Postmortem: Naked Sky Entertainment's *RoboBlitz*

By Naked Sky Entertainment **gIntroduction**

People often ask us: "How did Naked Sky, a small, independent game development studio, get into the business of making such an ambitious game for Xbox 360 Live Arcade?"

Well, it all started with a technology demo for the Intel dual-core processor. Developed in eight weeks and showcased at the 2005 IDF and GDC, the *RoboBlitz Tech Demo* (aka *RoboHordes*) was an Unreal Engine 3-based, single-level game that featured entirely physics-driven game play.

When people approached us at GDC, asking questions like, "Who is publishing this?" and "When is it coming out," we realized we needed to develop it into a full-fledged game. Unfortunately, the original game was built purely as a tech demo, so in order to develop a robust and engaging commercial product, we had to rip out and re-implement pretty much all the code, art, and design.

When it came to the question of how to go about distributing the game, we were confronted with several challenges. We wanted to keep control of the intellectual property while staying independent, which steered us away from the traditional publishing route. Thankfully, we met with the Live Arcade group at Microsoft when they were in the early stages of forming their plans for the upcoming Xbox 360. It was the perfect platform for *RoboBlitz*, as long as we could keep our title under 50MB.

Because we decided to go the self-funding route, we didn't enter full production until November 2005, after raising private funding and getting an office. The game was finished eleven months later. As an indie developer, we pretty much encountered every obstacle imaginable in the process. Fortunately, we pulled through and are now the proud parents of *RoboBlitz*!

RoboBlitz

Publisher:
Microsoft (Xbox 360)
Naked Sky Entertainment (PC)

Developer:
Naked Sky Entertainment

Platforms:
Xbox 360 Live Arcade, PC

Number of Developers:
~12

Length of Development:
11 Months

Release Date:
November, 2006 (PC)
December, 2006 (Xbox 360)

What Went Right

Great Team

Our biggest asset at Naked Sky is the people. We believe you can achieve *anything* as long as you work with the right people. This doesn't necessarily mean the smartest or most experienced people, but passionate, talented, dedicated, hard-working, and good people. In short, people with great potential. Everyone on the Naked Sky team possesses these qualities, and a whole lot more.

About half of our team is composed of recent college graduates that majored in game art or programming. This works in our favor financially, because as a small indie, we can't yet afford competitive wages for industry veterans. Of course, you can't have a half-green team without capable leaders to guide them, and we're very fortunate to have smart and savvy leaders spearheading our team. They're very hands-on in training the new hires—not just when demonstrating new tools and techniques, but also when illustrating different problem solving methodologies. Through their training, everyone has become self-sufficient and is capable of taking up any urgent or unexpected responsibilities from their team lead.

Our team is very tight—we're not just a company, but a family, or *famiglia* as we call it. We look after each other and we help each other out. We've always believed in and trusted each other, and we've built up a sense of camaraderie through both hard work and hard play. We have ping pong and *Street Fighter 2* matches daily, and outings to celebrate our major milestone completions. We also load up on healthy Costco snacks for our "growing boys."

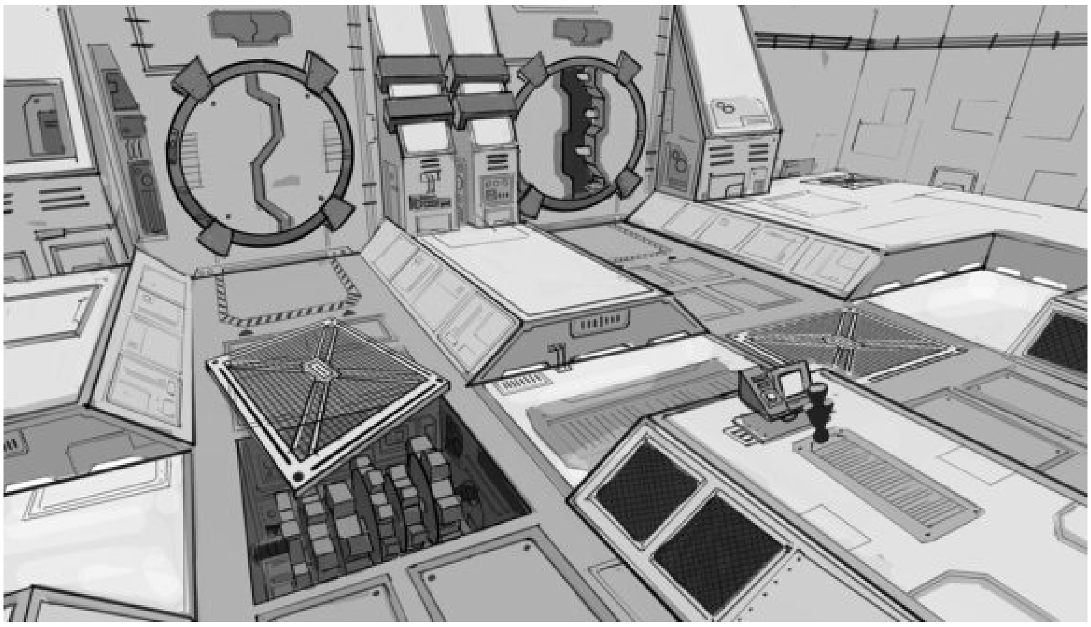
Together, we laughed through the good times and jumped over the challenging hurdles. Because our team was treated well and everyone was happy working together, magic happened.

