

making games

DESIGN | BUSINESS | ART | TECHNOLOGY



CONQUERING NEXT GEN



OPEN WORLD WITH UNITY
HOW TO CREATE VAST OPEN
WORLDS ON MOBILE DEVICES

SERVER BACKENDS
THE PROS AND CONS OF DIFFERENT
SERVER ARCHITECTURE SYSTEMS

MOBILE BUSINESS
HOW FREE APPS ON AMAZON CAN
BOOST YOUR REVENUE AND PLAYERS

CELEBRATING

GDC

30TH EDITION

GAME DEVELOPERS CONFERENCE®

Moscone Center · San Francisco, CA

March 14–18, 2016 · Expo: March 16–18, 2016

GDCONF.COM

#GDC16

Save up to \$300 before March 9, 2016



Readers, use code **GDC16MG** to save an extra 10% on All Access & Main Conference Passes to GDC 2016!



NEXT-GEN-CONSOLES

A CONVENIENT THING FOR CUSTOMERS, BUT WHAT ABOUT THE DEVELOPERS?

Consoles have always had a decisive advantage over PCs as a game platform for the customers: the convenience factor. The devices were usually less expensive than a comparable computer on top of which you need a screen, a mouse, a keyboard as well as software while the small boxes just had to be connected to the already existing TV (let's exclude the so-called early adopters from the price issue!). Once bought, the consoles are set for the following five or more years – no upgrading or stuff like that which ambitious PC geeks empty their pockets for every few months. Admittedly, over the years this factor drove even me, a hardcore PC gamer, too, to the PlayStation 4 and Xbox One camps which by now include roughly 55 million players according to EA's CFO Blake Jorgensen. Although 36 million of them stick to Sony, Microsoft can still claim 19 million users – even though these figures are provided by EA and not Microsoft itself. This means that the PC still prevails according to the numbers (Steam has 125 million users), however, consoles are undoubtedly an incredibly important market.

Easier than ever

Here's the good news: Since the introduction of the current console generation the initial hurdles for developers have been lowered a lot, too.

While there used to be considerable differences between PlayStation and Xbox, the devices are comparable these days. While it was nearly impossible for smaller developers and publishers to publish their games in one of the ecosystems, the manufacturers now offer indie programs. While it took an unbelievably large effort to publish games on several platforms, there are engines today which practically do the work for you. What gives me the right to say so? Well, I recommend reading the reports based on the experience made by our experts: KING Art, for example, explain how the Unity engine allows work on different devices at the same time (p. 30); Realmforge show how to port a classic PC game to consoles (p. 24); or indie team Juicy Beast recall which mistakes can be made during a console release (p. 38). After reading these reports you will certainly agree with me that the publishing of console games has never been easier.

Enjoy the new issue!

Your Making Games Team

Sebastian Weber
is Managing Editor
of Making Games Magazin.



»The publishing of console games has never been easier.«





HOW NEXT GEN CONSOLES ARE CHANGING DEVELOPERS' OPPORTUNITIES

Who would have thought: The current cycle of consoles is more developer friendly than any of them was before. Jan Klose from Deck13 explains how the situation for independent developers has changed.



Jan Klose
is Creative Director
at Deck13 Interactive.

Jan was one of the men in charge of the development of »Lords of the Fallen«. In 2001 he founded Deck13 Interactive together with Dr. Florian Stadlbauer and worked on about 20 games that were released since then.

Development for consoles never was an easy terrain for independent developers. Especially in Germany which was, and in parts still is, a strong PC country by tradition. While there were always notable exceptions with developers like Keen Games and Shin'en, the few blockbusters produced in Germany that sold significant numbers were PC titles like »Anno«, »Settlers«, and »Gothic«.

With the advent of the »next gen« consoles it wasn't very clear if the big platform holders had any intentions at all to support independent developers on a strong level. Especially when Microsoft started advertising their new platform more or less as a TV set top box.

So to the majority of developers it came as a bit of a surprise when all of a sudden the preconditions changed, and only a few would probably have believed that two years after their debut, the Xbox One and the PlayStation 4 – and even the Wii U – are turning into attractive platforms for independent developers, even enabling them to go without a publisher as a middleman at all. How did that happen, and how can even startup game studios benefit from the trend before this amazing window of opportunity might close again?

The Deck13 Story

Back in 2009, we had developed our own game engine at Deck13. We were using OGRE3D for rendering. Our game »Venetica« was in fact inspired by the »Gothic« series. Not having a multiplatform engine ready, we had to rely on a third party to do a port to PlayStation 3 and Xbox 360. There weren't a lot of German experts around to offer such a port, so we ended up with an experienced French team. The decision to work together with a third party didn't leave us much freedom in terms of the game's look and feel on the consoles. This led to some unfortunate decisions that had a heavy impact on the quality of the final product.

We didn't want this to happen again, so we worked on our very own engine, called »Fledge«, and this time we had a multiplatform approach in mind right from the start.

We needed the support of our publisher to get access to Dev Kits for Xbox and PlayStation. Being new to the console space, and with Dev Kits still costing a fortune, there wasn't any other way.

The games we were creating, called »Blood Knights« and »Tiger & Chicken«, were supposed to be released in the digital stores of PlayStation and Xbox. But even though this meant that we wouldn't need a publisher for

retail, in the end we still needed one because (besides funding) of a strange rule by platform holders that required the studio to have shipped physical titles before they were allowed to publish digitally. This rule led to the fact that basically only traditional publishers were offered »slots« in the stores because ... well, before doing something digital, you had to have a retail product in your track record. And yes, in your own track record, not in the one of your publisher. This effectively ruled out the possibility of digital self-publishing. (Unless you found a publisher who offered »selling« you their slot in the digital marketplace – yes, this did happen.)

