Postmortem: *Thief: The Dark Project*

By Tom Leonard

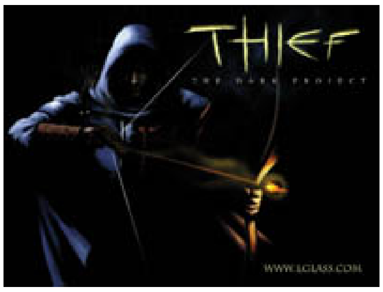
Editor's note: This Postmortem appears in the July issue of Game Developer magazine.

Thief: The Dark Project is one of those games that almost wasn't. During the long struggle to store shelves, the project faced the threat of cancellation twice. A fiscal crisis nearly closed the doors at Looking Glass. During one seven-month span, the producer, project director, lead programmer, lead artist, lead designer and the author of our renderer all left. Worse still, we felt a nagging fear that we might make a game that simply was not fun. But in the end, we shipped a relatively bug-free game that we had fun making, we were proud of, and that we hoped others would enjoy.

The Concept

The *Thief* team wanted to create a first-person game that provided a totally different gaming experience, yet appealed to the existing first-person action market. Thief was to present a lightly-scripted game world with levels of player interaction and improvisation exceeding our previous titles. The team hoped to entice the player into a deep engagement with the world by creating intelligible ways for the world to be impacted by the player.

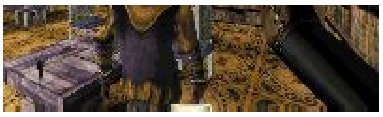
The central game mechanic of *Thief* challenged the traditional form of the first-person 3D market. First-person shooters are fast-paced adrenaline rushes where the player possesses unusual speed and stamina, and an irresistible desire for conflict. The expert *Thief* player moves slowly, avoids conflict, is penalized for killing people, and is entirely mortal. It is a game style that many observers were concerned might not appeal to players, and even those intimately involved with the game had doubts at times.



The project began in the spring of 1996 as "Dark Camelot," a sword-combat action game with role-playing and adventure elements, based on an inversion of the Arthurian legend. Although development ostensibly had been in progress on paper for a year, *Thief* realistically began early in 1997 after the game was repositioned as an action/adventure game of thievery in a grim fantasy setting. Up to that point we had only a small portion of the art, design, and code that would ultimately make it into the shipping game. Full development began in May 1997 with a team comprised almost entirely of a different group of people from those who started the project. During the following year, the team created a tremendous amount of quality code, art, and design.

But by the beginning of summer in 1998, the game could not be called "fun," the team was exhausted, and the project was faced with an increasingly skeptical publisher. The Looking Glass game design philosophy includes a notion that immersive gameplay emerges from an object-rich world governed by high-quality, self-consistent simulation systems. Making a game at Looking Glass requires a lot of faith, as such systems take considerable time to develop, do not always arrive on time, and require substantial tuning once in place. For *Thief*, these systems didn't gel until mid-summer, fifteen months after the project began full development, and only three months before we were scheduled to ship.

When the game finally did come together, we began to sense that not only did the game not stink, it might actually be fun. The release of successful stealth-oriented titles (such as Metal Gear Solid and Commandos) and more content-rich first-person shooters (like Half-Life) eased the team's concerns about the market's willingness to accept experimental game styles. A new energy revitalized the team. Long hours driven by passion and measured confidence marked the closing months of the project. In the final weeks of the project the Eidos test and production staff joined us at the Looking Glass offices for the final push. The gold master was burned in the beginning of November, just in time for Christmas.



Stealth is one of your best weapons in Thief. The game's designers made sure that expert players would have to make effective use of silent weapons such as the blackjack and the bow and arrow. zoom [left] [right]

In many ways, *Thief* was a typical project. It provided the joys of working on a large-scale game: challenging problems, a talented group of people, room for creative expression, and the occasional hilarious bug. It also had some of the usual problems: task underestimation, bouts of low morale, a stream of demos from hell, an unrealistic schedule derived from desire rather than reality, poor documentation, and an insufficient up-front specification.

