Postmortem: Unknown Worlds Entertainment's Natural Selection 2

PRINT

By Charlie Cleveland

On Halloween of 2002, Cory Strader and I released a free *Half-Life* mod called *Natural Selection*. It blurred the lines between shooter and RTS, in a wildly asymmetric, harsh and fast-paced sci-fi environment. It became the most popular independent *Half-Life* mod, and I became forever hooked on independent game development. I lived off my $40,000 of savings during its development and I became highly motivated to turn this into a living.

I moved to San Francisco to find investors and teammates, and to start a new life: one that I hoped would include a sequel to *Natural Selection* and enough income for me to become self-sustaining. After a few false starts, I started Unknown Worlds, and convinced my new partner Max McGuire to go in for half. At the end of 2006, we started working on *NS2* in earnest. We lived off our own savings until we got some angel funding in 2008 and then carefully started growing the team.

This past Halloween, in 2012 -- 10 years after the release of the original Half-Life mod, and after almost going out of business multiple times -- we released *Natural Selection 2* using our own "Spark" engine on Steam. It went right to Number One and has since sold around 300,000 copies. This article hopes to summarize what we learned during this epic period of toil.

What Went Right

1. Unstructured Development

We had no producer, no schedule, no milestones, and no meetings. We also had no design document, no technical plan and no business plan. Anarchy? Not exactly.

Our core team was seven people in a room, but we also had many critical remote contractors and part-time community contributors. Every time we tried daily meetings, we ended up deflated and demoralized. It made no sense why spending 10 minutes a day talking about our work made us so unhappy, but at least we were smart enough to stop. No status meetings naturally extended to no meetings at all.

We did try to have a schedule, but we found it was inaccurate and out of date instantly. It didn't seem worth maintaining something that was perpetually wrong, so we stopped doing it. After writing a business plan, I realized that I learned almost nothing writing it. A year or two later I realized I never looked at it, either. If I wanted to describe "our plan" to someone, it was easier to send along a couple paragraphs by e-mail. Very few people will get through a 40-page business plan anyways.

What we did do, though, is have lunch together every day. Anything important came up then, and the time there was naturally bounded, so talking didn't go on too long. The daily ritual of breaking bread was a powerful one for our team cohesion and happiness. And not having structure meant we could adapt to changes very quickly, and that we could spend all of our time working on the game, instead of planning it.

2. Empowered Design

One reason this unstructured approach worked for us was due to our design process. Instead of a design document, we used four "pillars" to describe the game. Each pillar is the name of a feature, described in specifics by a short paragraph. Imagine the bullet points on the back of your future game's box (even if it won't have one). All decisions about high-level features were compared to these four pillars.

This let everyone on the team have a set of "beacons" for the game direction, which allowed everyone to vet new ideas, have discussions and cut long conversations short if they weren't advancing one of these high-level goals forward. Having the pillars be high-level allowed us to be as flexible as possible, while still keeping the "spirit" of the game intact. You can see that these pillars are the main topics on our game page.

At a lower level than features, we kept artists, level designers, programmers, and animators working together cohesively through a set of design principles. These principles are "rules" that I implicitly followed when making decisions about balance and goals for the game. Examples include "No hard counters," "Public vs. competitive play," and "Non-obsolescence."

For each, I included a paragraph or two of reasoning and examples. I felt bogged down explaining my reasoning behind balance and design decisions, so I needed a way to quickly get everyone on the same page.