After about a year, Fledge was »console ready«. We worked on our next major title, the Action-RPG »Lords of the Fallen«. And all of a sudden, the publisher asked us if we could please »make it next gen« and switch to the new (still unreleased) consoles. Well ... yeah, sure. That felt like the only good answer, and so we started working towards a »next gen« game release, ditching PS3 and Xbox 360, even though we didn't even know the specs of the new hardware. We were just relying on rumors from around the Internet. (The Internet might be a terrible place to find out »the truth« in general, but in terms of the actual console specs it proved quite accurate!)

One of the most interesting rumors was that porting to the next generation was supposed to be incredibly straight-forward. Wait ... porting to a console should suddenly be a simple thing? Well, when the dev kits finally arrived, we found out that there was at least some truth to that. Compared to working with PlayStation 3 or Xbox 360, it was a lot more straightforward, indeed. It felt even more similar to developing for the PC, and the tools provided by the platform holders were much improved compared to »last-gen«.

A year after the release of Lords of the Fallen and two years after the consoles' debut, we're now working on our next big title, the Action-RPG »The Surge«. Was it worth taking the risk and switching to the new tech? Absolutely! With the experience we gathered, we are now able to utilize the consoles in other interesting ways too, and that's mainly because of an entirely new philosophy of the platform holders.

The Platform Holders' new Approach

After the apparent failure of the Xbox Live Indie Games (XBLIG) store concept, Microsoft stated that it will shut down the XBLIG and XNA Creators Club programs, with the XBLIG marketplace shutting down in September 2017. However, instead of ending their partnerships with these small development studios, Microsoft now focuses on their new ID@Xbox program for the Xbox One.

The ID@Xbox program currently has over 1,000 developers creating Indie games for the Xbox One, with many developers receiving

free Dev Kits, getting access to Xbox One and Xbox Live components such as Kinect and SmartGlass, and even a subscription to the Unity engine.

Wait, what was that? No more barriers for Indies on the Xbox? No more barriers to enter the marketplace? Self-publishing accepted? And even a free engine on top? Now how did that happen? And what are the other platform holders doing?

Well, Sony is doing something quite similar, even though it might be a bit more expensive for Indie developers to get Dev Kits. But again, no more barriers like in the old days. Feel free to release your game.

This is an amazing trend, and Nintendo doesn't want to be far behind, offering a comparable concept for their Wii U. So you're ready to go, just get in touch with the account managers and they will help you to get all set. Of course, it might take a while due to the huge number of Indie developers currently subscribing for these services.



Venetica was ported to Xbox 360 by an external partner as Deck13 was not capable of doing it by themselves, leaving the studio with almost no influence of the outcome.



Tiger & Chicken was created with Deck13's proprietary engine Fledge, but back then the engine was not yet fully optimized for consoles.

You can do it. (And everyone else, too.)

So here we go, two years after the new consoles were launched the platform holders are embracing independent developers. It's a wonderful move that makes a much more thriving games economy possible. However, it also causes a much stronger competition. When the barriers to enter the market are that low, a lot of developers will enter it and fight over the gamer's money. Take a look at what's happening with Steam: According to SteamSpy, 2015 saw over 3,000 new games on the platform, compared to about 1,900 in 2014. We can assume that the Xbox and PlayStation stores will see a strongly increasing number of titles in 2016 as well.

Goodbye Mr. Publisher?

So can we developers now sell directly, without a publisher, and finally become rich and famous without being forced to share the spoils? Well, it isn't that easy. Consider why you needed a publisher in the past: Retail, marketing, PR, project management, Quality Assurance tests (especially to meet the TCRs or XRs, the tough set of rules that platform holders require of a game to be published at all), age rating and localization tasks.

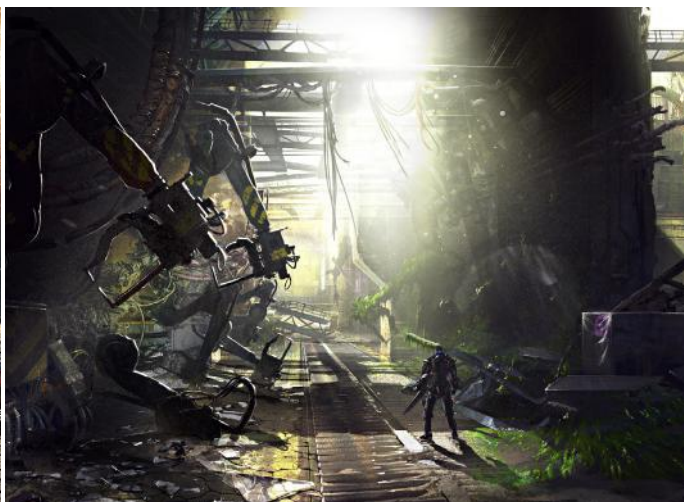
But first of all, most developers turned towards a publisher for (relative) financial security. The publisher offers to fund the development in advance, a thing that almost no bank in the world would do. And in most of the publishing contracts, the advances are defined non-refundable even when the game doesn't sell at all. In that case, the publisher loses the investment but the developer is allowed to keep what he already got. The price for this is being paid off with 15 to 25 percent of the publisher's income (after deducting most of his expenses!). This won't change much in the new world. For a big project, you will need the financial power of a publisher and you'll need to give away large chunks of the game's IP and royalties.