Risk in Innovation

We made a lot of technological gambles when creating *RoboBlitz* and luckily, they're all paying off. Early on, some of our friends in the industry said that building a game based on completely physically driven characters was impossible for a new team to pull off. After much hard work, we did manage to pull it off and it allowed us to implement innovative weapons like the P2P Beam, a band of elastic energy you can use to connect any two moveable objects in the game.

Because all the characters are driven by physics, it means you can use the P2P to rubber band three bad guys together, and watch them squiggle around like a giant worm while you pick them off one by one. Or, you can spring them into a giant shredder or one of many other spots where a baddie wouldn't like to be. The P2P, along with many of our other unique tools and weapons, wouldn't have been possible without our physical animation system.

We're also getting a lot of press for being one of the first Unreal Engine 3 games out, and that's another significant tech risk that is paying off. Developing the game at the same time UE3 was being developed was non-trivial. The engine is great, but it wasn't fully functional when we started work on *RoboBlitz* a year ago. We were putting all of our eggs in Epic's basket by relying on them to come out with features we needed by the time we needed them. Sometimes we had to implement the features ourselves while we were waiting, but in the end, this too was worth the risk. With zero marketing budget, being the first UE3 game to ship was a great publicity tool.



Finally, shipping a 19 level next-gen game in less than 50MB took a lot of clever tricks and techniques. From integrating the untested *ProFX* into our game, to making sure all animation was procedural (and thus very well compressed), to tweaking our package compression routines, to looping sections of our music just right, to the masterful UV kung-fu of our artists – it took a lot of hard work to keep our game under 50MB.

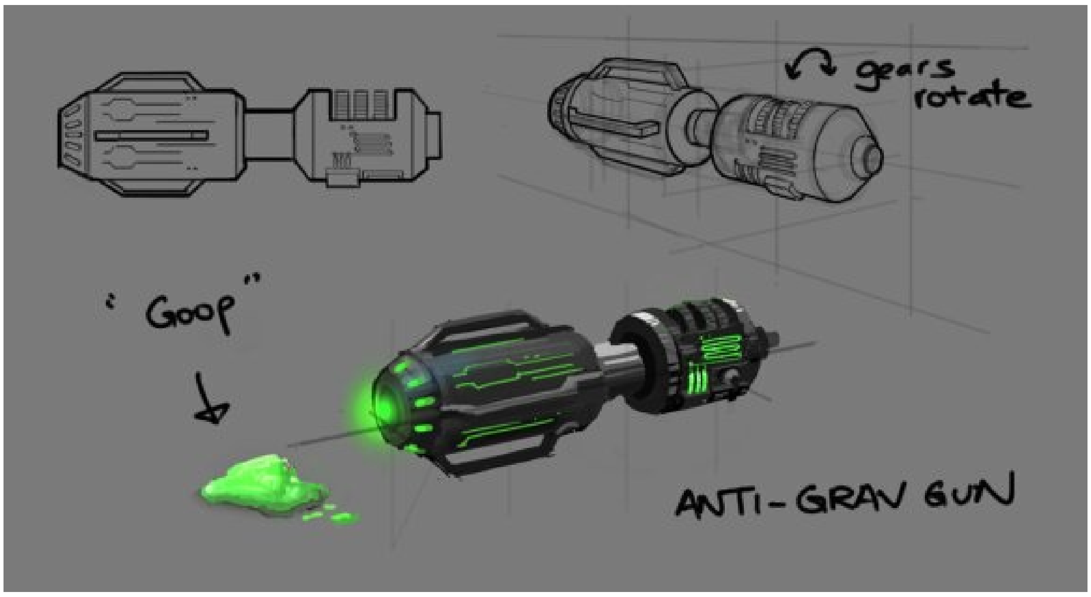
It helped tremendously that we planned for this right from the start, but there were definitely times we weren't sure we were going to make it. It was immensely comforting that last day when we built up our final package and it came in just shy of 50MB. Since most of the buzz we're generating is around our Xbox Live Arcade release, we feel the risk here was very worth it and all that hard work on file shrinkage is going to pay off.