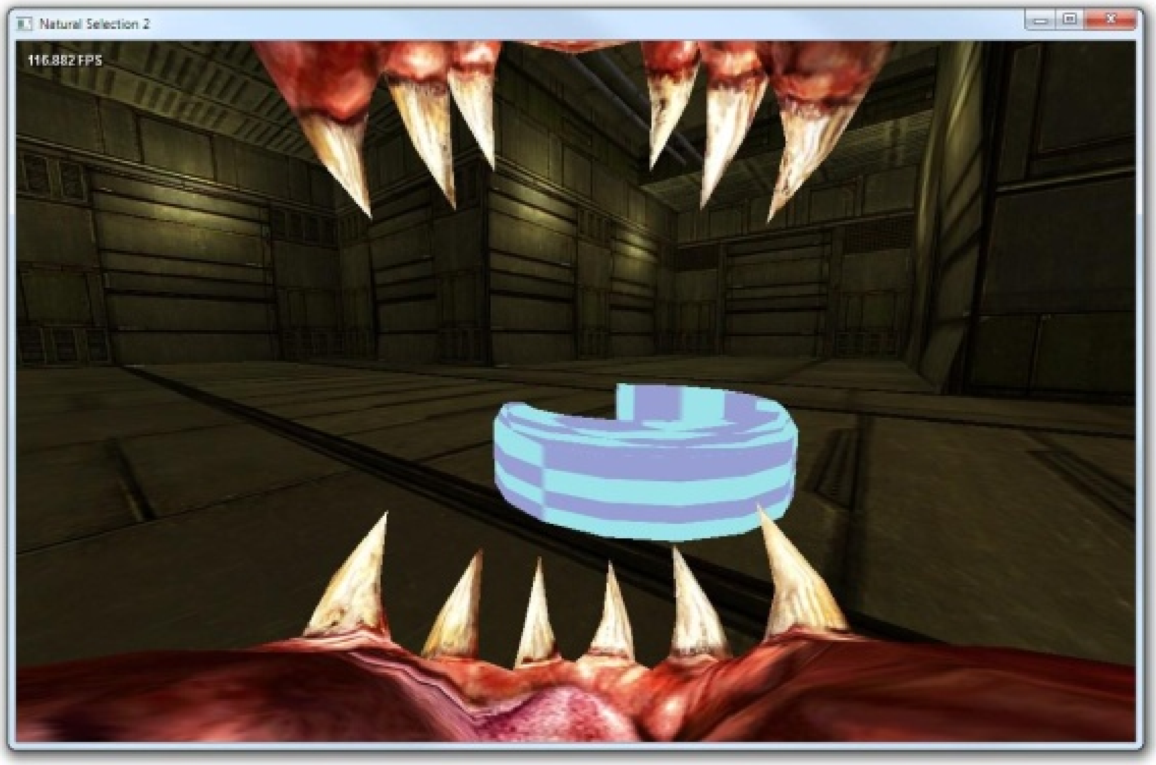
For instance, the marine team can research generic weapons upgrades, which let them do more damage. The aliens have no analog (following the "Two Unique Sides" design pillar) -- so in the late game, aliens can be ripped to shreds by even starting marine weapons.

Someone made the suggestion that aliens automatically have their armored shell able to take more damage, the more hives their team has (this was present in the original *Natural Selection*). But the design principles include "No Magic Numbers" -- in other words, hidden modifiers that are not readily apparent to the player shouldn't be in the game. This principle immediately changes the conversation away from any variant of hidden scaling armor, and towards adding new specific traits that could be used to compensate (extra speed, cloaking, etc.)

The specific goals in the document don't matter, but they shouldn't change much. I wrote these design principles in one sitting and they stayed throughout the rest of development. Embodying these cultural design principles in a public document empowered not only our employees, but our community members, and allowed everyone to contribute their art, sound, code or design that "fit."

3. Iterative Process

From day one, everyone on team played the game -- even before we had any art assets or game-specific code. The first thing we did was create a basic level, coded a rifle and an alien bite, and ran around a small box attacking each other. There wasn't even an interface or menu -- you had to know the right console commands to start or join a game. You can see here the Skulk view model from *NS1* and an untextured placeholder Infantry Portal model.



Our first playable prototype in Spark.

These tests were great for boosting our morale, and getting us in the right frame of mind: shipping. We had a game, albeit an ugly and boring one; now we just had to make it fun. Instead of the end goal remaining amorphous and undefined, we simply had to shape what we had into something great.

