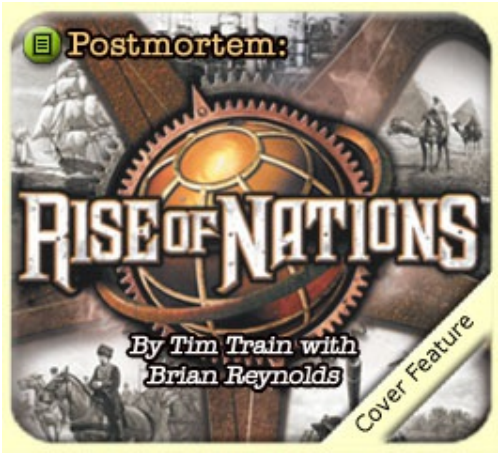
Postmortem: Big Huge Games' *Rise of Nations*

By Tim Train,Brian Reynolds



The name of our company is a summation of our corporate attitude: aim for the top, but don't take yourself too seriously along the way. In this case, "aiming for the top" meant putting together a company and a game designed to go head-to-head in one of the most competitive and resource-intensive segments of the PC marketplace: real-time strategy games. To succeed, our first game needed all the fun, depth, and polish of products that enjoyed bigger budgets and more manpower due to their recognizable franchises. Since they don't give out Game Developers Choice Awards for "Best Game Made With Fewer Than 30 People," we had to find ways to work both harder and smarter if we wanted to achieve our goals.

Big Huge Games was formed in early 2000 by a core team who had worked together for close to 10 years, creating best-sellers such as *Colonization*, *Civilization II*, and *Alpha Centauri*. This history of successful strategy games allowed us to go to publishers with a convincing pitch for a next-generation RTS. Although we understood the issues involved in creating turn-based games, almost all of the areas where we failed to address risks adequately involved areas where we had minimal experience, such as multiplayer matchmaking and making linear single-player campaigns. In addition to these, we also stumbled in some areas that were unique to our situation and company culture.

Rise of Nations

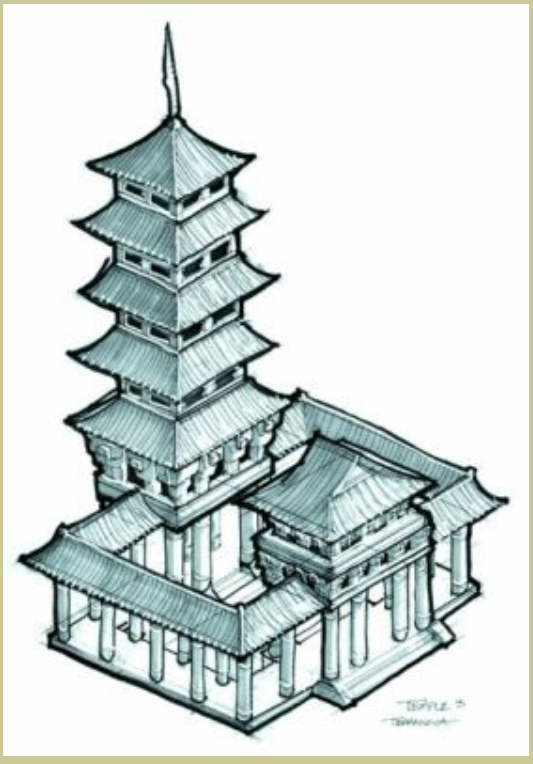
What Went Right

1. Prototype method of game design. Part of the core vision for *Rise of Nations* involved introducing gameplay innovations inspired by our experience making turn-based games into the "classic" real time strategy mix. We had 10 to 15 "wild" ideas about what might take realtime strategy in new directions, but we knew that only some, maybe only a small few, were going to work, and we didn't know which ones. It was essential that we find out as soon as possible which ideas were worth implementing, and we knew from experience that the only sure way to accomplish this is to throw the ideas into a playable prototype right from the beginning.