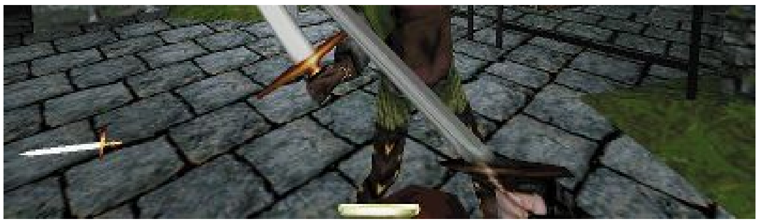
However, *Thief* also differed from a number of projects in that it took risks on numerous fronts, risks that our team underappreciated. We wanted to push the envelope in almost every element of the code and design. The experimental nature of the game design, and the time it took us to fully understand the core nature of that design, placed special demands on the development process. The team was larger than any Looking Glass team up until then, and at times there seemed to be too many cooks in the kitchen. Reaching a point where everyone shared the same vision took longer than expected. A philosophy of creating good, reusable game engine components created unusual challenges that didn't always fit well with schedule and demo pressures. The many risks could have overwhelmed the project, if not for the dedication, creativity, and sacrifices of the team.

Throughout the life of the project, more than 50 people worked in one way or another on *Thief* — some as part of the "Camelot" project, others as part of the Looking Glass audio-visual and technology support staff, some as helpful hands from other Looking Glass projects. The core team consisted of a number of veterans of previous Looking Glass titles (Underworld I and II, System Shock, Flight Unlimited, Terra Nova, British Open Championship Golf, and the unpublished Star Trek: Voyager), as well as some new industry arrivals. The project had a number of very talented people and strong wills. Although it took some time for the team to unite as a tight-knit creative force, the final six months were incredibly productive, spirited, and punishingly fun.

What Went Right

1. Designing data-driven tools. Our experience on previous titles taught us that one of the impediments to timely game development is the mutual dependence of artists, designers, and programmers at every development stage. One of the development goals for the Dark Engine, on which *Thief* is built, was to create a set of tools that enabled programmers, artists, and designers to work more effectively and independently. The focus of this effort was to make the game highly data-driven and give non-programmers a high degree of control over the integration of their work. Media and game systems were to be easily and intuitively plugged in and edited by the team members responsible for their creation, without requiring the direct involvement of programmers.

The Dark Object System stood at the heart of our strategy. Primarily designed by programmer Marc "Mahk" LeBlanc, the Object System was a general database for managing the individual objects in a simulation. It provided a generic notion of properties that an object might possess, and relations that might exist between two objects. It allowed game-specific systems to create new kinds of properties and relations easily, and provided automatic support for saving, loading, versioning, and editing properties and relations. It also supported a game-specific hierarchy of object types, which could be loaded, saved, and edited by designers. Programmers specified the available properties and relations, and the interface used for editing, using a set of straightforward classes and structures. Using GUI tools, the designers specified the hierarchy and composition of game objects independent of the programming staff. In *Thief* there was no code-based game object hierarchy of any kind.



Hand-to-hand combat is sometimes necessary.

Although the implementation of the system was much more work than we expected, and management of the object hierarchy placed significant demands on lead designer Tim Stellmach, it turned out to be one of the best things in the project. Once the set of available properties and relations exposed by programmers was mature, the Object System allowed the designers to specify most of the behaviors of the game without scripting or programmer intervention. Additionally, the relative ease with which variables could be made available to designers in order to tweak the game encouraged programmers to empower the designers thoroughly.

The second major component of our strategy was our resource management system. The resource management system gave the game high-level management control of source data, such as texture maps, models, and digital sounds. It helped manage the game's use of system memory, and provided the data flow functions necessary for configuration management.

Looking Glass's previous resource management system provided similar functionality, but identified resources by an integer ID and required a special resource compilation step. This technique often required recompilation of the game executable in order to integrate new art, and required that the team exit the game when resources were published to the network. The new system referred to a resource by its file name without its extension, used a file system directory structure for namespace management, didn't leave files open while working, and required no extra compilation step. Developers simply dropped art into their local data tree and started using it. To expose art to the rest of the team, lead artist Mark Lizotte just copied art into the shared project directories. Compound resources were treated as extensions to the file system and were built using the standard .zip format. This allowed us to use off-the-shelf tools for constructing, compressing, and viewing resource files. The system facilitated content development by allowing programmers, artists and designers to add new data to an existing game quickly.

The data-driven approach worked so well that through much of our development, *Thief* and System Shock 2 (two very different games) used the same executable and simply chose a different object hierarchy and data set at run time.

2. Sound as a game design focus. Sound plays a more central role in *Thief* than in any other game I can name. Project director Greg LoPiccolo had a vision of *Thief* that included a rich aural environment where sound both enriched the environment and was an integral part of gameplay. The team believed in and achieved this vision, and special credit goes to audio designer Eric Brosius.

