Postmortem: Vector Unit's Hydro Thunder Hurricane

PRINT

By Matt Small

[EA and Stormfront Studios veteran Matt Small shares successes and defeats in his small team's transition from large studio to small startup while developing Hydro Thunder Hurricane for Xbox Live Arcade.]

"So I've been having these thoughts... about quitting and doing the small games thing."

That line popped up in my chat window on a night in the summer of 2007. My pal Ralf Knoesel and I had been working on a little side game project, just for kicks. Or at least we'd been *talking* about working on it. The problem was we had regular paying jobs. And what we liked to think of as lives. Progress was achingly slow.

Every developer I know daydreams about breaking away at some point to work on their own game. Ralf and I had put over 12 years apiece into established companies, most recently me as an art lead at EA Redwood Shores and Ralf as a programming lead at Stormfront Studios. We had a little money saved up, and we were tired of working for The Man. Ralf's revelation was all the push I needed.

By January 2008 we'd quit our jobs, liquidated every personal asset we could live without, upgraded our PCs, and incorporated a new company -- Vector Unit. We absorbed details we'd never had to think about as employees: the nuts and bolts of small business management, contract negotiation, group health plans.

The game we set out to make was designed from the start to be a small-scale but big-attitude water racing game for XBLA or PSN, a throwback to classic arcade racers like *SF Rush,* and of course, *Hydro Thunder.*

We went with water because we love the emergence you get from the dynamic surface, and aside from minigames in titles like *Wii Sports* there hadn't been a decent water racing game in about 10 years. Plus we'd worked together years before on the boat combat title *Blood Wake* for the first Xbox. We figured we'd start with something we knew we could make.

Six months later we had a game engine, a playable PC prototype, and a PowerPoint pitch that we'd practiced in front of friends so many times I could recite it in my sleep. We shopped it up and down the West Coast publishing gauntlet, and eventually caught the interest of Microsoft Game Studios.

It was through our conversations with Microsoft that the idea of tweaking our design to fit the *Hydro Thunder* license first came up. We believed in our prototype, but we also knew that a known license would bring much-needed publicity to our first title as a new studio.

MGS acquired the distribution rights from Warner Brothers, who had just purchased the license from the recently defunct Midway Games. We rewrote our design docs, increased our scope threefold, and lawyered our way through the contract negotiations.

In April 2008 we finally signed the deal and received our kickoff funding and our first 360 dev kits. *Hydro Thunder Hurricane* was off and running.

Well, "running" might be an overstatement. We still had to hire a team, find an office, prove out our new design in an updated prototype, and get our PC engine running on the 360.

When we started Vector Unit, we figured the experience we had as leads managing large teams at established game development studios would scale smoothly to the development of a smaller game project. If anything, we thought, it would be easier -- fewer moving parts. To some extent that ended up being true. What we didn't count on were the countless small surprises that small game development had in store for us.