We got a playable solo prototype running within the first month, and a fully playable multiplayer version more than two years prior to ship. We could throw new ideas in and see the results almost immediately: some concepts needed a little tweaking to be fun, while others got trashed almost as soon as they went in. The value of prototyping is that core concepts end up being continuously refined over years, while providing lots of time to balance the game.

As part of the prototype approach to design, we make sure that everyone in the company is playing the game on a regular basis. After each daily play session, a member of the design team compiles everyone's feedback and sends a summary to the rest of the designers. However,

we found that we had to be willing to wade through some resistance to new features or gameplay tweaks. People would get very attached to certain strategies for playing or even just conventions of RTS games of which they were hesitant to let go.



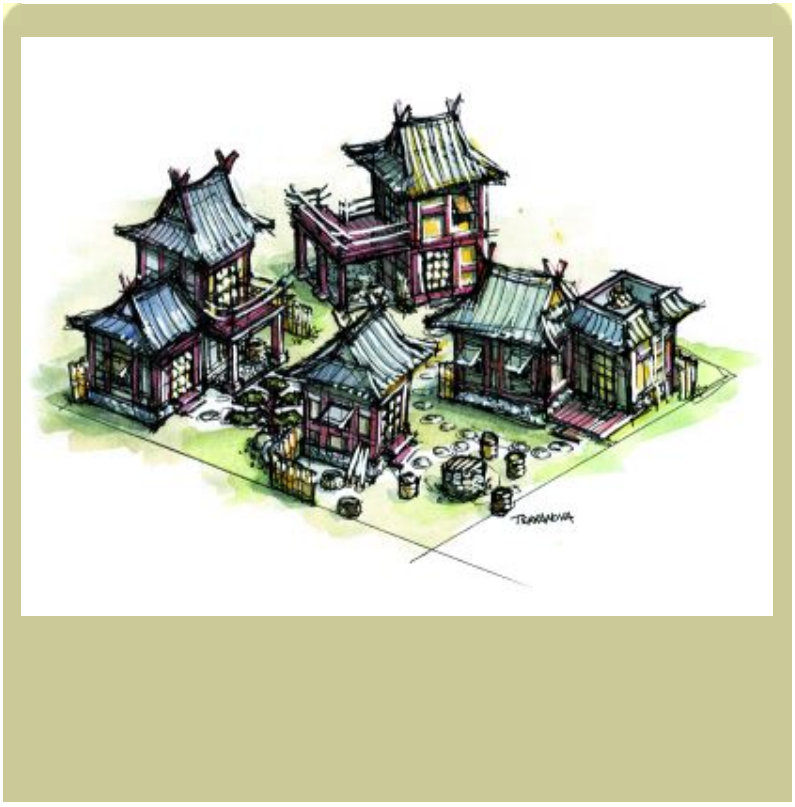
BHG's art team sends constant e-mail updates of works in progress, allowing everyone to comment early in the process.

2. Choosing the right publisher. Over the years, the production values and polish levels on RTS games have risen along with the popularity of the genre. Given the scope of the competition we were up against, a major concern for us was finding a publisher who would be willing to invest the resources necessary in a new company to produce a product that could go head-to-head with the "big boys" - in essence, finding a publisher who shared our culture and values. Our task was made somewhat easier by the fact that the core team had already notched a couple of million-sellers with other companies, but we still had difficulty in selling publishers on our business model.

In early 2000, when the company began, the industry was in the throes of online mania and at the height of the Internet bubble. The "smart" money was flowing to online game sites and massively multiplayer titles. However, in our pitch meeting with Microsoft, we were impressed with their approach: when we asked them which of our five proposals they were most interested in, they just asked us which game we'd be most interested in making. They seemed more interested in the team than in the specific proposal, which in our experience is a great approach to produce top-quality games.

