GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Crystal Dynamics' *Tomb Raider: Underworld*

By Eric Lindstrom

[This in-depth postmortem from the creators of the latest Tomb Raider title, originally printed in Game Developer magazine earlier this year, explains exactly what went right and wrong during the making of the critically acclaimed franchise update.]

The concept phase of *Tomb Raider: Underworld* began while its predecessor, *Tomb Raider: Legend*, was in final QA and nearing submission. Previews for *Tomb Raider: Legend* were very encouraging, and we felt that there was still plenty of unrealized potential to tap in the existing feature set.

Enough so, the reasoning went, that we could focus on content and leveraging existing functionality to develop a bigger and better Lara Croft adventure in less time. In many ways this is what the team accomplished, but as is always the case in game development, reality was more complex than we anticipated.

It is particularly interesting to note that much of what went wrong in development involved pitfalls that we anticipated but still fell into despite our efforts to avoid them. It's important to note that most postmortems talk about "What Went Wrong" and not just "What We Did Wrong" because sometimes you make mistakes, but other times you suffer from acts of god and do your best to cope.

A Game Developer article last year ("What Went Wrong?" December 2008) specifically questioned why game development seems to make the same mistakes over and over. In light of that, some of the "wrongs" will be discussed in terms of how our methods to avoid known issues fell short.

What Went Right

1. Long Alpha

We scheduled an unusually long Alpha to deal with unresolved pre-production issues and to set ourselves up for a shorter Beta to make room for more polish time at the end. This paid off handsomely with respect to art production, and it's one of the reasons the game looks so good. We also recovered from a fair amount of design deficit that had carried over from pre-production, and on the code side we managed to get most of our core functionality up to scratch.

Another thing we did right during this long Alpha was to have multiple scope reductions. This was our first "next-gen" title, and we repeatedly underestimated issues of complexity. We would assess and determine that the game was too big, and then cut enough content to bring it under with margin to spare. Then two months later we would see that we were again coming in too big, necessitating further scope reductions.

The game was designed to be able to handle this degree of reduction, as seen by the fact that almost all the features and areas originally planned for the game made it into the final version, only smaller, and connected to each other in fewer ways. We managed to reduce the scope by trimming branches everywhere without having to uproot any of the trees entirely.

2. Focus

When making a sequel, producing a game just like the last one with slightly different content and a few more features is an easy mistake to make. Despite the fact that we had some significant new goals, like making the traversal less linear and bringing back more free exploration, we almost fell into this trap. But many on the team saw that we needed an additional focal point to both rally the team around and to use as a unique selling proposition for the game. The result of this was the concept of epic exploration puzzles.

Making large-scale in-game devices and areas with multiple layers of connected puzzles gave the game an exceptional expression even compared to previous *Tomb Raider* games, and it also gave us a litmus test for spending production effort across the game. This led to the creation of a sub team devoted to these puzzles, which proved to be complicated constructs.

While we would have been better off had we foreseen this need and planned for it properly from the start, the fact that we saw the growing concern, created this special sub team, devoted more staff and resources to it, and assigned a dedicated producer was an example of how well we solved unforeseen problems in development.

3. Production Flexibility

Even though it was hard to significantly redirect the juggernaut that was *Tomb Raider: Underworld*, we made a lot of successful changes when confronted with breakdowns of production processes. We started out thinking that we knew the best way to design the ruined jungle gyms that form the bulk of a *Tomb Raider* playground, based on lessons we had learned from the previous game, but reality confounded us again, and we made changes accordingly.

One clear example is related to level design and art. From the beginning we all agreed that it was critical to have both disciplines work in tandem from day one to make environments that were fun, beautiful, and credible. Prior outings had either started with designgenerated block mesh, leading to geometry that was extremely difficult to turn into plausible ruins, or with beautifully architected tombs that did not provide fun climbing opportunities or properly authored gameplay.

Our solution was to pair level designers and environment artists together by location in the world; such as Mexico or Thailand, and have them work together, iterating on environment architecture to make it both fun and aesthetically appropriate.

While this proved to be the right direction, many of our earlier processes didn't work out. But our willingness to accept the reality that these initial attempts were flawed allowed us take the big step of changing production workflows in progress, often multiple times.

It sometimes felt chaotic and frustrating to change process so frequently, but had we stuck with broken paradigms the situation would have been far worse. In the end, our production methods weren't perfect, but they were superior to what we thought would see us through from the outset.