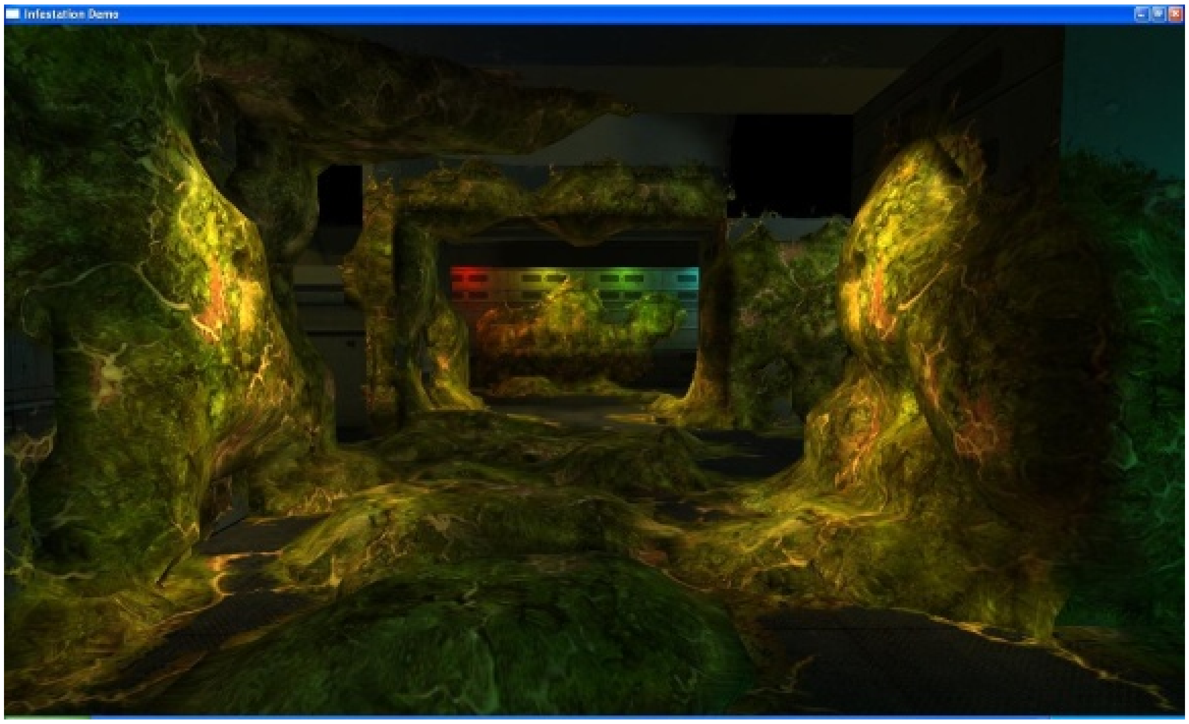
As development progressed, we played more and more, until in the beta the entire team was playing a couple hours a day. It's quite difficult to ignore gameplay or implementation problems when you're hearing or seeing their exasperation about some aspect of the game. Many features were reworked multiples times and/or cut to fit our tastes.

By the time we released, *NS2* had many thousands of games played, so we didn't have much to fear. Through this iterative development, I came to understand just how important it was to take feedback from non-designers, and the game was so much better because of it.

4. Open Development

As a small company without a marketing budget, how do you build an audience? We decided to take our fans "behind the scenes" and open up our development. We blogged, tweeted, recorded video footage around the office, and otherwise became as transparent about our game development as possible. When we didn't have game footage, we showed screenshots. When we didn't have screenshots, we showed concept art. Before concept art, we wrote about what we were planning to create, and why. Even the most mundane aspects of game development can be interesting for people outside the industry.

The first time we did this was in 2006, when we released our "dynamic infestation" prototype. It ended up getting linked all over the web, which felt incredible. We worked with one of our "super fans" who was a video whiz to create development videos about design and technology decisions. These regularly got tens of thousands of video views, and gave us feedback much earlier than we otherwise would've gotten. We even cut our "taser" weapon after we showed the concept art and basic functionality: our fans hated it so much we just dropped it.