Once we signed on board with Microsoft, we were amazed at the level of support they gave us. When we thought of Microsoft, we took their marketing and sales capabilities for granted, but we were equally impressed by the quality of their development support. Two key groups really helped us polish our game: the play-balancers and the usability labs. Throughout much of the last eight months of the project we had four to six full-time play-testers assigned to the project whose sole job was helping to balance the game. These guys were expert-level RTS players who could smoke the designers after very little time with the game. They helped us find and fix all kinds of broken strategies and degenerate gameplay, and ensured a much more balanced game for hardcore players right out of the gate.

The usability labs took care of the other end of the spectrum, the casual players who aren't as familiar with the ins and outs of RTS games. Between the various tutorials, core gameplay, and the Conquer the World single-player campaign, members of Big Huge took up five weeks of the usability labs as we watched beginner-level players struggle through basic game concepts. Our programmers were there in the labs, coding changes on the fly, able to put a new version up for the next subject. Usability's input resulted in hundreds of changes to the game, making it more streamlined and easy to jump into. It's hard to overstate the contribution both the play-test and usability groups made to the final product.



3. Disciplined hiring process. We started the company with a proven method of developing strategy games through prototyping, and soon afterward we had a publisher that shared our vision and complemented our strengths. What we didn't have, and what would certainly be the biggest single factor in achieving our goals, was a full team.

From the start of the company, we took great care in selecting our staff, adopting an interview system that we thought worked well for Ensemble Studios. All candidates that make it through two rounds of phone interviews are brought to Big Huge and interviewed by every person on staff. After each interview the company meets and discusses the candidate's strengths and weaknesses, at which point anyone can veto a hire for any reason.

Although this process has gotten more time-consuming and difficult as the company has grown, the end result has been well worth the effort. A major advantage of this system is ensuring that every new hire can work and play well with others; if they can impress through multiple interviews with a diverse set of people, then we can have confidence that they will fit in from day one. Perhaps most importantly, it ensures that none of our full-timers has the experience of being introduced to someone they've never met before in their life as ". . . And here's who you'll be working with on such-and-such for the next year."

4. Developing powerful in-house tools. Our programmers worked from the philosophy that taking a little extra time initially to develop a good tool or algorithm pays off manyfold in time saved over the course of the project, while improving the quality of the final product. We also learned not to rely exclusively on one particular tool but rather to use an array of tools to help narrow down a problem. Following are some of the internal tools and techniques which paid the biggest dividends for us:

The section profiler. Our lead programmer, Jason Coleman, created an interactive visual profiler, which helped us identify the correct time segments to bring optimization resources to bear, particularly when trying to identify intermittent spikes in performance that would ordinarily be averaged out when running a profiler over many game frames. Our programmers mark the beginnings and ends of key sections of our code as belonging to one of about 20 color-coded categories (such as render units, network, pathfinding, and so on). Then, as the game runs, the programmers have access to an interactive graphic window with slider bars for time and scale. The profile chart changes color each time the program moves from one section to another, and longer times in a particular section of course result in longer blocks of a particular color. It is easy to spot spikes in a particular section and then, using our "recorded game" feature to repeat a game precisely, a more powerful profiling tool such as VTune can be brought to bear on just that particular time segment, thereby avoiding the "averaging out" effect. The section profiler also helps us spot sections that are being entered too often, even if not for very long.

The parameter window. Graphics programmer Jason Bestimt created an interactive "parameter window" module, which allows our programmers to register as many variables as they'd like as parameters, which can then be interactively controlled during the game with their choice of slider bars, combo boxes, or edit boxes, using a special pop-up console without causing performance degradation. Being able, for instance, to pull up an interactive page that controls all of the render states for any desired graphics element made for great progress on special effects. For example, the nuclear blast effect could be fine-tuned without having to repeatedly recompile (or even rerun) the game: the programmer and artist just sat there and pulled on the slider bars until it looked just right.