For smaller projects, however, it is now entirely different. You don't need a lot of funding. Great engines are available almost for free. You don't need to feed a big team. Great indie games can be created by a handful of people. Smart teams are able to manage themselves.

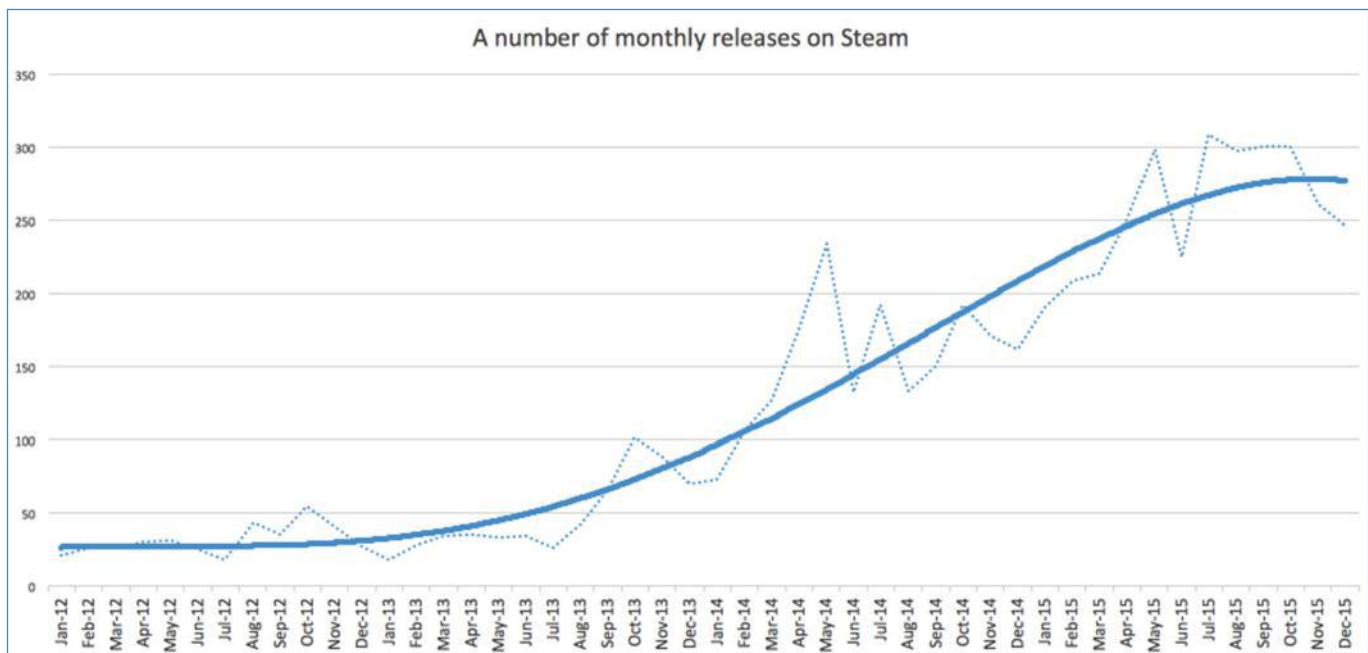
The downside is that, as said above, almost every small studio is able to release their games by themselves, creating an awful lot of competition. Having created a great game doesn't guarantee that you will sell more than a dozen of copies. There are just so many great little games out there that when you're a game developer, you aren't someone special anymore. What you need now is the attention of the players. You need to get the word out. You need to build a community. You need to scream and shout to be heard in the tempest that is social media. When advertising your game, you might feel like you are a salesman knocking on the door of every single player. Those of us who have done a Kickstarter project will know the feeling: Just having »a cool Kickstarter project going« may give you, well, zero backers. You might need to talk, beg, call up friends and family, collect all the favors that people might owe you, just to have your campaign succeeding, even if you had set just a modest funding goal. And that was just the start. If you finance, create, produce and market a game all on your own, this can sometimes feel like a very hard and lonely business – unless you were lucky and hit gold, with just the right product, at just the right time, in just the right place.

You don't need to be alone

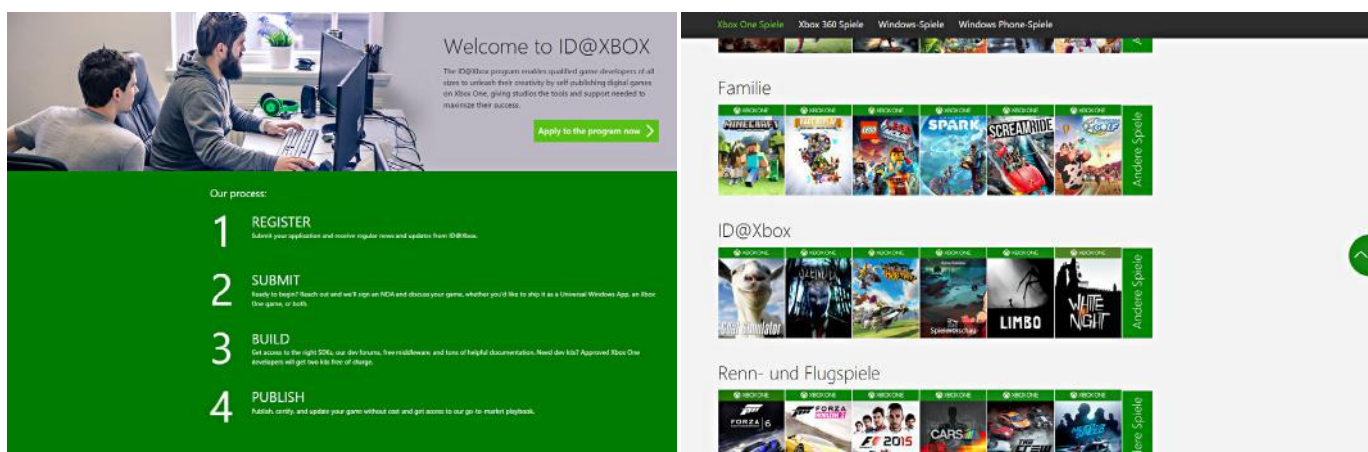
But all in all, for many studios the chances of being truly independent will outweigh the risks. And there are new ways of finding partners for your project. Deck13, as well as other larger developers, offers support in the areas of marketing, PR, production, release, and other fields. You can just call us and pick what services you require. In return, we take a share of the income, but it's not the major