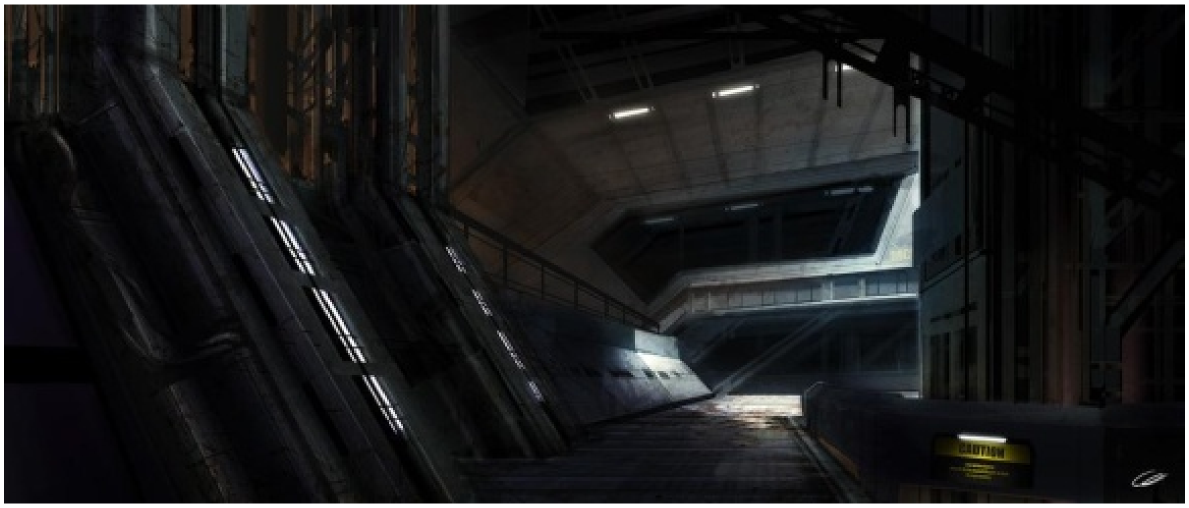


Our early infestation demo.

Besides the PR benefit and the early feedback, open development had another bonus for us. When we were almost out of money (the first time!), we thought: "Why don't we ask our community if they would pre-order our game?" We put together a polished video that shows the technology, and a first look at the new game, and launched it with our preorder program. It ended up making over $1M over the next two years of preorders, saving the company and buoying our morale.

Open development helped us again when the *Overgrowth* team suggested that we do an unprecedented "preorder bundle" of our two games, and make it a PR event. We thought it was a great idea and we generated another $40k each in just a couple days. It worked so well that they then formed Humble Bundle, Inc. and turned it into a profitable venture-backed business.

Another way we leveraged open development was way before our playable alpha. We didn't have any maps yet, but we did have assets for making maps. So we released these props and textures, along with a document describing how to build a map and our barely-functional editor. Some of our pre-ordering folks started making maps right away. We asked the most promising community members to work on official maps for the game, for a share of the eventual revenue instead of our sparse cash. Without releasing our assets to our community, I'm not sure how we ever would've built our first map.



Early community mapping shot using our released assets.

Open development helped us in countless other ways. We used it to assemble a die-hard group of volunteers that performed all our QA and gave us tons of valuable feedback. We hired a key game programmer from the community -- he had never worked on a game before, but proved his worth with dozens of gameplay videos he assembled using the source code we distributed with the beta. It's now clear to me that we would not have shipped the game without him.

At every step of the way, we would release what we had, talk about what we're building and why, and listen to feedback. Whenever someone talented emerged from the community, we did everything we could to work with them. Whenever we created anything of value, we would release it, and hope it would grow momentum in the form of revenue, talent, audience, or buzz.

5. No Crunch

One of the reasons we started Unknown Worlds is because we believed making quality games and having a great life went together. We view game development as a marathon, not a race. So we made sure to keep fit, have a life and not crunch. There were times when we did work late nights and weekends, but only when our company situation or funding was dire. Looking back on those times, I do believe we did the right thing to work extra, but we probably would've been better off had we taken a step back and figured out a way to raise more money instead of working harder.