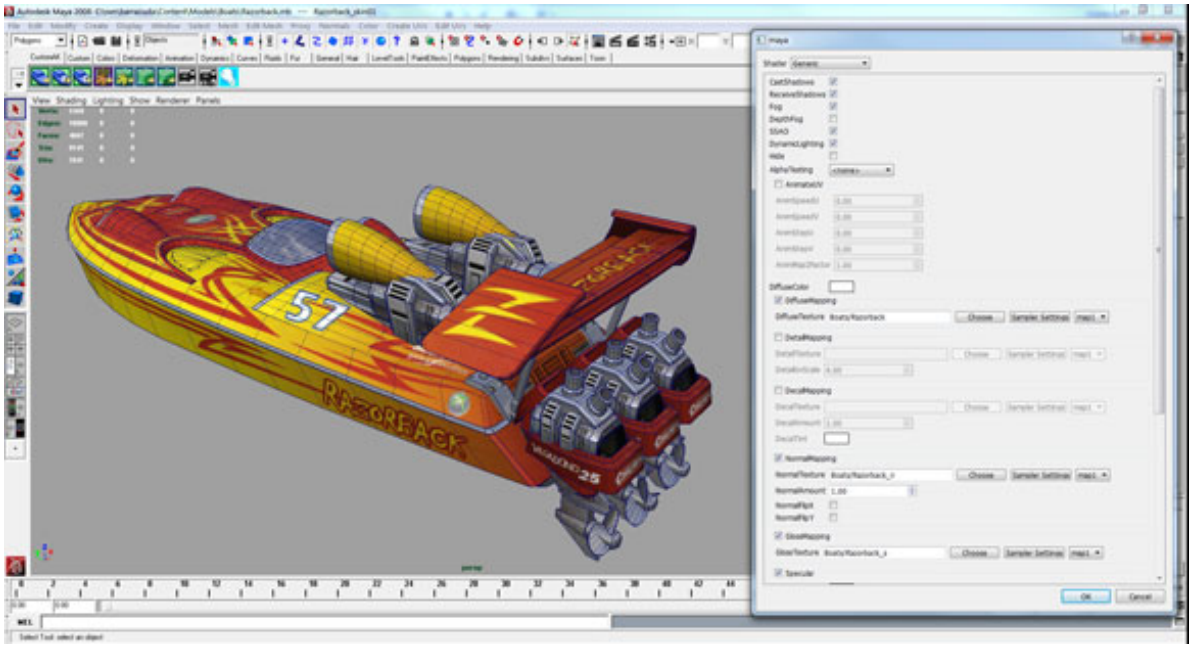
What Went Right

1. In the beginning there were Tools

When we began working on our original prototype, the very first technical task -- before the game engine itself -- was a suite of production tools. We knew that content creation would be our critical path, and our plan was to keep our core team small and outsource the bulk of the art.

We needed a pipeline that would be easy for contractors to pick up and learn, allow for rapid iteration with minimum build times, and be adaptable enough that we could add features as needed without having to rebuild existing assets.

We researched a number of third-party game development suites but decided against them. Engines like Unreal were way out of our price range and delivered more power than we really needed. And the indie suites like XNA and Torque didn't give us the low-level control that we figured we'd need. We decided early on to roll our own, and it proved to be one of our best decisions.



(click for full size)

The base art asset pipeline came first. Ralf developed a data-driven shader system that integrated into Maya with a shader plugin supporting full-featured hardware rendering in the shaded viewport using the game shader code. Artists could add shader layers like normal or detail mapping and see the results of their tweaks immediately, iterating freely

without having to export into game. Over the course of development this simple feature saved us weeks of development time, as our artists could work in a way that felt immediately natural to them.

The game editor, PFX editor, and other tools all export to a common format (xml or json), which the game itself builds at runtime, baking updated assets on demand. For the artists, this means that even for our most poly-heavy and texture-rich assets, the time it takes from hitting the export button in Maya to flying around the assets in game was never more than 30 seconds and usually much less. The tools also support live telemetry, which reduces the need to re-build when tweaking any parameters.

Finally, throughout development we maintained a working PC build of the game. Although final assets were always tested on the 360 and evaluated on TVs, the PC build freed us up from having to provide dev or debug kits for outsourcers, and also made it easy for team members to work at home and maintain flexible hours.

I should mention also that we didn't build everything ourselves: We used FMOD for audio, Bullet for collisions and rigid body physics, and the truly wonderful Subversion (specifically TortoiseSVN) for version control.

---

2. I'm on a boat!

I mentioned earlier that we're big fans of water-based gameplay, and we knew from the start that nailing the feel of powerful boats smashing across undulating waves would be key to the game's success.

The fluid dynamics simulation was the first part of the game engine to come online. It's based off buoyancy and flow calculations modeled around a low-poly "fluids" hull, and fairly accurately models the behavior of real boats in water.

This accuracy enabled us to create a compelling variety of boat control models based on the shape of the hulls -- the V-shaped speedboat hull rises convincingly out of the water when you accelerate, catamaran pontoons maintain lateral stability in turns, and flatbottomed hulls hydroplane across the surface at high speeds.

One thing we learned early on however was that reality was just too... well, real. Actual offshore powerboats typically top out around 100mph in calm waters, but we needed our boats to exceed 200mph in high waves.