Lords of the Fallen (left) was the first game that Deck13 developed for next gen consoles, using their Fledge engine. Now the team is working on The Surge (right).



This chart by SteamSpy shows how the amount of released games has increased on Steam as now Indie developers can do that on their own - the same situation will come true for consoles sooner or later.



Joining programs like ID@Xbox makes it very easy for independent developers to publish their games on consoles and Microsoft is even advertising those games on its store page - the same is true for Sony and Nintendo.

share, as it was with traditional publishing contracts, and you're free to decide which services you need. Deck13 is just one of many seasoned developers stepping into this field of partnership. As each of these studios can only take on a certain amount of partner developers at a time, there's plenty of space for more players on the market. In any case it's worth a try. Why do it on your own if you can do it together with a partner? Especially, if the partner has experience with developing and releasing for the »next generation« (or now rather: current generation) platforms.

The Road Ahead

The new consoles are established, they have gained decent market shares, their online stores are thriving. Indie developer programs are in place. That's not a bad time to start a project on those platforms. And while big titles with big ambitions and big budgets will

most likely still need big publishers to back them (and things will just stay as they were for many years), smaller independent projects, can now secure more fitting partnerships that allow your studio and your project to stay independent. Where getting a contract and getting funded was the toughest task in the past, today it is getting the players' attention so that they recognize your game out there. But today you are free to choose a partner who helps you just in the area where you need help, so you don't need to end up in a do-or-die situation where you might be forced to sign a crappy contract just because it's your only chance to become a part of the industry.

Jan Klose



PORTING VAN HELSING TO XBOX ONE

The team of Neocore Games calls themselves a traditional PC developer and still the team took the rocky road of porting their Hack'n'Slay Van Helsing to Xbox One. It was a rough journey but in the end it was worth it.



Balázs Farkas
is Communications Manager
at NeocoreGames.

Balázs Farkas, Communications Manager, is working at Neocore-Games since 2015, where his main responsibilities are written communication (mainly the devblog) and research. He holds a Bachelor's degree in Library and Information Science, as well as a Master's degree in Knowledge Management.

The end of 2015 marked a new milestone for NeocoreGames – we finally released the first console port of »The Incredible Adventures of Van Helsing«. Our game has appeared in the Games With Gold program on Xbox One, and essentially doubled our audience. This is nothing unheard of, but we're a small (but growing), independent Hungarian studio, with a portfolio of PC exclusives, so this was a great learning experience.

We have worked with publishers before, but with the Van Helsing franchise, we were on our own since its first release on Steam. Publishers often help with porting to other platforms, but independent developers who can't afford third parties need to make the necessary steps themselves.

Porting games wasn't something that required our immediate attention, nor was it something that we were experienced in, but judging by the success of our game on PC (and the demand for similar games on consoles), we decided that we'll get into it. It was much

more difficult and complex than we anticipated, we weren't quite prepared for all the requirements of the console market. So here's how these things go.

With other Platforms in Mind

First things first, we have our own game engine. This is a rare thing as developers nowadays have the luxury of using very capable



multiplatform engines, even using them for free (up to a point), but if a developer studio decides to use their own game engine, they need to prepare it for porting.

The Incredible Adventures of Van Helsing is an Action-RPG that was developed with a PC gaming mindset. While the divide between PC gaming and console gaming is starting to vanish, it still exists, especially in Hungary where console gaming gained popularity at a much slower pace due to various reasons. Most of the senior developers in this country are more comfortable with PC games.

This was true for NeocoreGames as well. Previously we made games in genres that are traditionally for the PC (real-time strategies with RPG elements), and we were used to a kind of mental picture of a gamer sitting in front of a monitor with the convenience of clicking on things with a mouse.

You can see where this is going, but fortunately, early in the development of Van Helsing we were getting to know the console gaming mindset as well. PlayStation 3 and Xbox 360 were quite capable machines, and mobile gaming was on the rise (that was around 2011 and 2012).

So we kept this in mind, and we started experimenting with porting to OSX (this was successful), Linux, iOS and Xbox 360, but we soon encountered difficulties.

For iOS, we managed to complete a port, technically speaking, but we soon realized that the game interface wouldn't work and all the skill trees and gameplay mechanics needed to be redesigned according to resolution, screen size and ratio for multiple models of iPads. As for the Linux version, we just didn't have the capacity to test multiple distributions extensively, so we never released it.

We were always considering a controller support, though. Playing an isometric Action-RPG with a controller was a neat idea, but there were artifacts of the PC gaming mindset – some skills needed a cursor to point, what needed to be addressed (later on console ports as well), and little things like this made the development always slightly more complicated.

