GAMASUTRA
The Art & Business of Making Games

Postmortem: NinjaBee's A Kingdom for Keflings

By Steven Taylor

[In this Gamasutra-exclusive postmortem, NinjaBee explain what went right - and wrong - while creating Avatar-enabled Xbox Live Arcade worldbuilding game A Kingdom For Keflings.]

A Kingdom for Keflings is NinjaBee's fourth self-published Xbox Live Arcade game for the Xbox 360 (after *Outpost Kaloki X, Cloning Clyde*, and *Band of Bugs*). Following our usual passion for something new, we created an original IP and designed a new game with unique game mechanics.

We hoped a happy avatar-themed city-builder with a focus on the inhabitants of the world would appeal immediately to sim game players and be easily accessible to anyone else who wanted to give it a try.

The core team was the usual mix of programmers, artists, and designers, but in this case most of them had worked on one or more of our previous Live Arcade games and brought helpful experience with them.

After the release of *A Kingdom for Keflings*, we made a particular effort to play the game online with strangers and hang out in forums, listening to feedback, helping with questions, and generally collecting notes.

The total response to the game has been more positive than we ever expected, and playing with enthusiastic players has been a joy. On the other hand, there's plenty of negative feedback to go with the positive, and we've dutifully taken our lumps there as well.

As one of the designers of the game, I've put more of my own weird ideas on the line in this game than in any game I've worked on before. It's thrilling for me to hear praise and deeply painful to listen to well-deserved criticism.

This contrast results in great psychological trauma, and rather than work it out on a psychologist's couch, I'm going to seek therapy through this post-mortem analysis of the project.

What Went Right

1. Happy Creation-Focused Design

Play as a giant, interact with the little inhabitants of your world, and build them a kingdom. This was the core idea of the game, and from the first few days of actually putting together a production plan, it was clear that what we wanted was happiness, love, and creation instead of pain and loss.

This was questioned continually throughout production, usually by any new member of the team. Where's the fighting? Where's the conflict? When does my village get attacked by Godzilla? I personally lost a lot of sleep over this question. But we stuck to the original vision that the game would be compelling and fun because of what you were *creating*, not because of what you were fighting over.

We couldn't be more pleased with the results. As a designer, the first time I finished testing a feature and *kept playing* because I was having so much fun was a pretty good moment, with many moments like that to follow. For every player who suggests we add combat, there are five players who thank us for creating a fun, relaxing experience that their whole family can appreciate.

2. Evolving Design

While holding on to the core concept of the game, the team leads tried to be as flexible as possible with new ideas and feedback and let the design evolve naturally. The original prototype for this game, which art director Brent Fox and I threw together as a game-in-a-day challenge, featured villagers and seasons. Pretty much everything else changed many times over, with good results.

The idea of playing the game with an avatar character came well after the initial concept document, as did the tech tree interface, the intro cinematic, love as a resource, banner towers, ground markers for blueprints, and many other core elements of the design. If this meant throwing out half-baked ideas and spending extra time experimenting, we accepted this as part of the process.

We consider this one of the huge advantages of an indie studio developing a self-funded game -- this approach seems most viable when the development team has the final say and the final responsibility for success or failure.

As a side note, using a Wiki system (like MediaWiki) to maintain design documentation is a big win when the design is continually mutating.

---

3. Playtesting the Initial Experience

Partly as a result of concerns about how people would react to the low-conflict design, we started inviting playtesters to come in regularly and give us feedback. In particular, we focused on the initial reaction people had to the game and how easy it was to learn how to play.

We tried to apply what we've learned from Microsoft about getting good feedback (primarily, shut up and let them play) and we collected detailed notes from every session. It's tough not to overreact to individual comments, and it's important to use playtesters for feedback on existing design, not for new design.

Some very clear feedback came from these sessions, and as a result we rewrote the tutorial several times, reworked the initial building sequence more than once, and made hundreds of small changes to controls, UI, and interaction with Keflings.

We also made a series of large changes and additions (such as the introductory cinematic the player sees before playing the game) to help the user understand the focus and scope of the game, rather than expecting it to be something it wasn't.

4. Design, Technology, and Production Experience

Our previous experience designing and building Xbox Live Arcade games heavily influenced the decisions the team made on this game. From design ("there's too much waiting around in *Outpost Kaloki* -- let's fix that") to technology ("our compressed file system code is working well -- let's use that again"), to production ("*Band of Bugs* had way too many features and modes, let's not do that again!") we tried to apply what we've learned in the last few years.

Of course, we keep making new games which are unlike our previous games, and a lot of new game mechanics have to be developed, but this is easier when we know what common pitfalls to avoid and have a lot of core technology on which to base the new stuff.

5. Avatars

A whole lot of little things went right with implementing Microsoft's Xbox 360 Avatars, all adding up to an amazingly positive experience.

First, we were lucky to be in the right place at the right time for the new Xbox Live Avatars. We had a game based around an avatar character that (after some initial concerns about art style) ended up being a perfect fit for Microsoft's Avatar system.