As an element of the design, sound played two roles in *Thief*. First, it was the primary medium through which the AIs communicated both their location and their internal state to the player. In *Thief* we tried to design AIs with a broader range of awareness than the typical two states that AIs exhibit: "oblivious" and "omniscient." Such a range of internal states would be meaningless if the player could not perceive it, so we used a broad array of speech broadcast by the AIs to clue in the player. While very successful for humanoid AIs, we feel the more limited expressibility of non-human creatures is the heart of why many customers didn't like our "monster levels."

Second, the design used sounds generated by objects in the game, especially the player, to inform AIs about their surroundings. In *Thief*, the AIs rarely "cheat" when it comes to knowledge of their environment. Considerable work went into constructing sensory components sufficient to permit the AIs to make decisions purely based on the world as they perceive it. This allowed us to use player sounds as an integral part of gameplay, both as a way that players can reveal themselves inadvertently to the AIs and as a tool for players to distract or divert an AI. Moreover, AIs communicated with each other almost exclusively through sound. AI speech and sounds in the world, such as the sound of swords clashing, were assigned semantic values. In a confrontation, the player could expect nearby AIs to become alarmed by the sound of combat or cries for help, and was thus encouraged to ambush opponents as quietly as possible.

In order for sound to work in the game as designed, we needed to implement a sound system significantly more sophisticated than many other games. When constructing a *Thief* mission, designers built a secondary "room database" that reflected the connectivity of spaces at a higher level than raw geometry. Although this was also used for script triggers and AI optimizations, the primary role of the room database was to provide a representation of the world simple enough to allow realistic real-time propagation of sounds through the spaces. Without this, it is unlikely the sound design could have succeeded, as it allowed the player and the AIs to perceive sounds more as they are in real life and better grasp the location of their opponents in the mission spaces.

3. Focus, focus, focus. Early on, the *Thief* plan was chock full of features and metagame elements: lots of player tools and a modal inventory user interface to manage them; multiplayer cooperative, death-match and "Theft-match" modes; a form of player extra-sensory perception; player capacity to combine world objects to create new tools; and branching mission structures. These and other "cool ideas" were correctly discarded.

Instead, we focused in on creating a single-player, linear, mission-based game centered exclusively around stealth, with a player toolset that fit within the constraints of an extension of the Quake user interface. The notion came into full force with two decisions we made about seven months before we shipped. First, the project was renamed *Thief* from the working title "The Dark Project," a seemingly minor decision that in truth gave the team a concrete ideological focus. Second, we decided preemptively to drop multiplayer support, not simply due to schedule concerns, but also to allow us as much time as possible to hone the single-player experience. In the end, some missions didn't achieve the stealth focus we wanted, particularly those originally designed for "Dark Camelot," but the overall agenda was the right one.

4. Objectives and difficulty. One of the *Thief* team's favorite games during development was Goldeneye on the N64. We were particularly struck by the manner in which levels of difficulty were handled. Each level of difficulty had a different overlapping set of objectives for success, and missions were subtly changed at each level in terms of object placement and density. Relatively late in the development of *Thief*, we decided such a system would work well in our game. Extending the concept, we added a notion that as difficulty increased, the level of toleration of murder of human beings decreased. We also allowed players to change their difficulty level at the beginning of each mission. The system was a success in two ways. First, it made clear to the player exactly what "difficulty" meant. Second, it allowed the designers to create a very different experience at each level of difficulty, without changing the overall geometry and structure of a mission. This gave the game a high degree of replayability at a minimum development cost.

Concept sketch of a burrick [zoom]

5. Multiple narrow-purpose scripting solutions. Although the Object System provided a lot of flexibility, we also needed a scripting language to fully specify object behaviors. Rather than create a single all-encompassing scripting system, we chose to develop several more focused tools for scripting. This tiered scripting solution worked well.

In creating our core "high-end" object scripting technology, we wanted to allow designers with moderate programming skill to create complex object behaviors easily. Scripts were event-driven objects attached at run time to game objects, and contained data, methods, and message handlers. The game provided a selection of services to allow the script to query the world state and the game object state, and also to perform complex tasks. Our goal was to create a scripting language that offered source-level debugging, was fast, and was dynamic. The solution was essentially C++ in .DLLs, compiled by the C++ compiler, using a combination of classes and preprocessor macros to ease interface publishing, handle dynamic linking, and provide designers a clear programming model. Though used by both programming-savvy designers and programmers, the fact that it was a real programming language prevented widespread use by all of the designers.