Licensing Good Tech

Two of the smartest choices we made on this project were licensing Epic Games' Unreal Engine 3 and using Allegorithmic's ProFX procedural texture generation tool.

UE3 is simply beautiful. It has the capability to power next-gen games, and it's designed to be modified and extended in any way its licensees need. None of our programmers were graphics experts coming into the project, so licensing the engine gave us a huge leg up graphically.

However, it's not just the feature set which makes the engine great, but the team behind it. The tech guys at Epic are very friendly and supportive, and just a pleasure to work with. Building a completely physics based game around UE3 took some extreme modification to the engine, but whenever we had a question about the inner workings into which we were hacking, someone at Epic was always ready and able to help us out.



Allegorithmic's ProFX procedural texturing tool might have a little less publicity than UE3, but it was instrumental in shipping *RoboBlitz* for the Xbox Live Arcade platform. As we mentioned, Live Arcade has a 50MB limit for their games, and there's no way all the textures in *RoboBlitz* could have fit within that limit without procedural generation.

Using ProFX, we were able to generate 80 percent of the textures for each level at load-time, which allowed us a much greater palette of textures when decorating the levels. This helped tremendously when it came to squishing 19 levels of a 3D, next-gen game down into 50MB. Similar to our experience with Epic, the Allegorithmic team was also very helpful and knowledgeable. It was great working with them.

Self-funding the Project

Making a game on your own dime is not for the faint of heart, especially when it's a next-gen title using the latest work-in-progress technology, based on an original IP, and incorporating innovative ideas from a team that has never collectively shipped a commercial game. So why did we do it?

The answer is simple, because we knew only we could make *RoboBlitz* the way we envisioned it. To have complete creative control over this project, we had to fund it ourselves. Plus, given all the risks involved, no publisher would touch us with a ten-foot pole.

The great thing about funding your own game is that you can be as creative, unconventional and bold as you want. Would any publisher greenlight a game featuring an orange robot rolling around on a giant metal ball? What about replacing all hand-animation with physics-driven movements and behaviors? Now try this one: would anyone have believed us if we said a year ago that we could make a 19-level, Unreal Engine 3 game in less than 50MB?

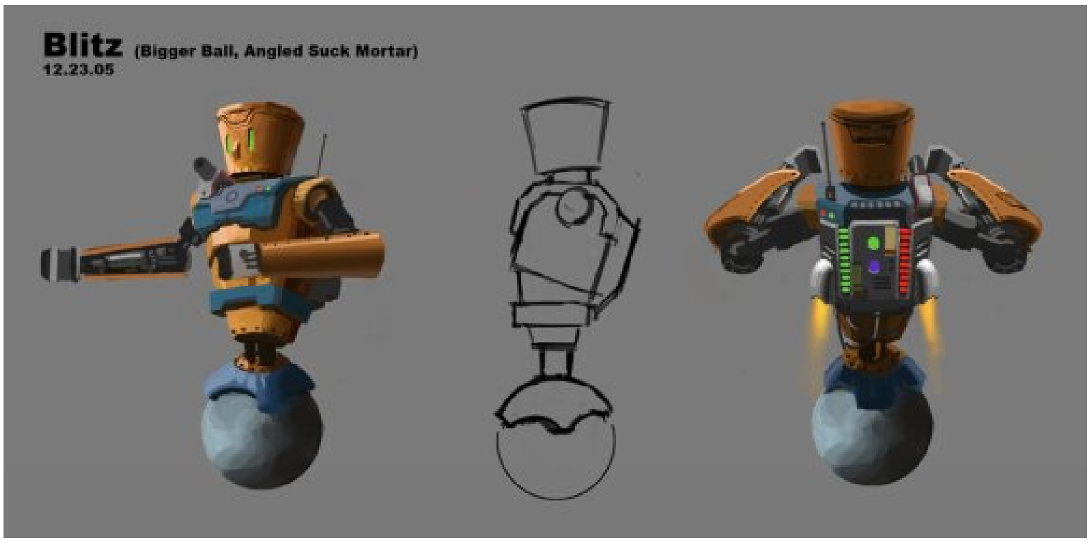
Working with unfinished cutting-edge technologies always leads to production delays, unforeseen design/art/code changes to accommodate a new build of the tech, and more production delays. Unless we were working on an established franchise, any publisher would have aborted *RoboBlitz* a long time ago. But since this is our baby, she was born a healthy 49.32MB!