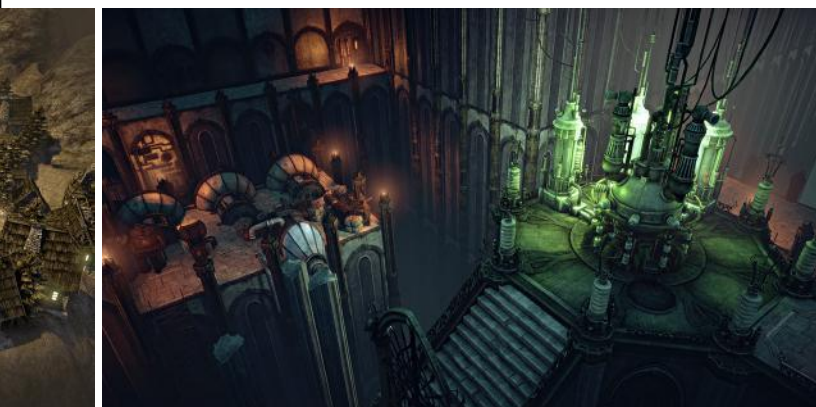


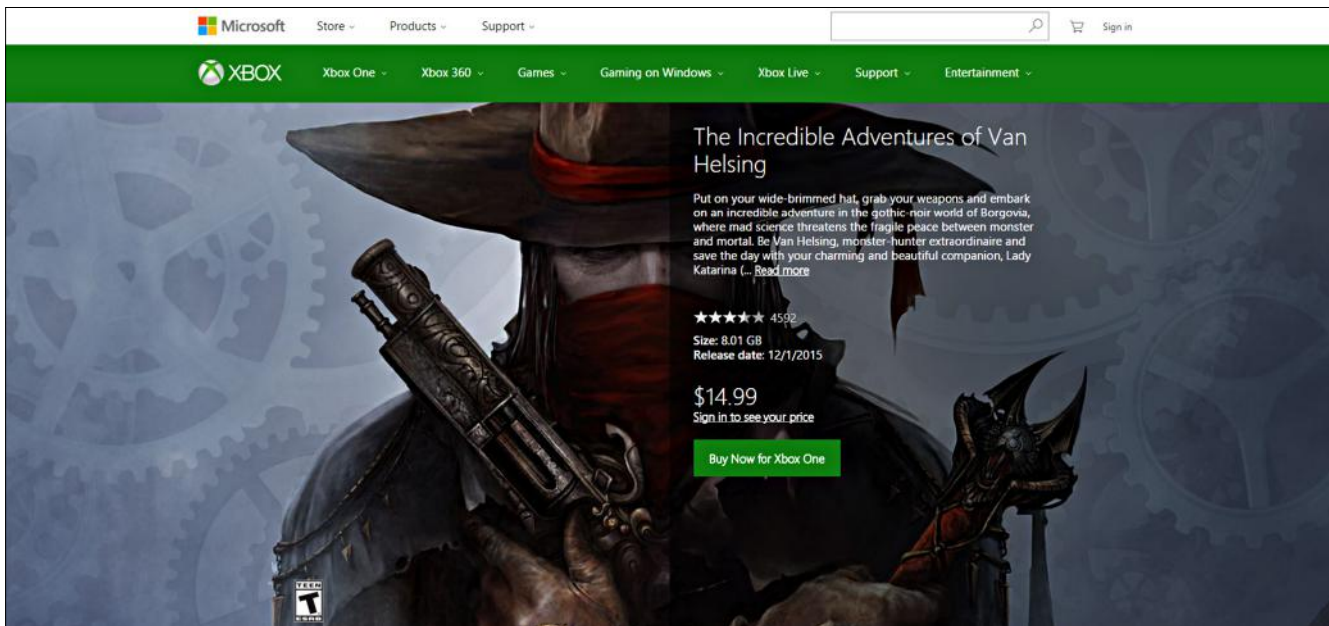
The Incredible Adventures of Van Helsing is a Hack'n'Slay comparable to Diablo. The team released three games within the franchise so far.

Preparing for the Next Generation of Consoles

Good news or bad news, you decide: We started porting our game to consoles on the turning point of generations. PlayStation 4 and Xbox One were coming, and they were coming fast. We already announced (or at least hinted at) ports for the previous generation of consoles, so the communication to our fans got awkward for a while, and we were unsure about how this would play out.

Originally the team planned to release the game for PlayStation 3 and Xbox 360. But NeocoreGames quickly realized that the performance of the next gen consoles would fit the game better.





The Incredible Adventures of Van Helsing is available for \$14,99 in the Xbox Store, the same price that PC users have to pay on Steam. In December 2015 the game was available for free for Xbox Live Gold users.

But the porting itself provided the answer: Van Helsing needed much work to make it run on PlayStation 3 and Xbox 360. For optimal performance, we needed to redesign and scale down much of our graphical content, but we had no capacity to do so. Eventually, we gave up on the idea to release the game on PlayStation 3 and Xbox 360. We looked forward to the next generation of consoles, as they were much more powerful and flexible machines.

This helped much indeed. But these consoles have very strict guidelines and quality control to ensure the comfort of players, and once we started working on the console ports, meeting the requirements of the certificates was taking up 70 percent of the time spent on developing them.

These aren't unreasonable requests, mind you, but for a PC developer, they are sometimes unexpected. These include cloud saving, suspended game (you need to ensure your game doesn't crash or quit when the console goes to rest or sleep mode), different requirements for PlayStation Network and Xbox Live features, the framework for achievements and trophies, the different aspects of multiplayer, interface, controllers, and so on.