---

4. Metrics

In a game like *Tomb Raider: Underworld*, where the player can interact with so many different elements of the environment in so many ways, metrics are hugely important. They form the basis of getting Lara Croft off the grid and eliminating the tractor controls that kept her from evolving into the fluidly moving character we have today.

Tomb Raider: Legend suffered from changes in jump distances, ledge parameters, and other metrics until late in development, resulting in countless hours of rework for both designers and artists (it was the most painful for the latter), and we were determined to avoid this in the

sequel.

Our lead level and systems designers collaborated to establish metrics for every aspect of player interaction until a full set was defined early in development. Some changes were made later, and holes were discovered and needed to be filled, but on the whole the degree to which we maintained and enforced metrics without major changes was a huge improvement over past efforts. If not for this success, the amount of game real estate we included in the game would have been significantly reduced.

5. Sanity

Most people working on the game had never been on a project this large, in terms of team size, degree of coordinating efforts, and the amount of game assets we had to fit into the timeline. Add to that how passionate and invested such talented people are in the quality of the final game, and the mixture could have been explosive.

Yes, there were differences of opinion, often strong ones, and certainly there was plenty of stress and the occasional meltdown, but the professionalism rate was extremely high, and this had a large positive impact on the production in ways that went far beyond people getting along.

When scope reductions had to be made, when work had to be redone or discarded, or process changes were needed, maturity was high and ego was low. We had no tantrums over trying to cut someone's favorite elements of the game. Everyone remained rational and evenhanded throughout the length of the project, right into the manic bug-fixing days before submission.

Lesser projects with weaker stresses have broken people and teams, and that is why this deserves to be on this list of what went right- because on a project like this, navigating the many and lengthy trials and tribulations intact contributed as much to its ultimate success as any other element.

What Went Wrong

1. Shared Technology

At the start of *Tomb Raider: Underworld*, Crystal Dynamics as a studio decided to pursue the Holy Grail of internal development: a robust and powerful shared code base to use as a jumping off point for all future games. This proprietary engine would be augmented and maintained by dedicated engineers who could provide the common functionality our games would need, while team programmers could focus on features specific to their games.

The studio believed that because *Tomb Raider: Underworld* and the shared code base would both be based on *Tomb Raider: Legend* code, the efforts could be combined even given our tight production timeline.

A lot of talented and hardworking programmers were put on the shared code team, including many of those who engineered *Tomb Raider: Legend*, so skill was not an issue. The biggest problem here turned out not to be over-ambition or complicated dependencies (though these were certainly issues).

The problems were much more related to ownership and priorities. Within a team, when schedules begin to slip and need to be put back on track, the entire team can get together, redefine its mission, and make whatever collective changes are needed to bring production back into alignment with the calendar.

But the shared technology group was not on the team. While its charter was to serve the teams, it had to serve multiple teams with conflicting needs. From the point of view of each team, the shared technology group was a cadre of programmers that didn't report to our lead engineer. It was an enormous dependency that we could only influence via petition and persuasion, and frequently could not even schedule around as we had limited visibility into their progress.

We knew from the beginning that basing *Tomb Raider: Underworld* on a nascent and evolving code base was an enormous risk, a big potential pitfall. So why did we walk into this trap with our eyes wide open? At the time, it seemed that the potential payoff was worth the risk, and that we had the right people working on the problem, so we marched along with the shared technology group and did the best we could knowing that some of the challenges we were facing could ultimately yield more efficiencies down the road.

Ultimately, however, some of our fears were realized as we had indeed overestimated our ability to overcome all the known risks.

---

2. Refactoring

One of the earliest engineering tasks involved refactoring the player code to make it more robust and more able to accept new functionality. This effort was important to be able to efficiently expand our feature set, but it wasn't understood at the time that this refactoring meant taking apart the engine and putting it back together in such a way that it became unplayable for the duration.

The design team had been anticipating a comfortable period of working on layouts, experimenting with interactions, creating more evolved setups, and otherwise engaging in exploratory and iterative design that wasn't possible on *Tomb Raider: Legend*. But our preproduction hopes disappeared as much of the player functionality went offline.

The usual delays and schedule slips prolonged this dark period, during which the engine parts were scattered across the lawn. The problem was compounded when we had to move into production and deeper into Alpha without many core player functions working. This had a big chilling effect on early design and layout efforts, and the effects of this lasted all the way to the final product.