The Const System for multiplayer. One of the great nightmares of creating multiplayer strategy games is keeping the game world synchronized across each of the player's machines. Game code and random number generators must run in virtual lockstep across every machine in the game, or the whole game world shatters and goes out of sync - the multiplayer programmer's worst-case scenario which effectively ends gameplay. Interaction with the outside world (such as the commands players give to their units) must be carefully propagated to all machines before they can be safely executed or even safely "seen" by code that can write to the game state. The smallest unintentional bypassing of this rule can result in disaster (such as a graphics routine that accidentally uses the game-side random number generator, or an input routine that directly affects the game world without passing it through the network protocols).

To avoid most of the potential catastrophes of this type we created the "Const System," essentially a compiler-assisted firewall between the game-world side of the code and the I/O (graphics and interface) side of the code. There's game-side data (the simulation) and non-game-side data (everything else). Game-side is allowed read-write access to itself but write-only access to the non-game-side (rendering, for example). Non-game-side is allowed read-write access to itself but read-only (const) access to the game-side (user input isn't allowed to directly modify the game-state).

The real compiler tricks involved the fact that C++ doesn't have an inherent notion of write-only (new standard, anyone?). Also, once this was worked out, all remaining sync issues involved either the occasional, foolhardy overriding of the system, or bugs such as uninitialized data or memory overwrites. Two sets of macros (one each for game and interface access) made this scheme mostly transparent to programmers. The end result was that many of the potentially thorniest multiplayer bugs became easy-to-find "compile errors" instead of nearly impossible-

to-find intermittent out-of-syncs.

Development profilers greatly speeded debugging and prototyping.

5. Great third-party productivity tools. One of the great life-changers for our team was Xoreax's Incredibuild tool. Essentially it lets us turn every machine on our company network into a vast compile farm - the tool automatically makes use of free cycles on everyone's machines to compile our game at jaw-dropping speeds. Even when Rise of Nations was in gold release, our entire game could compile, including all of the libraries, in just under two minutes. Optimizations added an entire extra 15 seconds to the process. Linking the code, the one step which must be performed entirely on the programmer's own machine, took an extra 45 seconds. So, in other words, the final release of *Rise of Nations* that we delivered to Microsoft probably compiled and linked in around three minutes.

The ability to compile (and recompile) the code so quickly led not only to greatly speeded debugging and development, it also resulted in much cleaner code: programmers no longer feared changing header files with the half-hour-plus recompile formerly associated with such an action, so they no longer felt tempted to resort to messy workarounds and hacks just to avoid a full recompile. An amusing side-note is that programmers who had structured their lives around using long compiles to grab drinks, chat, and play chess suddenly had to rethink their whole day.

Other tools well-loved by our programming team include the Visual Assist and Workspace Whiz add-ons to Microsoft's Developer Studio. These two tools add a ton of little improvements to the development environment that end up completely changing the way programmers use Dev Studio. Features include toggling directly from .CPP to .H files, opening any file in your workspace (without needing to provide a path), and automatic grammar correction. Our programmers always comment about how sad they are when they have to debug on a machine without these tools installed.

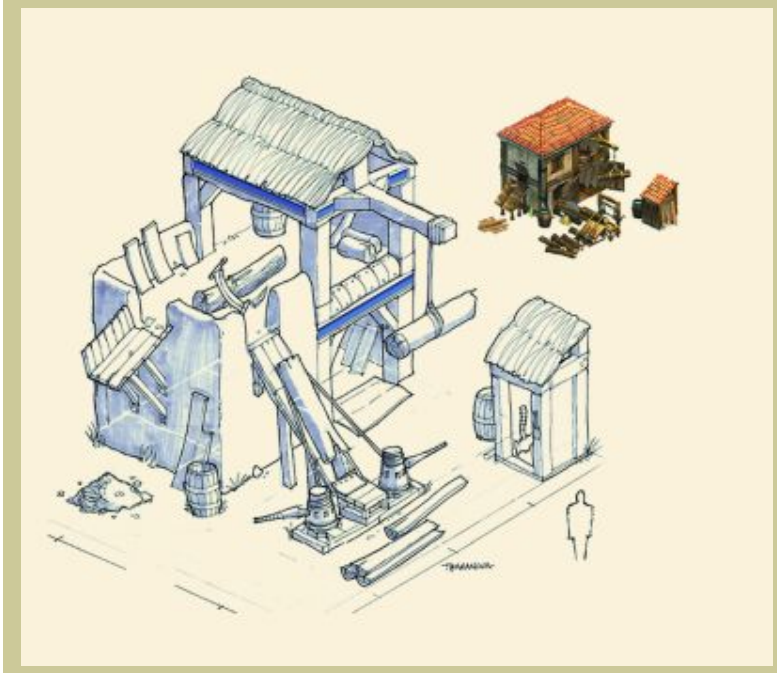
From the standpoint of task organization, the lifesaver for us was Alexsys' Team software. This system replaced our "Post-It note" method of project management with an extremely configurable setup and easy access to tasks. One of the greatest features of Team is its capacity to send e-mail alerts whenever a task is created or modified. As a company we use e-mail a lot, and having a project management system that essentially forced everyone to stay current with tasks was invaluable.