Again, these are all very, very reasonable requests. But we were often surprised as we did not consider them previously. This is not because of lack of research or laziness, as we were aware of these features (the guidelines of the console manufacturers are very clear), but we were surprised how these features can mess up our work that we considered done (our carefully selected font looked nothing like on PC for example), and our engine wasn't quite on the same wavelength, so to speak. This meant extra work as well as extra time.

The Gravity of the Task

But how much extra work exactly? By the time we arrived at Xbox One's certificate process, the whole NeocoreGames team consisted approximately of 40 to 50 people. This number seems big, but it includes marketing, localization, side projects, support, communication, etc. Considering all this, we are a small team for such a big project (and we can safely say that Van Helsing got really big in just two years).

We have two coders working on the console ports, full time. We have one or two internal testers, depending on what stage we are at in the development, but we work with a third party company that specialises in testing the games, so this helps. We also have an interface designer, and one or two people communicating with Microsoft and Sony (there are technical and business related questions).

And how much extra time? Well, this is a bit unorthodox, since we made small mistakes, oversights in the process, and our engine needed much attention, and ... it took us several months to get a former PC exclusive released on Xbox One. With the sequels and our future projects, it won't take this much time, it will go much faster. We have learned practically everything during this work.

But, yes. Our first big console release took months. This is partly due to the fact that the development of The Incredible Adventures of Van Helsing III and eventually Final Cut (a free upgrade that includes all three campaigns in a much bigger game) overlapped with the development of the port, but still. PC developers who are considering console ports with a custom engine, need to prepare for the gravity of the task. It gets easier, it gets faster, but as a developer, you need to be very careful

NeocoreGames

NeocoreGames is an independent video game development company that was founded in 2005 and that is located in Budapest, Hungary. Currently the team is working on »Warhammer 40,000: Inquisitor – Martyr« that was announced for PC, Mac, Xbox One and PlayStation 4. In addition, the team is said to be still working on »Broken Sea«, a roleplaying game that was announced in 2012. Before that Neocore Games worked on the King Arthur series and the series of games that are dealing with the theme of Van Helsing.



to see if you have the capacity, knowledge and time needed. Fortunately, we did. And we can safely say, that it was worth it.

After Release

We were lucky enough to have gotten accepted for the Games With Gold program. This is something that helps gain audience for first time console developers. Since we have sequels and a spinoff on Steam already, this will help us find an audience for them on Xbox as well.

Other than that, communication and support, for now, seems the same as on PC. One exception is the question of patches and updates, as they need to go through the same certification process and tests as the main game. There's no workaround, there are no beta patches, so if you're going to release your game and your patch, it better be top notch, because it takes a bit more time to work on them.

All things considered, it's an exciting process in an exciting time for console gaming, and we look forward to release our other games as well, with the hope that our future projects (including »Warhammer 40,000: Inquisitor – Martyr«) can be released on three platforms simultaneously. That would be great.

Conclusion

If traditional PC (Steam) developers are thinking about porting their game to other platforms, they need to be familiar with all the tasks that are needed to make a successful and working port. It boils down to the following principles:

- Make sure you create or license a game engine that meets all the requirements of different platforms (popular open source engines are developed and updated with this in mind).
- Consider the hardware capabilities and possibilities of the targeted platform, including suspended game mode, sleep mode, removal of different cables, etc.
- Consider the features of different controllers, motion sensors, audio, VR, etc.
- Consider that multiplayer works differently on various platforms, and that online features can be used in single player games as well (cloud saving, leaderboards, etc.).
- Design your game interface with the resolution, font set, screen area of the target platform in mind.
- Make sure that the team can afford the extra time and work that is needed (remember, this could mean months!). Make sure that your work will be worth it, ask yourself the question: is there an audience on the targeted platform that will be interested in your game?
- Consider the different mindsets of PC gamers and console gamers.



A comparison with the PC version of the game (lower image) shows how the team adapted the user interface to console controls.

- Read the quality and certification guidelines of the target platform thoroughly, and make sure you meet every requirement before submitting.
- Ensure that there are people in your team assigned for support and communication, especially around the time of release and patches.

And that's our wisdom on the subject as of now. Hopefully it helps others. **Balázs Farkas**





EXPERIENCE GAINED FROM PORTING PC GAMES TO PLAYSTATION 4

Gaming Minds published their strategy game *Grand Ages: Medieval* 2015 not only for PC, but also for PS4. Daniel Dumont looks back and explains which problems occurred during the port and what progress Sony's console has made.



Daniel Dumont
Co-Founder and
Creative Director
at Gaming Minds Studios

Daniel has been developing games for more than 16 years. He developed later titles for the »Patrician« series at Ascaron and designed »Port Royale« and »Darkstar One«. In 2009 he co-founded Gaming Minds Studios together with Kay Struve and Kalypso Media, which have since worked on games like »Patrician 4«, »Port Royale 3«, »Rise of Venice« and recently »Grand Ages: Medieval«. His focus lies on the vision and the concept of computer games.

Our team has been developing games on various platforms for nearly 20 years; among them are three titles for Xbox 360 and two for PlayStation 3. »Grand Ages: Medieval« is the first title we developed for PlayStation 4.

Kudos to Sony: Basically, the development environment of Sony has made substantial progress since PlayStation 3. The tool chain is very solid, the documentation is comprehensive and sufficient in almost all areas. Sony's development support usually replies within a few hours even to low-priority questions.