We added artificial downforce, reduced water buoyancy at high speeds, and tweaked countless other numbers until we were able to achieve the breakneck speeds required for an over-the-top arcade racing experience.

We began user testing early in the prototype phase, grabbing any friend who'd spare the time and sitting them down with a controller and no instructions on how to play. The first experience was humbling. For weeks we'd been racing comfortably around our test track, shaving quarter seconds off each others' best times. Our first test subject spent most of his game slamming back and forth from one wall to another.

It's a classic design mistake, and one we had to relearn: designers get too close to the control scheme, and eventually acclimate to its idiosyncrasies. Multiple iterations later, we finally arrived at a simplified turning model that felt realistic enough but was intuitive for a newcomer to control.

The driving mechanic in our first prototype was still a long way from the one that eventually shipped in *HTH*, but it was close enough to prove out our concept and convince the publishers we pitched that the game would feel good off the stick. And the lesson we learned about user testing stayed with us throughout development, prompting us to constantly tune and improve the mechanic. We're proud of what we delivered. As one reviewer put it: "It handles like butter."

3. Making friends with The Man

It may sound like butt-kissing, but it would be remiss to not list this as a "Right": Microsoft Game Studios was a great publishing partner.

A bad relationship with a fickle publishing partner can sink the most well-laid-out plans. We've seen talented indies flounder when publishers pivot on design mandates, or hold milestone payments until the last minute that they're contractually obligated to pay.

MGS was great. Our producer gave us the creative freedom we wanted, and offered editorial feedback that actually made the game better. He was always willing to talk and compromise on points where we disagreed. The MGS team agreed to work with our SCRUM-lite scheduling methods, without requiring us to maintain intricately detailed charts and diagrams. And -- most wonderfully -- they always paid promptly on milestone acceptance.

Which is not to say we got all this for free. Managing the publisher relationship takes conscious effort. Above all, we went to great pains to avoid any red flags: we delivered milestones on time, and tested them thoroughly beforehand to ensure that the deliverables worked as we'd agreed.

Towards the later part of the schedule, we introduced preemptive internal MAT testing to smooth cert compliance. We responded quickly to any issues or concerns that MGS raised. And we made sure to keep delivering a flow of new content and demo-ready builds, so there was always something new and pretty or fun for them to look at or show off at review meetings.

The effort paid off: We never had a milestone rejected, which kept the money flowing and our company afloat. And we kept our producer happy, which gave us the creative breathing room we wanted.

4. PWNage

We had splitscreen multiplayer up and running very early in our original PC prototype. Splitscreen allowed us to playtest our racing experience before investing time in AI, and we tuned most of the game mechanics -- particularly the physical boat wakes and drafting -- to the local multiplayer experience. The experience of racing elbow-to-elbow on our nappy office couch, smack-talking each other and squeaking out photo-finish victories, convinced us that this would be a key appeal for the game.

When it came time to implement the online game we chose Microsoft's QNet framework, which allowed us a quick

first pass implementation on the 360. After that came months of tuning and experimentation to balance the physics and collision testing against the effects of latency and other networking artifacts.

The water physics engine gave us some wiggle room; boats naturally tend to bounce and slip sideways as they move across undulating water, as opposed to cars which travel in a crisp line and quickly reveal the lerping effects caused by network lag. We continued to tweak the client-side physics interpolation throughout the project, and ended up with an extremely smooth online game that I believe rivals any racing game on the market today.

One additional design improvement that came online later in development was the switch to a "loser helper" system in multiplayer, which rewards trailing players with additional boost and a chance to catch up. This suggestion came from our producer, and although initially greeted with skepticism by some on our team (including myself) the results of our first tests spoke for themselves. Races became intensely close and exciting, and although the most skilled players could still win consistently, it was often only by 10ths or 100ths of a second.

The inclusion of four-player splitscreen racing in *HTH* and the support for local players to race online together drew widespread praise from players and reviewers alike, many of whom lamented the absence of splitscreen from modern racing games. For us it was a big win, and will undoubtedly be a part of our plans on all future games.

---

5. Staffing flexibility

With Vector Unit entirely funded by a combination of personal savings and project duration milestone payments, we knew from the start that we'd have to keep our permanent payroll to a minimum and build our game exclusively with the help of independent contractors and outsourcing.