Lastly, because we self-funded the title, we were able to negotiate a great revenue split with the publishers and distributors.

Supporting the Community

Several months before launch, we opened a forum on the *RoboBlitz* website where users could discuss and ask questions about the game. This helped generate viral interest, build up a user base with our extremely limited budget, as well as provide a way to quickly spot and address technical problems once the game was out. Since launch, we've been very protective of that user base, and have dedicated over a third of our tech team to technical support e-mails and forum moderation.

Shipping the *RoboBlitz* editor and immediately setting up an editor Wiki at our site were also strong community building steps that have gone over quite well. Right now our Wiki is still the number one public source for info specific to a UE3 based editor, and that helps generate traffic and interest in *RoboBlitz*. We're also actively engaging the modding community, encouraging people to push the limits of physically based game play.



Providing an online forum for the *RoboBlitz* community really paid off. At launch, we learned that people with certain graphics chips were having trouble getting the latest drivers required to work with our game. However, members from our growing user base quickly stepped in with hacks and tricks to solve these problems. Some of our more dedicated users even volunteered to moderate our forums. With all the users helping each other out, the *RoboBlitz* community is taking off.

What Went Wrong

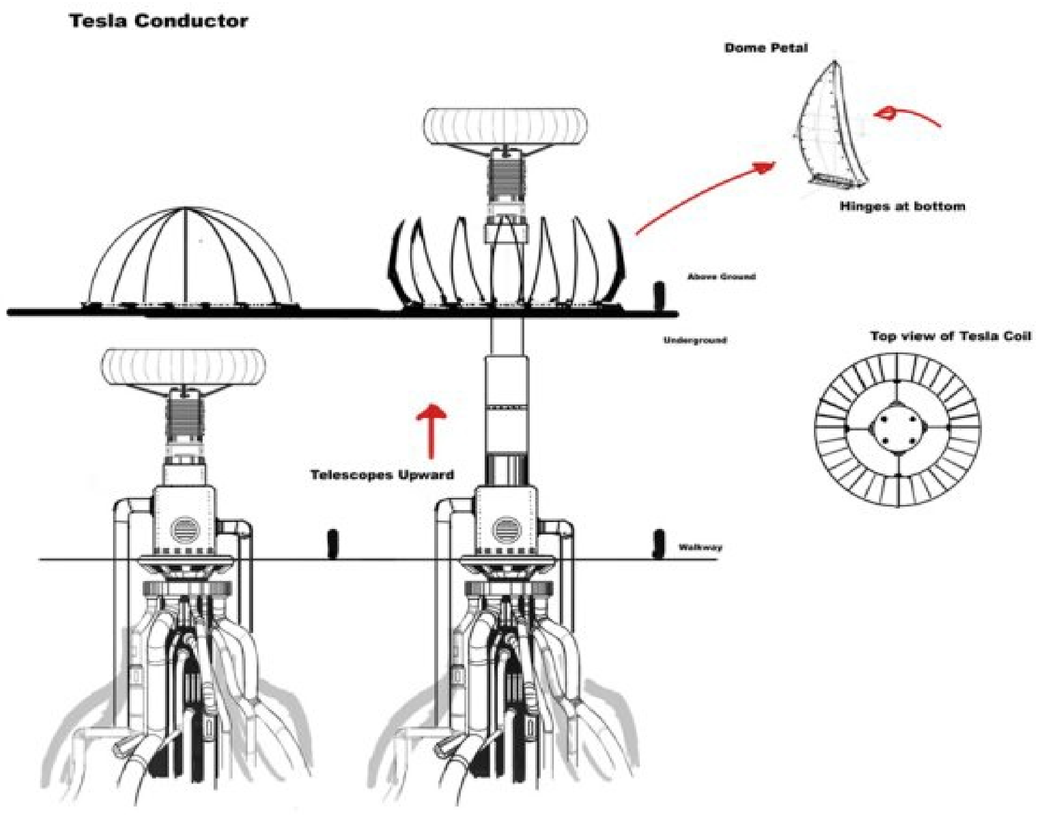
Lack of Pre-Production

When we started development on *RoboBlitz*, we jumped into the level creation process with very little pre-production. We knew the basic functionality of our robot and we knew we wanted action and puzzle oriented game play.

In general, this put us in a mode of running along with whatever good ideas we had at the moment and keeping whatever worked best. This allowed for very fast, dynamic changes in the game design, which would have been fine if we were in pre-production. However, when we were adding new weapons and controls six months into an eleven-month project, it did not work out too well.

The biggest problem with our lack of pre-production was that we didn't nail down key game play mechanics and character movements before implementing the levels. Once the majority of the game was built, it became extremely burdensome whenever we had to add new features or tweak character movements. This meant we had to go back through the entire game and alter levels so that the new features didn't break any of the existing ones. With a product that allows as much freedom of movement and game play as *RoboBlitz*, this was not a small task.

We were left changing significant play mechanics, controls, camera angles and level features only weeks before the content complete milestone. These new changes then required retranslation of localized text, sound effects tweaks, and art adjustments, which in turn generated their own regression issues. As a result, a few areas and aspects of the game were much less thoroughly tested than we would have liked.