Graphical adjustments

Due to the modern PlayStation 4 hardware no special textures, 3D models or shaders were necessary. Even the assets for the highly detailed PC graphics worked well on PS4. All we had to do was convert the textures and

shaders into PS4-specific formats before we could use them.

Our engine uses an abstraction of the rendering API, which is somewhat similar to DirectX 11, and could quite easily be extended by a renderer for PlayStation 4. In order to reduce the time effort the port of *Grand Ages: Medieval* would take, we decided not to use some of the more advanced features like compute shaders (see the chart regarding the development effort in [image 1](#)).

PS3 legacy

The user interface in the game and many of the PSN-specific features are still like they were on PS3; that's why a lot of experiences made from working with the last generation are still valid (like trophies, leaderboards, PSN user administration, PSN store integration).

Since we developed the game with the PC as lead platform, the gamepad support was quite a lot of work, especially in terms of designing and programming the interface (e.g. HUD, dialog, 3D map etc.), even more so since our goal was to build a typical console interface. Normally, we develop the mouse and the gamepad interface at the same time. In case of *Grand Ages: Medieval* it wasn't clear at the beginning whether we would need gamepad support, which is why we left it aside for the time being and then had to implement it afterwards. Altogether, this causes more work and isn't recommendable in general: If both designs are being done at the same time, it's easier to find compromises for the controls.

Generally, we stick to two rules regarding gamepad implementation, which are illustrated in the sample image ([image 2](#)):

1. There can only be one clear focus on dialogs. Unlike with the PC version, you can't expect

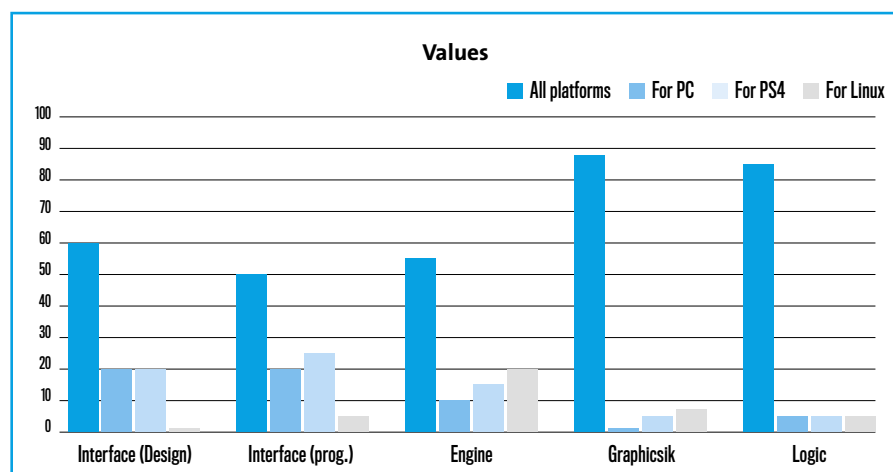


Image 1: The graph shows how the effort for various development steps is distributed among the supported platforms.



Image 2: The two images show the PC and the console interface in comparison.

users of the console version to click on various elements of different sizes in a dialog.

2. Ideally there should be no buttons in the dialog; especially the many small buttons which are sometimes common in PC dialogs (e.g. for »next« and »back«) must be solved via gamepad. Loosely speaking, you replace a button in a PC dialog with a gamepad button which is then displayed on the HUD.

For comparison, take a PC dialog with a lot of buttons and the console equivalent without buttons (see image 2):

The buttons in the PC dialog are numbered 1 to 7 in image 2. They are of different size and style. Some consist of text only, others of symbols or graphics. Also, they are spread out throughout the whole dialog. All that's left in the console version are the large buttons 5 and 6. They are controlled by the left stick and selected by the X button. The buttons 1, 2, 3, 4 and 7, however, were moved to the HUD since they are controlled directly by a gamepad button.

Hardware

The similarity of the PlayStation 4 hardware to PCs and the integration of the compilers in Microsoft Visual Studio, too, have simplified the porting process significantly. The optimization of the CPU usage was more in focus for the PS4 version than for PC, from which the other platforms benefited as well though.

Our user interface library, for example, uses quite a lot of ActionScript code. These program parts were often responsible for frame rate drops. Through thorough analyses and experiments we found out which GUI elements were responsible for that and then revised or replaced them.

Sony provides excellent profiling tools which help check the performance in every detail. The CPU analysis tool, for example, can record the program sequence through many frames and then analyze which functions on which thread are responsible for a frame rate drop.

That said, compared to the porting of »Sacred 2« or »Port Royale 3« for Xbox 360 and PlayStation 3, the optimization required a lot less time.

Submission

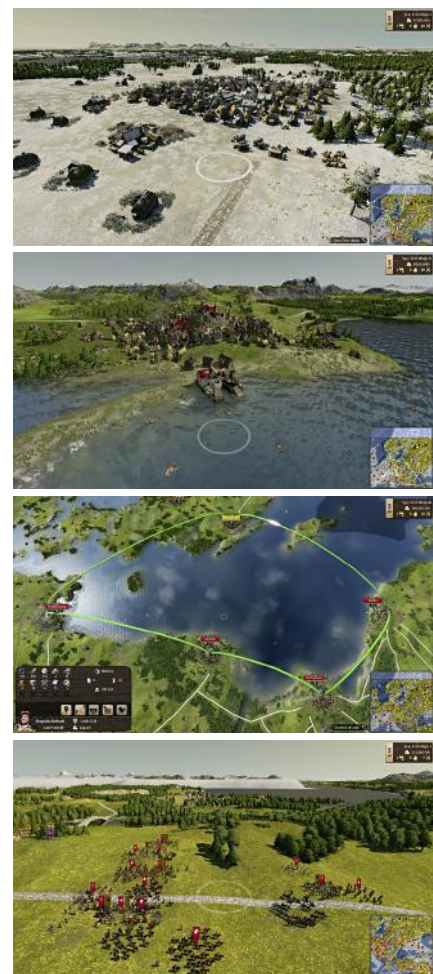
The gold master creation for PS4 has become a bit easier and the submission test runs have been significantly shortened – which is particularly helpful at the end of the project and after release. And yet, submission was a nerve-wrecking milestone: The checklists have been thoroughly revised, therefore the experiences from previous PS3 submissions aren't always helpful.