Our staffing plan called for a project-duration programmer in addition to Ralf, a sound designer, and two or three artists to work with me on building and texturing our racing environments. That was the bulk of our project duration staff, what I think of as our "internal" team, although most contributors worked some combination of on-site and off. The boats and the rest of the art work, UI design, and other odds and ends were "outsourced" -- meaning they were meted out in smaller, discreet chunks to individuals or groups that worked only offsite or overseas.

I'd like to say that we planned our staffing needs meticulously, but the truth is we were amazingly fortunate to assemble such a talented team in just a matter of months. In this respect we were helped by the economic meltdown. Throughout 2009 in the SF Bay Area alone, thousands of game development professionals were given their pink slips.

Fortunately this tragedy had a silver lining -- as our project got rolling, we were able to offer employment to a number of experienced friends and former colleagues who needed the work.

As a consequence we were able to ramp up fairly smoothly as the project swelled into full production in fall 2009. When development needs spiked, we added shorter-term contractors in firefighting positions.

As contractors worked closely with us on an almost daily basis, we were able to avoid much of the costly inefficiency and management overhead that sometimes befalls outsourced projects. For purely offsite content creation, we worked primarily with Concept Art House, an art outsourcing group with overseas artists and a local San Francisco office. The availability and responsiveness of that local team further helped reduce our internal overhead.

Because there are only two of us permanently on staff, people sometimes get the impression that Vector Unit is a two man shop. Nothing could be further from the truth. Without the senior-level experience, creativity, and ownership of each of these contributors, *Hydro Thunder Hurricane* would never have happened.

What Went Wrong

1. Small team gotchas

When you're managing a team of 20 artists at EA and one gets sick, it's a speed bump. When you have three artists, it's a 33 percent drop in production.

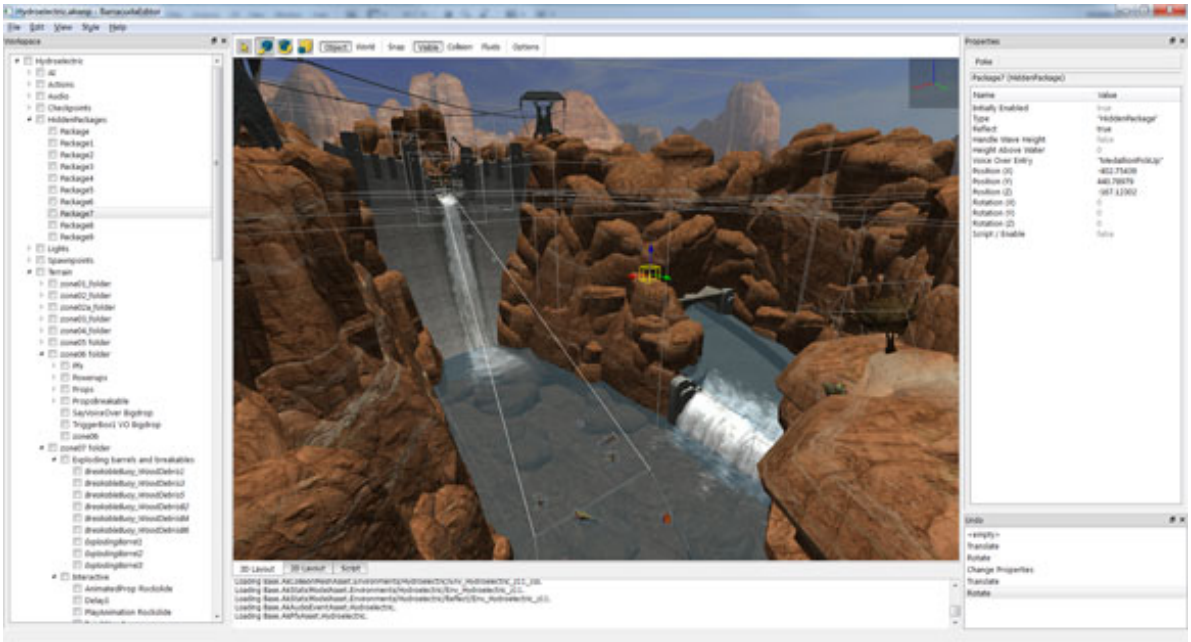
That's one lesson we learned the hard way on our first project as a small studio: the production dynamics are extremely volatile. Productivity fluctuations are a natural part of project managment. People get sick, take paternity leave, quit. Some team members are more productive than others, and some tasks end up taking longer than you predicted.