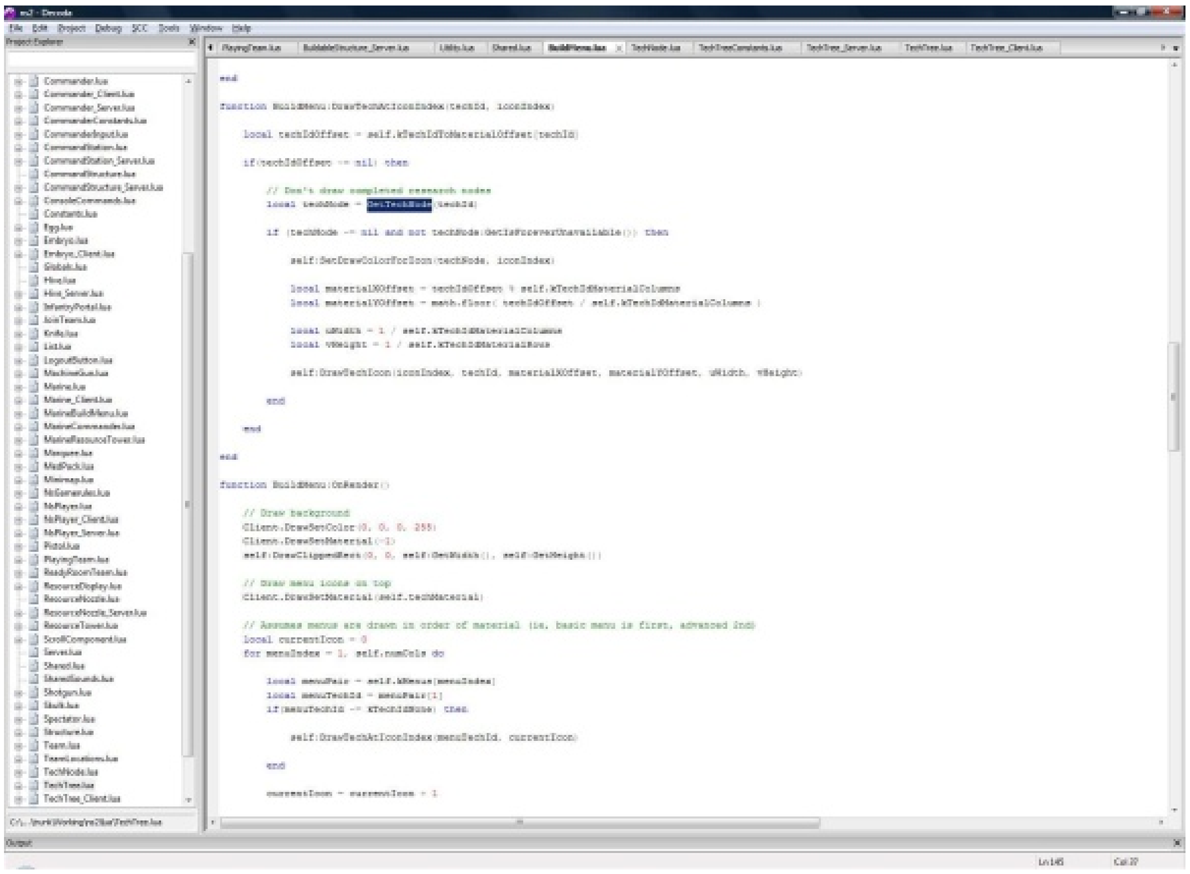
Now we are actively trying to improve our ability to both hit deadlines and make outcomes and goals clear for everyone. We want to constantly improve our efficiency, as well as give more work flexibility to our employees. At the end of the day, employee retention is the most important success factor for us, so we never want people to crunch.

What Went Wrong

1. Project Scope

Natural Selection 2 was an ambitious follow-up to our free independent *Half-Life* mod, *Natural Selection*. Our original plan was to make a quick commercial sequel on the Source engine to start making money. Instead, we spent six tough years developing our own engine and almost going out of business multiple times.