Fundamentally this was an issue of miscommunication, which is a traditional gremlin lurking on every project. We fought this gremlin from the start, and given the size of our production did a fair job of it, but this particular mishap was unique in how irreversible it was. Improving communication is about better information trafficking up front, but just as importantly, quickly addressing miscommunication when it arises, so we don't consider this one of those traditional mistakes that we repeated.

You can't catch everything, and the effects of refactoring slipped through the net, and once started, there was no going back. It was a particularly bad piece of miscommunication to have, and we just had to ride it out. In the end, a lot of valuable hands-on design time was lost at the start, and we would have had a significantly better layout in the shipped game had we caught the issue earlier.

3. Preproduction

Our preproduction got rocked by the previous two issues-the shared technology effort and how refactoring rendered previously playable mechanics unplayable-but we didn't have the option at the time of moving our ship date. This meant getting much less out of preproduction than we hoped for in two major areas.

First, it put us in a position similar to our time with *Tomb Raider: Legend*, with designers creating layouts based on paper metrics, where the mechanics would not be playable until later. Though this is a clear violation of good design practice, we felt that the damage could be contained because the design team had just finished making a *Tomb Raider* game with many of these same mechanics, so we could get by for longer without them. But the situation was far from optimal.

Second, aside from porting *Tomb Raider: Underworld* to the Xbox 360, this was our first project made for what were then next-generation consoles. It turned out to be far more complex and content-intensive than we anticipated, and because our capacity to create layouts was so compromised in preproduction, we couldn't adequately test, measure, or scope the art side of the production equation.

Because we needed to enter production by a certain date to have a reasonable chance at shipping on time, we ended preproduction before we had an adequate understanding of our art production needs and processes, and just as critically, before we had developed the pipeline and support structure we would need later for outsourcing art.

We were also understaffed in certain areas due to slots on our team being held for team members who were still finishing up *Tomb Raider: Anniversary*. In the end it was as though we planned to have a concept phase, a prototyping phase, and a preproduction phase, but in practice only did some concept, some prototyping, and then jumped straight into production before finishing preproduction.

Much of this pressure came from the collision between ambition and capacity, made worse by the cognitive dissonance between seeing a game on paper that you want to have, and knowing that the production data indicates you can't have it by the prescribed date, but being unwilling to change either one.

At the time, and indeed throughout the project, we knew it was risky to move on to the next phase of production before the previous one had completely finished, yet we did so anyway. This is one of those repeated mistakes, so why did we do it? Cognitive dissonance is part of the answer. Wishful thinking is another part. But mostly, in an imperfect world, every risk you take is a gamble that may fail, and at the end of the endeavor, you end up needing to explain only the risks you took that didn't pan out, like this one.

---

4. Acts of God

This is the category that some postmortems title "Too Many Demos" or some other problem that comes at a team from the outside. We certainly had the problem of too many demos, but those were only one entry in a class of problems that dropped on us unprepared.

Demos were particularly aggravating because we specifically set out to avoid this issue from the start. We demanded long term demo schedules, and we received them. We refused to bow to some requests for demos that weren't on the schedule, and this was honored by the publisher. But there was also a slippery slope, where some unscheduled demos were accommodated because of various circumstances.

We sometimes faced a dilemma where marketing gave us a choice in which a particular demo, at the expense of two weeks of production time, seemed like the better long term choice for the success of the product. There are too many stories of great games disappearing into the noisy marketplace to ignore the importance of demos and good PR.

Yes, these demos often result in a reduction of product quality, and yes they distract from team focus, but in the end, as painful as it feels, it's sometimes best to do the demo. This is a clear example of walking into a trap with eyes wide open-making a mistake that you know about in advance-but it happens repeatedly because the answer isn't to refuse demos, or to plan for them better. The answer is to make a game that doesn't come together only at the end.

We also had an unusual number of weddings, honeymoons, production babies, and untimely departures on this project throughout the team, but most painfully among the discipline leads. We lost our art director midway through production; not to another company, but to another industry, so there wasn't much we could have done about that.

We lost our lead designer toward the end of production when she had a baby; also something we couldn't have done much about (nor would we have wanted to, as the baby is beautiful!). And most tragically, our lead level designer died suddenly during the first half of production.