We were accustomed to juggling resources on larger teams, but we didn't foresee the contortions we'd have to go through to firefight scheduling shortfalls when each and every team member is on the critical path.

We planned for this to some extent when we contracted the services of Concept Art House. Our intent was to offload as much art production as possible onto them, and use them as a pressure release valve for unforseen content needs.

They performed admirably, particularly in modeling and animating most of the boats, but unfortunately we never really figured out how to use them effectively for environment art, which consumed the bulk of our production time. We outsourced as many decorative and non-gameplay-dependent props as possible, but it wasn't enough to fully balance the occasional speedbumps.

In the end, we made up the difference the old fashioned way: we worked our butts off.



(click for full size)

2. QA Meat & Potatoes

Microsoft Game Studios took charge of all of the QA on *HTH*, which they subcontracted out to VMC Gamelabs. VMC stepped into development shortly after our Alpha milestone, in the fall of 2009, and continued to provide coverage through our final cert submission in May 2010.

Generally VMC was great to work with -- we had direct contact with our leads there and at Microsoft, and all three of

us worked together to define initial test plans, coordinate milestone deliveries, and triage bugs during the critical final months of production.

In hindsight, however, their mandate was overly focused on TCR compliance testing, and didn't allow for enough of the meat-and-potatoes coverage required for a fully polished racing experience.

Specifically we should have invested more resources on collision testing throughout each environment, boat balancing, and general usability. We tried to cover these areas on an ad hoc basis at Vector Unit, but when your internal QA department is also your production team -- well, let's just say it wasn't always at the top of our to-do lists.

As a result the game shipped with a number of issues that should have been caught and could have been easily fixed, resulting in, among other things, some highly annoying leaderboard exploits. While these were not noticed by the majority of players, these bugs did diminish the overall polish of the finished product.

On future titles, we plan on hiring an internal QA liaison during the last half of development to supervise internal and external coverage and ensure that these issues don't fall through the cracks.

3. Networking edge cases

Although network multiplayer was a key pillar of *HTH*, and one of the things that went "right", the implementation was much thornier than we'd originally provided for in our scheduling.

The hard parts weren't related to player experience -- dealing with latency, physics synchronization, managing data bandwidth. All these took effort and patience but they were at least predictable and solvable problems that our small networking team (one guy) could test for and solve.

The tricky stuff was all the service-side communications -- talking with the Live servers, dealing gracefully with dropped players, local and network disconnects, etc. When we scoped out our networking schedule we'd assumed Microsoft's native libraries would come with packaged solutions to most of these relatively generic Live-related issues. In practice, we found that we had to engineer most of the solutions ourselves.

Compounding this was yet another small-studio challenge: it's hard to stress test eight-player online gameplay when you only have six people in the office.

VMC did a fair bit of stress testing on their end, and they and the team at Microsoft were sometimes available to fill ou our roster for our internal MP tests. Even so, there were a handful of connectivity bugs that only made themselves known after the game had shipped and we finally had a real-world test kitchen with hundreds of people rapidly joining and leaving games simultaneously.

Fortunately we were able to address these issues in a post-release Title Update, but the occasional glitches caused frustration for some of the very people we most wanted to please: the core players who bought the game early and formed the start of our online community.

---

4. The user experience forest

Hydro Thunder Hurricane is one of those rare games that the team never got tired of playing. Nearly every day we'd spend 30 minutes or more battling it out in multiplayer contests, or trying to best each other's times with the Leaderboard Opponent split-time feature.

Innumerable tweaks and tunings came out of those play sessions -- widen a corner here, adjust a wave height there -- and I credit the playability and fun of the moment-to-moment experience in *HTH* to those many hours of play.