Second, to combat the limited window of time we had to integrate Avatars, we worked out a detailed plan for how the implementation would work, including some contingency planning and prioritization of features in case anything needed to get cut.

We ended up following this plan fairly closely, and knowing what everyone needed to be working on at any given time was nice.

Third, the small team at NinjaBee doing the integration work had the right skill set, expertise, and passion for the project. They dug in fast, solved some very difficult problems, and put in the extra time required to make everything work right.

Fourth, the extra technical and marketing support we received from Microsoft (as a result of being part of the New Xbox Experience launch) was hugely valuable. Of course, this is part of why we jumped on the chance be part of the Avatar launch, but the attention the game got was beyond what we had originally hoped for.

What Went Wrong

1. Lack of mid-game and end-game playtesting

Sure, we extensively playtested the early experience. But it was a long time before we had a stable build that let us play through the game. That plus too much focus on the early game and not enough on the middle and the end resulted in balance and quality issues.

We played through to the town hall what feels now like a thousand times, but played the mid-to-late games far less. There are some confusing points mid-game where it's easy for players to get stuck.

There's a bit of a late-to-mid-game grind that I wish we had eliminated. There's a general lack of balance in some of the late game when it comes to time between accomplishments for the user. And there are crashes later in the game that more thorough testing would have uncovered.

2. Dynamic Music System

I know we're far from the first developer to tell this sad story. The original design called for a flexible dynamic music system, avoiding the monotony of the standard music loop by using a system of layered, changing riffs and variations.

We put huge amounts of time into this from designers and programmers and audio people. We rewrote the code several times, spent many hours massaging the music assets, and wrote pages and pages of scripting. But...

While there were moments where it seemed like it was going to work great, the final results were ultimately not acceptable.

At the very end of the project we scrapped the dynamic flow of music and rewrote the system one more time, keeping only a simplified version of the dynamic layering of tracks based on city complexity. And even that is a subtlety that was easily not worth the effort we put into it.

3. Texture Management

One problem with an open sandbox world is that we don't directly control the complexity of the scene being shown to the user. Almost the entire tech tree can be unlocked and built in one world.

Add in the multi-texture effects in the terrain, support for user-painted roofs and walls, and season-specific building and ground textures, and texture management gets pretty complicated.

We knew it would be an issue, but we underestimated the painful impact this would have on total memory usage and we planned poorly from the start.

It would have been harder to arrange, but we should have been more aggressive in making building assets share common elements (roof materials, wall materials, etc.), and we should have been more consistent in building design.

---

4. Naïve Multiplayer Expectations

Sure, we knew that mean people would tear other players' kingdoms down, but we treated it like an abstract theoretical problem that didn't need a lot of attention. Hosts can kick other players, games can be saved and reloaded, anything broken can be rebuilt, and, after all, people will mostly be playing in private games with their friends anyway, right?

In practice, kicking other players is slow and not obvious. Expecting an ongoing game to be stopped so the host can reload it is quite unreasonable.

Broken banner towers cannot be rebuilt, and other repairs are a pain when components are broken all the way down to their resources.

It also turns out there are a ton of people playing in public games, especially when they're looking for their banner tower-related achievements.

A simple system of controlling who has permission to break things would have gone a long way toward helping with this very painful problem. Of all the things we wish we had done differently, this probably tops the list.

5. Small Developer Blues

Our pockets are not deep. We had only the money we had saved up and additional cash borrowed from friends to make this game. Budget pressure meant cutting a few corners here and there, not fixing a lot of things we would loved to have fixed at the end, and cutting features we would loved to have added, like local multiplayer.

On top of this, we need to keep everyone at the company gainfully employed, so as team members came off other projects we had them help out with this one even when the timing was wrong.

Designers were working on particle effects before the game-specific technology really let them see what the results would look like. Artists were doing production work when the leads weren't available to work with them on direction. All of this meant less efficiency and more pressure on the pocketbook.

We went over budget and way over schedule, risking damage to the company. As a consequence of playtesting and an evolving design, we felt there were certain features that *had* to be included, and felt it was critical to scrape up the resources necessary and delay release to get

them in. We don't regret this decision, but we certainly will consider these lessons when planning future budgets.

Thank You, Doctor

That's about it. I believe we can directly apply every one of these "what went wrong" lessons to our next project, and build on the things that seem to be going right.

The experience of making and releasing this game, even with the bad moments mixed in with the good, ranks among the most interesting and rewarding things I've done with my career so far. Life is good. This concludes our therapy session.

Game Data

Developer: NinjaBee
Publisher: NinjaBee
Release Date: Nov 19, 2008
Platform: Xbox Live Arcade
Number of Full-Time Developers: 3 -- 12 depending on stage
Number of Contractors: about a dozen, including audio, QA, and localization
Length of Development: 18 months
Development Software: Visual Studio, Photoshop, 3DS Max, Excel
Technology: NinjaBee's Wraith Engine

Return to the full version of this article