Most designers were interested in customizing AI behaviors. For the AI we created a simpler scripting system, "Pseudo-scripts," that were implemented as properties within the Object System. Pseudo-scripts took the burden of coding scripts off of the designers. The AI provided a stock set of triggers, such as "I see the player near an object" or "I see a dead body"; the designer provided the consequence of the trigger. Each Pseudo-script was edited in a dialog box presenting parameters to tweak the "if" clause of the trigger, and space for a list of simple, unconditional actions to perform when the trigger fired. In this way, the custom behavioral possibilities of the AI at any moment were described by the aggregate of Pseudo-scripts that were attached to that AI. This approach had three benefits. First, it was simple enough so that designers with no programming experience were comfortable using it. Second, it narrowed the range of triggers a designer could use to a good pre-selected set, rather than giving them an open-ended system that might not have worked as well. Finally, when and how to evaluate AI triggers, a potential run-time expense if not carefully constructed, could be custom built by a programmer.

Concept sketch of *Hammer* [zoom]

The final scripting system built into *Thief* was the Tagged Schema system. When the game required motions and sounds, it requested them as concepts modified by optional qualifiers, rather than directly. For example, an AI who had just heard the player would request the concept "moderate alert," qualified with an optional tag like "+sense:sound." A potential set of resources was then chosen using a pattern matcher; in this example, it would choose all samples in that AI's voice expressing a generic "something's not right," all samples expressing "I heard something fishy," but no samples expressing "I saw something fishy." From this set, the specific resource was then chosen using a weighted random selection. The tables used were specified by the designers using a simple language. Specifying motion and sound selection this way, designers created an interesting variety of randomized environments and behaviors without changing the code of the game.

What Went Wrong

1. Trouble with the AI. If one thing could be called out as the reason *Thief's* gameplay didn't come together until late in the process, it would be the AI. The AI as a foil to the player is the central element of *Thief*, and the AI we wanted wasn't ready until late in the spring of 1998. As lead programmer and author of the final AI, I take full responsibility for that.

The original AI for *Thief* was designed by another programmer before the requirements of the revised stealth design were fully specified. Six months after it was begun, the project director and overseer of the system left the team, and the most of the programming staff was temporarily reassigned to help ship another game that was in trouble. During the following months, development on that AI continued without any oversight and without a firm game design. Soon after, the programmer working on the AI also left. While the core pathfinding data structures and algorithms were basically sound, the code that generated the pathfinding database was extremely buggy. The design of the AI decision process was geared towards an action fighting game requiring little designer customization, rather than a stealth game that needed much more customization. Even worse, the high-level decision process in the AI had drifted away from a rigorous design and the code was extremely brittle. The whole situation was a disaster.

These might not have been serious issues, except for one key mistake: I didn't realize the depth of the problem quickly enough, and despite concerns expressed by programmer/designer Doug Church, I didn't act fast enough. I think highly of the programmer involved with the initial AI and wanted to avoid the natural but often misguided programmer reaction within myself that I should just rewrite it my way. So, I took the position that, while buggy, the system as a whole was probably sound. Several months and many sleepless nights later, I concluded that I had been sorely mistaken.

By November 1997, I had the basics of a new design and began working on it. But all work had to stop in order to pull together an emergency proof-of-concept demo by the end of December to quell outside concerns that the team lacked a sound vision of the game. This turned into a mid-January demo, followed by an early February publisher demo, followed by a late February make-or-break demo. During this time the only option was to hack features as best we could into the existing AI. While better than losing our funding, constructing these demos was not good for the project.

In the end, work on the new AI didn't begin until mid-March. Despite the fact that our scheduled ship date was just six months away, we threw away four-fifths of our existing AI code and started over. After a hair-raising twelve-week stretch of grueling hours, the AI was ready for real testing. Had I committed to a rewrite two months earlier the previous autumn, I believe the AI would have been ready for real use three to five months sooner.

2. An uncertain renderer. The project was started because of the renderer, rather than the reverse. The basic core of the renderer for *Thief* was written in the fall of 1995 as an after-hours experiment by programmer Sean Barrett. During the following year, the renderer and geometry-editing tools were fleshed out, and with "Dark Camelot" supposed to ship some time in 1997, it looked like we would have a pretty attractive game. Then, at the end of 1996, Sean decided to leave Looking Glass. Although he periodically contracted with us to add features, and we were able to add hardware support and other minor additions, the renderer never received the attention it needed to reach the state-of-the-art in 1998. The possibility that we might not have a point programmer for the renderer weighed heavily on the team. Fortunately, Sean remained available on a contract basis, and other members of the team developed sufficient knowledge of the renderer so that we shipped successfully. In the end, we shipped a renderer appropriate for our gameplay, but not as attractive as other high-profile first-person titles.

One of the featured weapons is the fire arrow.

This may prompt the question of why we didn't simply license a renderer. When the project started a few months into 1996, the avalanche of Quake licenses hadn't really begun and Unreal was still two years away. By the time licensing was a viable choice, the game and the renderer were too tightly integrated for us to consider changing.