What Went Wrong

1. Not listening to all the other Postmortems ever printed in *Game Developer*. The Postmortems are the most widely read feature in *Game Developer* around Big Huge, and yet somehow we still managed to make many of the mistakes developers are cautioned against in these pages. We underestimated the amount of coding time necessary, which resulted in an extremely overworked programming staff. We misjudged the amount of revision time that we'd want for various systems. We overloaded our lead programmer such that he became the bottleneck on a number of critical systems, including multiplayer and matchmaking. Most of this could be traced back to simply not hiring enough programmers early in the project, and was compounded by lack of scheduling and technical oversight. We also never knew what was "enough" - since this was our first project in the RTS world, we were desperate to cram everything in that we could think of. For the next project, we will certainly hire more programmers and not schedule our lead programmer for anything other than management and support, with the expectation that he will have some flexibility to jump in and help out wherever it's needed. We also expect that we'll have a much more straightforward perspective on scheduling to meet our goals.

Another classic blunder (comparable to getting involved in a land war in Asia) was in undervaluing single-player tool creation. We always assigned our most recent programmer hire at any time to be the "scenario tools" guy, meaning we not only always had our least-experienced team member doing that work, we also had no continuity since we'd pass the torch each time we hired someone new. The editor also suffered grievously from several revamps of the game's terrain system and other parts of the engine. Hence, we had a great deal of difficulty creating single-player scenarios, and we were very fortunate to have the Conquer the World campaign turn out as well as it did. In the last

few weeks of development, programmers Ike Ellis and Scott Lewis finally whipped the editor into shape with some intense hours and smart coding, but for the next game we will have someone working on this module early and throughout the project. This task will also be easier because we'll be working from a more mature engine.



A degree in archecture helped Ted Terranova create sketches of all the buildings in the game.

2. No clear idea of the kind of game we were making. In the first year, both we and the publisher waffled back and forth on whether we should be doing a "classic" RTS game, a completely new kind of strategy game with some real-time elements, or something in-between. The prototype would swing back and forth between those two poles. Eventually, external events (such as the release of *Empire Earth* and Microsoft's purchase of Ensemble Studios) made it clear that a straight-down-the-line "classic" RTS was not the right game to be working on, but not before several months of work on art and game design were wasted going back and forth.

For the next game, we have a much better sense of what our style of game should encompass - epic scope, strategic depth, and large armies clashing - and hopefully we won't have to be so concerned about standing in the shadows of the other giants of the genre. We've also got a much stronger sense about what works and what doesn't in an RTS game.

Over time, the art team developed "style sheets" that standardized the elements for each building set and type.

3. Hard time finding the look from the art perspective. Partly because of the preceding problem, we took longer than usual to nail down an art look for the game. Among other issues, it took us some time to decide whether we'd be fully 3D. The rest of the market was going full 3D, but we really loved the detail and crispness that 2D offered for things such as building graphics. The only engine feature we would give up to go 2D was the ability to rotate the camera, which we'd never found to be very useful in RTS games anyway. However, there was also a lot of hand-wringing about whether we'd be considered behind the curve graphically.