We also decided we wanted to make the most moddable game ever. We did this by implementing 100 percent of our game code in Lua. There wasn't a Lua IDE on the market that we thought was good enough, so of course we spent more time to write our own (called Decoda).



```
end

function BuildMenu:DrawTechAtIconIndex(techId, iconIndex)

    local techIdOffset = self.kTechIdToMaterialOffset[techId]

    if(techIdOffset ~= nil) then

        // Don't draw completed research nodes
        local techNode = GetTechNode(techId)

        if (techNode ~= nil and not techNode:GetIsForeverUnavailable()) then

            self:SetDrawColorForIcon(techNode, iconIndex)

            local materialXOffset = techIdOffset % self.kTechIdMaterialColumns
            local materialYOffset = math.floor( techIdOffset / self.kTechIdMaterialColumns )

            local uWidth = 1 / self.kTechIdMaterialColumns
            local vHeight = 1 / self.kTechIdMaterialRows

            self:DrawTechIcon(iconIndex, techId, materialXOffset, materialYOffset, uWidth, vHeight)

        end

    end

end

function BuildMenu:OnRender()

    // Draw background
    Client.DrawSetColor(0, 0, 0, 255)
    Client.DrawSetMaterial(-1)
    self:DrawClippedRect(0, 0, self:GetWidth(), self:GetHeight())

    // Draw menu icons on top
    Client.DrawSetMaterial(self.techMaterial)

    // Assumes menus are drawn in order of material (ie, basic menu is first, advanced 2nd)
    local currentIcon = 0
    for menuIndex = 1, self.numCols do

        local menuPair = self.kMenus[menuIndex]
        local menuTechId = menuPair[1]
        if(menuTechId ~= kTechIdNone) then

            self:DrawTechAtIconIndex(menuTechId, currentIcon)

        end

        currentIcon = currentIcon + 1
```

NS2 is all written in Lua script. (Click for larger version)

The final product was more commercially successful than we had originally hoped, but that came at a big cost. The game was too large in scope, took too long, and because of that, ended up being very high-risk.

If we could do this over, we would have reduced the scope of the game significantly, or made a smaller game, so we could get cash flow positive sooner. We certainly wouldn't have made our own technology as part of our first product. We bit off more than we should have, and the jury is still out if the engine flexibility, modding, or lost time will make up for it.

2. Ignoring Team Problems

We have a completely unstructured environment, without task lists, priorities, managers, producers, or even job descriptions. We rely on people to figure out the best thing for them to work on, and how to do it efficiently.

Early on, we hired some talented folks that didn't fit in. They had the skills and we liked them as people, but we now realize that they were demoralized without structure. We were so consumed with making the game that we didn't notice these team problems -- only that morale was low. We would've been better off had we realized that personality fit and talent aren't enough: people need to mesh with your working style. When people are unhappy, it spreads through the whole team.

The original happy team.

Now, when we interview, we try to determine from the candidate's personality and experience if what kind of work environment and structure they expect. We tell them that we are completely unstructured and it's up to them to figure out what they should be working on: not a manager or producer. If we get a blank stare, grimace, or nervous laughter, we know the candidate is probably not a fit. We are also being more proactive about asking people what they want to work on and how they want to work, and trying to accommodate both.

3. Inexperienced CEO

We were perpetually broke. There's no other way to say it. And as CEO, it was my responsibility to make sure the company had proper funding. But as a first-time CEO, my primary skill and interest lied in game design and development -- not finance, nor fundraising. But we were learning about fundraising as we went along.

Unfortunately, not having any money is not a good position to be in raise more money. Legal expenses were exorbitant, so we handled the contracts and paperwork on our own as much as possible. And every time we started to run out of money, it was a huge burden to drop everything and try to raise more. It was like a wound that would never heal.

Common startup advice says you should raise twice as much as you think you need. In our case, three or four times would've been better. The same work is required to raise $1M as it is to raise $100k -- you just have to talk to different people. No matter who you talk to, you have to convince them you're going to make them a profit, and you're still in foı months of negotiating and legal work.

I often say that starting a company is similar to getting your MBA: it takes about the same amount of time and money (more, in our case) and will teach you everything you need to know to start a business successfully. Maybe your first business won't be a success, but at least you weren't in school, where there's not even a chance of success.

Looking back, I spent most of my time working on the game, not pitching to investors. If you decide to raise money, jump into it completely. Pitch constantly and refine your technique every time. Constantly meet prospects and work your contacts. Get multiple parties interested so you can leverage their offers against each other. Make sure each party knows you're going to get funding with or without them. In short, do it all at once and make it count.