3. Loss of key personnel amid corporate angst beyond our control. Midway through 1997, *Thief* was just starting to gather momentum. We were fully staffed and the stealth design was really starting to get fleshed out. Unfortunately, Looking Glass's financial situation was bleak.

Few emotions can compare to the stress of heading to work not knowing who might be laid off, including yourself, or whether the doors would be locked when you got there. The company shed half of its staff in a span of six months, and while the active teams tried to stay focused, it was hard when one day the plants were gone, another day the coffee machine, then the water cooler.

Some of the *Thief* team couldn't continue under these conditions. We lost two programmers, including the former lead programmer, and a designer. When we were forced to close our Austin office, we lost our producer, Warren Spector, as well as some programmers who made valuable technology contributions to our engine. All of these individuals are now on Ion Storm's Deus Ex team. Although it took some months to fully restore the spirit of the rest of the team, we held together and the company eventually rebounded. Perhaps it bestowed a stoicism that comes from knowing that however bad things might seem, you've already seen worse.

4. Undervalued editor. One of the boils never lanced on the project was our editor, Dromed. Although it was sufficiently powerful and provided the essential functionality we needed to ship the game, Dromed was a poorly documented and sometimes disagreeable editor. Dromed was first developed as a demonstration editor when the target platform of the game was DOS. As a demo, it never received the kind of formal specifications and designs one would expect for the central experience of the design team. As a DOS application, it lacked the consistent and relatively easy-to-use user-interface tools of Windows. An early mistake was our failure to step back and formally evaluate the editor, and then rebuild it based on our experience constructing the demo editor. We also should have designed a proper editor framework, and hired a dedicated Windows user-interface programmer to support it through development. In retrospect, the time lost cleaning up the editor probably would have been saved on the back end of the project.

5. Inadequate planning. Although it is a cliché in the software industry to say our scheduling and budget planning were woefully inadequate, the *Thief* project suffered greatly from this malady. There were several elements to our deficient planning.

During "Dark Camelot," and continuing through the first half of *Thief*, we staffed the team before the design and technology was sufficiently mature. In *Thief*, this led us to rush towards finishing the design, when we didn't necessarily understand the design and technology. With insufficient specifications of both the code systems and mission designs, we ended up doing lots of content that was essentially wrong for the game we were making. Code was written and spaces were built that weren't well directed towards the goals of the project.

To make matters worse, we failed to reassess core scheduling assumptions carefully once the schedule began to slip. Captives of a series of unrealistic schedules, we didn't leave enough time for the sort of experimentation, dialogue, and prototyping a project like *Thief* needs. Late in the winter of 1998, many of our scheduling mistakes had been corrected. Still, during the remainder of the project, the legacy of our earlier missteps required cutting missions that relied on technology we didn't have, and reworking missions not focused on the core gameplay.

Stepping Back from the Project

Thief was constructed as a set of appropriately abstract reusable game components designed for creating object-rich, data-driven games. Although increasing the cost of development, this approach allowed Looking Glass to leverage various technologies across disparate types of games, from the first-person action game System Shock 2 to our combat flight-simulator Flight Combat. In our next-generation technology, some of the systems, such as the AI and the Object System, will merely be revised, not rewritten. We intend to continue with this development philosophy in our future games.

The next time around, our approach to constructing the engine will differ. The engine will be scheduled, staffed, and budgeted as a project in its own right. The editor will be treated as more of a first-class citizen than was the case in *Thief*. Finally, a content development team will not be geared up until the technology is sufficiently mature to allow for an informed game design process.

Oh, and we'll get our schedules right — really.

Info Box:

Thief: The Dark Project
L ooking Glass Studios Inc.
Cambridge, Mass.
(617) 441-6333
http://www.lglass.com

Team size: 19 full-time developers. Some contractors.

Release date: December 1998
Intended platform: Windows 95/98
Project budget: Aproximately $3 million
Project length: 2.5 years

Critical development software: Microsoft Visual C++ 5.0, Watcom C++ 10.6, Opus Make, PowerAnimator, 3D Studio Max, Adobe Photoshop, AntimatorPro, Debabilizer, After Effects, and Adaptive Optics motion-capture processing.

Tom Leonard was the lead programmer for *Thief: The Dark Project*, writing the AI and core architecture of the game. He lives in the Boston area. Tom has been at Looking Glass for three-and-a-half years, prior to which he spent seven years working on C++ development tools at Zortech and Symantec. He is currently working on next-generation technologies, and can be reached at toml@lglass.com.

Return to the full version of this article