However, in our focus on the playability trees, we lost sight of the user experience forest.

HTH's single player game offers a lot of content for the player to unlock: there are eight tracks, each with a Race, Gauntlet and three Ring Master variations. That's 40 events to unlock and play -- and doesn't even include the Championship seasons.

All these events, and the boats you race with, are unlocked using a credits ladder: placing third or better in an event earns credits, and each item gets unlocked at a specific credit score.

One usability issue that we realized too late is that -- shockingly! -- not all players want to play all the different types of events. Some just want to race. Some love Gauntlet but hate Ring Master -- and vice versa. Unfortunately the game's linear unlock system pushes you to play everything, whether you want to or not.

Another issue is that the difficulty ramp is too broad. The early Novice events are just too easy for hardcore racing fans, while more casual players (and even some hardcore) were sometimes frustrated by the punishing difficulties at the Expert levels.

So while the game is full of fun experiences for every type of player, not all events and difficulties are fun for all players -- and the game steers all players towards all events and difficulties.

As it was, by the time we lifted our heads out of crunch, it was really too late to do anything about the overall structure of the game. If we had set aside enough time internally to play the game through from beginning to end -- or better yet invested in more external long-format usability testing -- I believe we would have noticed and had time to address these issues, and ended up with a much stronger experience.

5. DLC Forecasting

To pass Microsoft Certification testing we needed to include "placeholder assets" for representative downloadable content (DLC), and we delivered that to the letter. Unfortunately we didn't foresee the full feature set that we eventually decided to include in our first expansion (the "Tempest Pack", released in October 2010).

Many particulars -- such as precisely what Achievements to award, or how new boats and events would be unlocked -- arose organically or changed during the course of DLC development. To get the DLC we wanted, we realized we'd have to patch the base game with a Title Update (TU). Unfortunately each TU has to pass through cert, which means they're not free, and it behooves the small developer to keep the number of TUs to a minimum.

We issued one TU with a bundle of bug fixes and anticipated DLC features in August -- about a month after the base game's release and a month before DLC content complete. Even so, there were new features we thought about adding after the TU shipped. Unfortunately at that point we had to work with what we had.

It would have been far more effective to create the DLC concurrently with base game development. This would have allowed us to freely add or adapt features as needed, test both complete packages together and submit both for a single certification -- with the added benefit that we'd have the DLC in our pocket when the game shipped and could time its release strategically.

Of course the matter wasn't entirely up to us -- the final commitment for DLC didn't come down until about February 2010, right around the time we hit Content Complete for the base game. We know that next time this is something we need to push for on future projects.

Conclusion

If you're a game developer with a few years experience, you probably have a lengthy list of project do's and dont's in the back of your mind that could be titled, "What I'd Do Differently If I Made My Own Game."

For us, *Hydro Thunder Hurricane* was a chance to put that list into practice. And one of the most gratifying aspects of making this game was learning that, for the most part, our list was good. We wrote the tool set we'd always wished we'd had, pulled together a small, tight-knit team of talented contractors, and delivered a well-received sequel to one of the most beloved franchises in arcade racing history. We delivered it on schedule, and on budget. And to top it all off our game was selected for the Summer of Arcade.

Which is not to say the process was without its problems. There were gaping holes in our previous project experience that had to be filled, assumptions we'd made that turned out to be flatout wrong, and innumerable little surprises and pitfalls to sidestep and stumble over along the way.

Many of the items on our list had to be rewritten or scratched out in heavy, heavy Sharpie, never to be spoken of again.

But after all, you can't learn if you don't try, right?

The list is a work in progress.

Data Box:

Developer: Vector Unit
Publisher: Microsoft Game Studios
Release Date: July 28, 2010
Platforms: Xbox LIVE Arcade
Length of Development: 14 months (not counting initial prototype)
Internal team size, peak: 7
Man months, code: 26
Man months, content: 45
Budget: About the price of a house in Marin

Lines of Code: 170K
Development Tools: Autodesk Maya 2008, Adobe Photoshop, Crazy Bump, TortoiseSVN, Dev Studio 2008
3rd Party SDKs: FMOD, Bullet

Return to the full version of this article