These key figures in our effort were extremely valuable not only because they were smart and capable, but also because they were a part of the success of *Tomb Raider: Legend*, and therefore knew intimately how to make this kind of game. People like that are rare, because *Tomb Raider* games are hard to make for many reasons, and in the timeline we were under, they were irreplaceable.

We compensated for these losses with people on the team assuming new responsibilities to fill in the gaps, and some of these team members did amazing work and really saved us, but there's no denying that we would have had a much smoother production and a better end product without these losses.

5. Scope

Yes, we tried to do too much. It's human nature to reach as far as you can, and to think your arm is longer than it really is, but some games are more sensitive to scale than others. If you make a sudoku game, you can keep making grids until time runs out and you're done, but with games that have interlocking feature sets, plus a beginning, a middle, and an end, it's much harder to scale properly, and harder still to change scale during production.

There were success stories with respect to our scope, and with reductions we made all the way to Beta, but in the final analysis we tried to do too much, got locked into needing to do it, and scrambled to get it all done to the quality standard we hold ourselves to.

The reason why scale kept outdistancing our deadlines is mostly because we underestimated the complexity of next-gen development and did not spend enough time in preproduction smoking out all the issues.

The biggest casualty of making too much game wasn't the crunch time-we had less crunch than on past games-and it wasn't that we left parts of the game undone, since there are no holes in the experience. The most damage was to an element that was very important to us: polish. We padded our schedules an unprecedented amount, and then some, in order to have sufficient time to polish.

Unexpected complexities in next-generation development, along with communication throughout a team larger than the studio had ever seen, consumed our polish time and more. This makes the mistake of underestimating our polish time more understandable, even though this is also one of those repeated "what went wrong" scenarios, because we were dealing with a whole new class of unknowns. This won't be the case going forward, however, so the team should be equipped next time to avoid the usual mistake of not scheduling enough polish time.

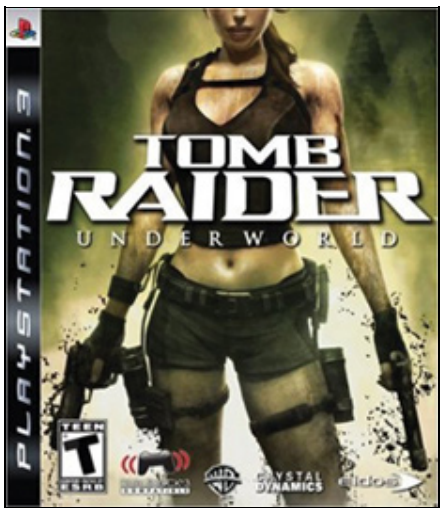
Risks and Rewards

Our game was intended to be a strong sequel that drafted heavily off the successes of *Tomb Raider: Legend*, but turned into a major product in its own right; bigger and more lush than anyone could have reasonably expected at the outset, because of the hard work, talent, flexibility, commitment, and level heads of the many team members who created it. It's very gratifying that some reviews have hailed *Tomb Raider: Underworld* as the best *Tomb Raider* game since the first one, and the fan response has been phenomenal.

So why did so many things that we guarded against go wrong anyway? The categories described here are only five of many more. (Broken builds and long build times get honorable mentions, and the added distinction of raising blood pressures more than probably all other issues combined.) Doesn't the foreknowledge of pitfalls make possible the absolute avoidance of them?

No, it doesn't. Creative endeavor by its very nature is chaotic, and where creative chaos, ambition, and dedication collide with production concerns and the hard walls of a publisher's deadlines and goals, you're going to fall into many of the same holes, or worse, get steered into them. The answer isn't to do everything possible to avoid all known traps, because doing so would kill innovation and creativity; the trick is to be aware of potential pitfalls, and to be prepared to quickly and nimbly mitigate the negative impact of stepping into one.

Game Data



Publisher: Eidos

Developer: Crystal Dynamics

Number of Developers: 84 internal staff, 15 contractors, not including shared technology staff

Length of Development: 2.5 years

Release Dates: November 18th USA, November 21st EU

Hardware: Quadcore PCs with 64-bit OS, 4 GB RAM, 500 GB RAID 1

Software: Turtle rendering package, Bableflux, Perforce, MotionBuilder, Maya 8.0, Zbrush, Photoshop, Bink, FMOD, Test Track Pro, and proprietary graphics and physics engines

Platform: Xbox 360, PS3, PC (also Wii, PS2, and Nintendo DS)

Return to the full version of this article