Over Ambition

Even though we had great people on our development team, there were only a few who had gone through the full development cycle for a retail title before, and this had far reaching repercussions.

On the most basic level, we were not able to accurately estimate the impact of untested technology on the time it would take for us to implement the game. Therefore, because we had tons of great ideas, we chose a scope for the game that included more features than any sane, experienced studio would dare accept given the budget and timeline.

Tesla_control_room_texture_guide
1. control button
4. pannels
2. ceiling light and reflection
5. floor
3. wall
open space, Blitz can jump out

Because we underestimated the time and budget required for a game of this scale, we were understaffed right from the start. This required us all to wear several hats, sometimes leading to negative consequences. For instance, our lead programmer was also our system admin. Whenever there was an issue with the network or the bug tracker or the version control software, he had to stop programming to take care of it.

Our overambitious design and understaffing sent the entire team into crunch mode for the majority of production. We had no choice but to work long hours to make deadlines, and this took a toll on our constitutions and quality of life. Luckily, every one on board was young and hearty, and in love with the project, so there were no permanent effects, but we will definitely do things differently from now on.

Underestimating the Trouble with New Technology

Building a game based entirely on physics is difficult. Building a *fun* game based entirely on physics is OMGWTFBBQ difficult! The problem we encountered was that reality just wasn't very entertaining, so we were constantly balancing between "real" physics and "fun" physics.

One example is the way Blitz drives – it took weeks to balance the programming and tweak the parameters so that he'd drive as smoothly as he does now. He still doesn't move around like Mario, but in a game where everything is physical, that's just not possible.

Blitz's grabby arms were another sore spot. We re-wrote the code 14 times until we came up with something that was user-friendly and enjoyable. The key there came down to building several A.I. heuristics into Blitz's grabby routine. He'll actually auto-adjust his alignment to pick things up better, and if something is too low for him, he'll bend down just the right amount to pick it up. We decided to implement this feature because it's just not fun for the player to have to do everything manually. We believe using A.I. to drive physical animation is going to be the future of procedurally animated games, but we didn't realize it until late in the development process and it cost us quite some time.

