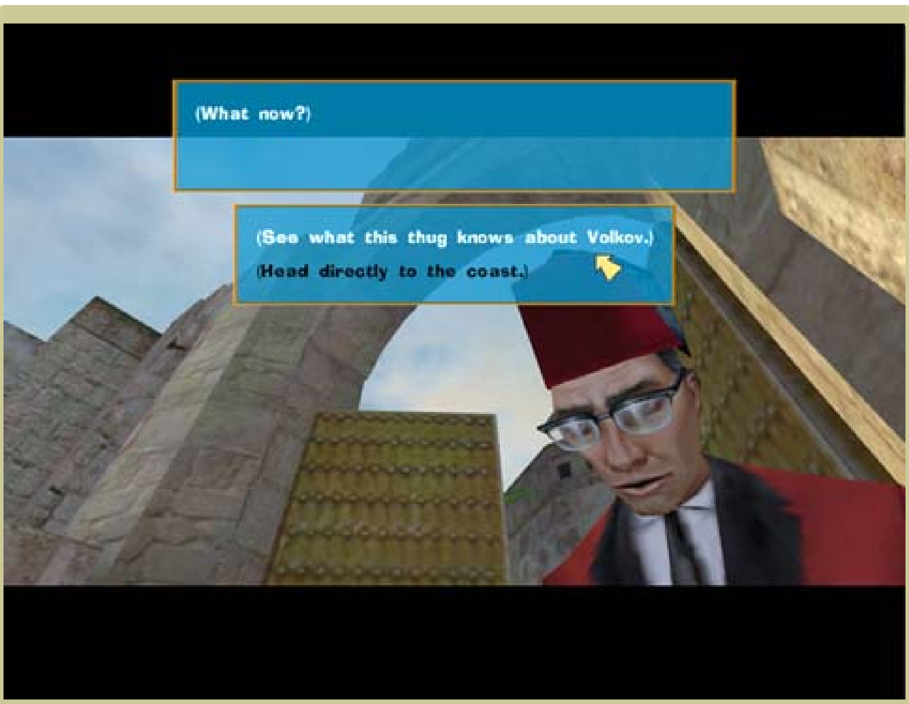
4. Waiting on technology. Although our technology goals for *No One Lives Forever* were relatively modest, they were all so organic to gameplay that the delays we encountered in finishing them prevented us from tackling certain aspects of the game until far too late in the project. Probably the chief culprits were the terrain system, the animation system, and the vehicles, which all proved to be more challenging than we'd expected because, as I mentioned, they weren't investigated thoroughly enough before we agreed to them.

Ultimately, we were able to complete each of these features at least to our satisfaction, but the fact that they all took so long meant we didn't have time to take the fullest possible advantage of them. In retrospect, it was a mistake to sign up for so much technology centered on gameplay for a project of this scope.

5. Cinematic overload. This point is admittedly a bit self indulgent, but I'm the one writing the post mortem, after all. Anyway, if there's one aspect of the game about which I remain ambivalent, it's the in-game cinematics. I'm generally pleased with the understated camerawork and shot composition, and there are definitely scenes I'm proud of, but overall there's plenty of room for improvement. The problems fell into a couple of distinct categories:

Technical difficulties. Implementing in-game cutscenes in*NOLF* proved to be a frustrating, time-consuming ordeal, especially considering how many of them there were. As a result, I generally had to go with the easiest solution rather than the most desirable one. Due to time constraints, I also had to sacrifice many of the cutaways I'd hoped to do to keep the briefings interesting. To make matters worse, we'd done the motion capture based on the original script, so with the cutaways excised, I had a very limited pool of applicable animations to draw from.

The bottom line is that rudimentary cinematic techniques that filmmakers rely on and take for granted can be a massive headache for game developers. A movie director can say, "Hey, Joe, can you pick up the gun, check to see if it's loaded, and then go over and peek through the shades?" For a game developer, just getting a character to pick up the gun convincingly can be a technical nightmare. Any time a character interacts with an object, the environment, or another character, you're likely to spend a lot of time simply trying to make it look natural. When time is in short supply, complicated blocking that adds life to a scene get simplified.


(What now?)
(See what this thug knows about Volkov.)
(Head directly to the coast.)

Implementing in-game cutscenes in *NOLF* proved to be a frustrating, time-consuming ordeal, especially considering how many of them there were.

Conceptual flaws. The biggest problem was a conceptual blunder on my part. Instead of relying on my understanding of scene structure and exposition in film, I fell into the trap of thinking of game cinematics as stylistically unique, partly because games run so much longer than films and partly because the player, by stepping into the role of the hero, would seem to require more information than an audience watching the story unfold. The most obvious byproduct of this oversight was that I deviated from standard screenplay format, which made it impossible to gauge the duration of a given scene. In filmmaking, a page translates to roughly a minute of screen time. The *NOLF* script was far more dense than a screenplay, which made it easy to underestimate the how long a scene would run.

If you've played *NOLF*, you might be surprised to learn that my screenwriting problem has always tended to be divulging too little information. I favor tight pacing, defining character by action rather than dialogue, and subtlety over obviousness. When I was writing this script, probably because I was inventing my own format, I fell into novelistic conversational rhythms. The problem is that what flows on the page tends to drag on screen.

Unfortunately, by the time I recognized the mistake, it was way too late to do anything about it.

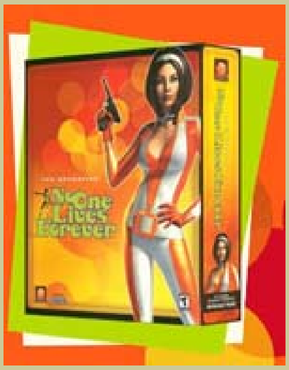
Applying the Lessons

Historically, Monolith has always been brilliant at making mistakes. We've already done most of the boneheaded things that tend to sink development studios. Fortunately, we've also been exceedingly lucky when it comes to surviving these mistakes. Perhaps part of the reason is that we're eager to learn from them so that we don't repeat them. In the five years I've worked here, Monolith has matured from a disorganized but enthusiastic young company to a focused, professional business.

Already, the experiences of *NOLF* have led to significant improvements to our scheduling, preplanning, team structure, and development process. We're more careful than ever before about what we're willing to commit to. We're even more tightly focused on quality and polish.

The expense of developing a game continues to rise dramatically. Meanwhile, the market isn't expanding fast enough to offset these costs. Unless you are financially independent enough to do whatever suits your fancy, the only viable solution if you want to stay in business is to find smarter, more efficient ways to create a competitive product.

Game Data



Monolith Productions
The Operative: No One Lives Forever

Publisher: Fox Interactive

Number of full-time developers: Approximately 18 Core Team Members

Number of contractors: Music, Motion Capture Actors and Voice Actors

Length of development: Approximately 2 years

Release date: Gold on October 20th; Released around November 18th

Target Platforms: PC

Development hardware: Pentium 2/3 300-550, TNT2 and Geforce video cards

Development software: LithTech DEdit/ModelEdit, Microsoft Visual Studio (C++), Photoshop, SoftImage

Notable technologies: LithTech 2.x GDS

Project size:~ 1000 files; ~ 110,000 lines of code

Return to the full version of this article

Copyright © 2016 UBM Tech, All rights reserved