Next time I would probably look for an additional partner who is an experienced fundraiser and CEO so I could stay focused on what I most cared about: the game.

---

4. Optimizing Late

Donald Knuth is famously quoted for saying, "preoptimization is the root of all evil". Even as relatively experienced game developers, we have always taken that to mean: optimize after you've implemented most of your game. We've since learned that this couldn't be farther from the truth.

It wasn't until close to release that we realized we had some fundamental engine optimizations to make that we should have caught earlier. So we ended up making some massive technical changes within a week or two of launch. This included making the client prediction multithreaded (!) and making fundamental changes to networking and our server update rate.

After seeing even tiny code changes cause huge problems in playtests, the idea of making these fundamental engine changes right before ship made our stomachs churn. But we knew the game needed these performance boosts, so we kept a clear head and made the changes. Miraculously, they went over beautifully, and we had no horrible problems on launch.

But we now understand the father of algorithmic analysis to mean something closer to: "Don't optimize your system before that system is done and can be properly evaluated." From now on, we will measure and optimize individual systems as they hit functionality milestones, instead of waiting until everything is in.

5. Launch Press

We launched *Natural Selection 2* on Halloween of 2012 -- exactly 10 years after the original *Natural Selection* mod. We didn't plan on hitting this date; it just happened. It was the only open slot on Steam before the deluge of high-end holiday shooters. We were unable to predict this day more than a few weeks in advance, so we didn't have much time to prime the press.

As a result, we didn't get much press coverage or reviews on launch, and it proved difficult to get covered after as well. We wrote approximately 600 members of the games press and dozens of YouTube channel owners. Each received a copy of the game, a press kit and an invitation to play, review, or interview us. After a full month, we finally had about 35 reviews published, but not many that were recognized by Metacritic. Our rabid fan base then went wild over a poor review and a subsequent Metacritic controversy, but that wasn't the kind of attention we were looking for.

With such a long development cycle, it was very difficult to keep the excitement or attention of the press. We are pretty inexperienced in the PR and marketing aspects of game development. Maybe we shouldn't have released our early videos six years in advance, or maybe we should've gotten a PR company to help us. It was so difficult to make a good game on new technology that we didn't have any energy or resources left to promote it properly. By the time the game was done, there was no time left for promotion anyways, as the holidays and big releases were going to blow us away.

We did learn that press people are totally overwhelmed with e-mail and that they appreciate phone calls. It's not enough to e-mail and expect them to cover you: always call. We found 37signals' Highrise to be a good way to keep contact info in one place and our efforts coordinated.

Conclusion

We are now figuring out how best to support *Natural Selection 2* with new maps, weapons, etc. while also allowing us to start working on a new game. While the road we traveled was hard, we consider ourselves extremely lucky that we never had to modify our vision for the game to fit a marketing plan, publisher desire, current trend or brand. The game we set out to make is the game our players (eventually) got.

A final anecdotes sums up our experience.

When we were a cash-strapped company trying to figure out how we could show *NS2* at PAX East, we couldn't believe how expensive it was to have a professional booth built. Instinctively, we decided to build our own, out of plywood. To make our hastily erected, unpainted plywood planks look intentional, we asked conference-goers and *NS* community members to doodle on them, permanently inscribing their forum handle on the structure that was literally holding up our game.

By relying on our community and our instincts, we had a great show. It was more work and stress than we had originally anticipated, but everything worked out fine, and no one died or got sued -- just like creating *Natural Selection 2*.

Development Stats

Developer: Unknown Worlds Entertainment, Inc.

Publisher: None

Release Date: October 31st, 2012

Platforms: PC/Steam

Number of Developers: 7 in-house, 11 contractors

Length of Development: 6 years

Budget: $2.9M ($1.1M pre-orders, $1.8M investment).

Lines of Code: 250,000 C++ (Spark engine), 116,000 Lua (NS2)

Development Tools: Microsoft Visual Studio 2005, Decoda, Adobe Creative Suite, 3D Studio Max, Dropbox, Perforce, Subversion, Balsamiq Mockups, Google Apps/Chat, Campfire, Pivotal Tracker, Notepad++, OpenOffice.

Return to the full version of this article