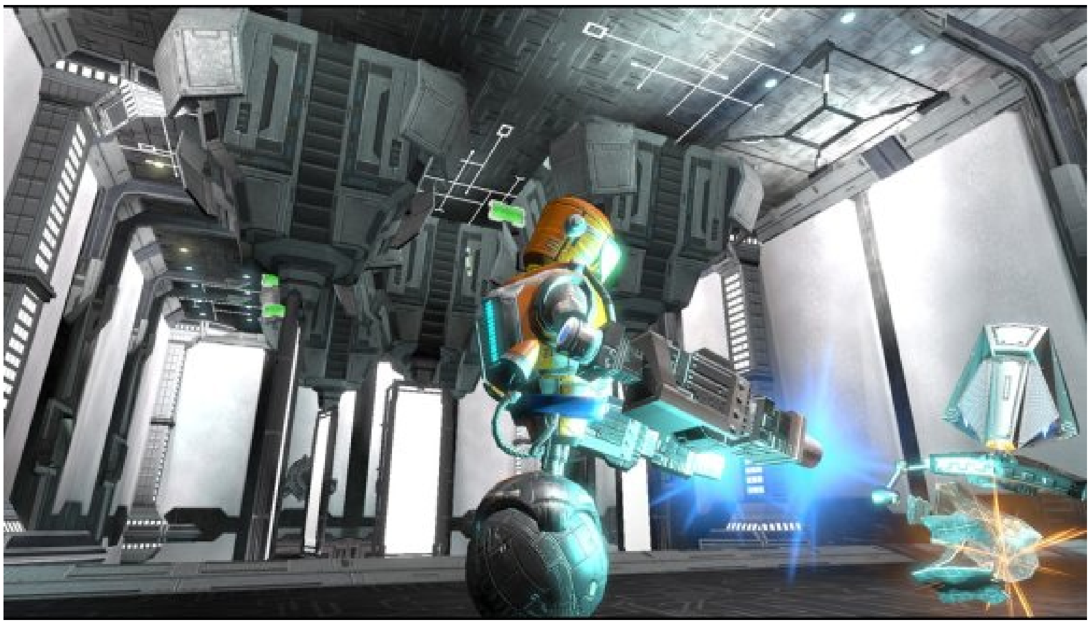
On top of all that, building a physical simulation using an engine still in development made things a whole lot more challenging. We began development using version 2.3 of AGEIA's PhysX tool, and we found that a lot of their features just weren't working the way we needed them to – usually because no one else was using those features in their games yet.

AGEIA was responsive to our needs and tried hard to get things functioning the way we needed. However, every time a new build of the engine came out, we had to re-adjust and tweak all of our old settings, and work around any regression bugs in the betas. All the unexpected iteration work hurt us when trying to stay on schedule.

Ran Out of Money

Cash flow, cash flow, cash gone! This is not a recommended scenario for any game developer, or any business for that matter. Unfortunately, this is what happened to us. The game was delayed by multiple months due to all the production and technical hiccups mentioned above. Because we weren't willing to put out an inferior product, we just kept going until we were satisfied with the quality.

This, of course, led to an empty bank account and an over-stressed management. Fortunately, we were able to borrow some money from our family members and close friends. Even so, calling up your ex-girlfriends for $20K is not something you'd want to go through, unless your skin is as thick as ours.



To avoid or at least minimize such a scenario, the rule is to always raise more money than you *think* you need. Trust us, you'll need it, and more. We raised several hundred thousand dollars to fund *RoboBlitz*, which even included a significant buffer (or so we thought), but we still ended up needing more to take the game to completion.

Lack of In-House QA Lead

Because we outsourced our testing to a professional testing company, we neglected to hire an internal QA manager, and this proved to be one of our biggest stumbling blocks. The external company did a fine job, but without a QA lead in-house to manage our own testing process and make suggestions to the outsourced team, our testing was not as directed and effective as it could have been.

Halfway through the project, we hired an in-house testing intern and even recruited our art team to help with the QA process, but they faced a similar problem – without any structured guidance, they were just testing blindly. Because of it, we were still finding and fixing not-so-minor bugs up until a week before our PC release. This never would have happened had we had an in-house QA lead.

In addition, we should have developed a rigorous, automated test suite. Without automated testing, every bug fix had a greater potential of introducing hidden bugs into the game, and this cost us a lot of time when regression issues weren't identified until long after they were introduced. The lesson here is that we needed to have at least one experienced QA person in the company dedicated to nothing but testing, no matter how good our external testing solution was.

Conclusion

Passion, talent, great partners, and perseverance are what ultimately enabled us to realize the dream of *RoboBlitz*. Looking back, we were fairly inexperienced going into this project and learned a bit more than expected, but that's the way it goes with inexperience – you don't know what you don't know.

Despite all the setbacks, our conviction and dedication were strong, and we managed to pull together a game of which our entire team is proud. We hope everyone will enjoy playing *RoboBlitz* as much as we enjoyed making it, and stay tuned for our upcoming physics-based multiplayer madness!