We did numerous tests with 3D and discussed the issue with marketing, but in the end we went with a 3D engine that utilized 2D for buildings. We've been happy with how the game looks and think the high detail on the buildings adds a lot to the world. Next time, we'll do a lot more prototype art and concept work before going into full production, and we'll be more confident diving into a fully 3D world off the bat. Having an experienced, full staff will also help with this issue - at the beginning of *Rise of Nations* we made do with just a couple of artists.

We were also faced with the classic dilemma of doing a history game - it's hard to differentiate yourself from earlier products when you are drawing from common source material. A pikeman looks like a pikeman, no matter how radical an approach you try to take. For the most part, we attacked this issue on the marketing end. We focused on creating screenshots from the later eras of the game and highlighted the diversity in cultural art sets from nations that hadn't been covered as much in other products. Art leads Bill Podurgiel and Ted Terranova worked hard to create units and buildings that were both realistic and varied, ultimately helping to help differentiate the look from that of other products.

4. Solo scenario meltdown. When we started work on *Rise of Nations*, we assumed that the single-player game would feature classic RTS-style linked scenarios that followed a nation's history over time. However, we started work on those scenarios and found that they just weren't particularly appropriate for our subject matter - it's not a lot of fun to take a game about all of history and then constrain players to a smaller canvas and scope. This approach also did not play to our strengths as developers; we are more interested in creating open-ended and infinitely replayable experiences than in making a scripted, linear campaign.

Our prototype design process ended up helping the situation. Starting from scratch, programmer Ike Ellis coded up the skeleton of the module that would become Conquer the World, which aimed to bring context and meaning to linked scenarios that change depending on choices the player makes throughout the game. This campaign mode went from a prototype experiment to a central selling point for the game in less than nine months, and we are planning a new version of Conquer the World for the next game.

Artist sketch for the caravan.

5. Refusal to acknowledge the true state of the game in the final months. Our insistence that we could power through our bug counts at a faster-than-light clip meant that we worked our team much harder than we should have going into the final months of the project. There

were a couple of modules that were going to cause us to slip by the month that we did, and recognizing reality sooner would have saved much of the team a death march that was extremely difficult for all involved.

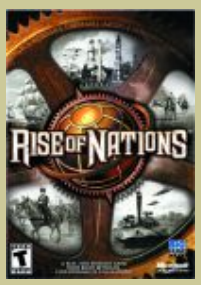
On the next project, we'll retain our faith in our programming team while making more effort to be up-front with ourselves about the status of the project at any given time. Specifically, we need to pay more attention to our bug counts as reflective of the state of the project. We will also be more careful about not calling something "done" until it is truly "done-done-done," and adjust our schedules and expectations accordingly.

A shot of the finished product, *Rise of Nations*.

It Takes More Than Effort

We're extremely happy with how *Rise of Nations* has turned out. We started the company with a vision of how games should be created and how teams function best. In the end, the only tangible validation to our approach is the quality of our games. We said many times during the project that the gaming public only rewards success, not effort; we hope that *Rise of Nations* demonstrates our commitment to both.

Game Data



Rise of Nations

Publisher: Microsoft
Number of full-time developers: 26

Number of contractors:16
Length of development: 3 years
Release date: May 20, 2003
Target platform: PC
Development hardware: WinXP PC
Development software used: Boundschecker, Altova XMLSpy, Araxis Merge, MacroExpress, PCLint, 3DS Max, Character Studio, MS Developer Studio 6.0, Perforce Source Control, Xoreax Incredibuild, Visual Assist, Workspace Whiz, Alexsys Team, Adobe Photoshop, Adobe Premiere, Intel VTUNE
Project size: *.C, *.CPP, *.H: 1,721 total source files, 837, 939 total lines, 24,610,223 total bytes; *.BHS: 46 total files, 24,330 total lines, 966,289 total bytes

Return to the full version of this article