We had a critical fail during submission, for example, which occurred due to a misunderstanding of the definition of »user-generated content« (UGC). In our game, the players can rename cities that are controlled by them. This feature is in the UGC category, so it's important to make sure that young players for whom UGC is blocked according to the parental control settings can't see the city names chosen by other players in multiplayer mode.

We also had an error in the IP address inquiry, which didn't occur with us or the test studios since we didn't do any tests in multiplayer mode where users are registered behind different routers. Unfortunately, due to this error only very few people were able to join multiplayer games at the time of release, and we had to provide an update at short notice.

Updates on PS4 are very easy: The patch system is straightforward and has many features, the submission tests are very fast, and making a patch available is done with just a few clicks.

Daniel Dumont



Since PlayStation 4 has a similar technical structure as the PC, only a few graphic amendments needed to be made for Grand Ages: Medieval.



TECHNICAL CHALLENGES OF A PLAYSTATION-PORT

It was only after Dungeons 2 had proved to be successful that the decision was made to adapt the strategy game to Sony's console. The technical team of Realmforge Studios explains why Unity is a good choice for that and which technical issues PC developers may have to face.



Christian Schlütter
Games Producer
at Kalypso Media

Christian Schlütter started his career in the games industry as a volunteer and editor at »PC Games« magazine. Since 2010 he is a Games Producer at publisher Kalypso Media which is based in Worms, Germany.

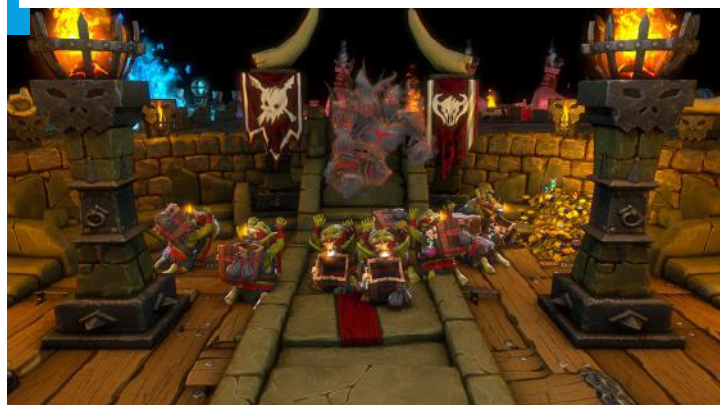
After Dungeons 2 was released on April 24, 2015 for PC, Mac and Linux, the Munich-based development studio Realmforge quickly realized that they would have some extra work to do – in the best possible way.

Players and critics alike loved the mix of dungeon management and real-time strategy, and publisher Kalypso Media couldn't have been happier about the sales figures either. Together they decided to target a port for PlayStation 4 parallel to the planned DLC for the desktop version. But with regard to the console environment, Dungeons 2 was a bit special: First of all, it was a strategy game; plus, it was also a project which hadn't originally been developed with a console version in mind. Both aspects were to pose challenges to Realmforge in the months to come.

Fundamental changes to the controls

»Never think you can map the controls one-to-one!« says Immanuel Scholz of Realmforge – who is in charge of programming the controls among other things – when asked about tips for porting a PC game to PlayStation 4. Apart from the obvious adjustments (like no mouse input, only limited number of buttons etc.), there are additional issues a developer has to face during a PS4 port – especially with strategy games. To take the keyboard controls of a PC version and map them one-to-one to the buttons of a PlayStation 4 controller simply won't work. There are too many context-sensitive areas, too many special cases, and too many things that need to be done with sticks and D-pad. Basic »manual programming« of the individual commands is therefore more effective. What should also be considered beforehand is that apart from the actual

Just like Dungeons (2011), Dungeons 2 pays homage to the classic Dungeon Keeper, but was extended by a strategy part in the world above, which is reminiscent of Warcraft.





controls the respective GUI and texts need to be adjusted, too.

For example, PC strategy players are used to getting mouse-over texts and explanations displayed to a lot of buttons and graphics, and it's these texts that are tricky to transfer to the console version. Realmforge still shows the required information context-sensitively on the PlayStation 4, but in a frame on the bottom of the screen (see [image 1](#)). The only way for this area to be available though is because we are arranging the action buttons that were originally located there in a well-presented circular menu. This menu shows all possible actions, depending on the currently selected unit or room.

And then there is the adaption of all sorts of texts. Tutorial texts such as »Right-click on this object« have to be exchanged and adapted to the new platform. Also, being a game



Image 1: For the PS4 version, Realmforge decided to arrange action buttons which are otherwise operated by mouse or hotkey in a circular menu.

that heavily relies on the mouse, Dungeons 2 faces a challenge on PlayStation 4, which most jump 'n' runs and shooters of the console world are exempt from: There is no hardware cursor. »On the PC, the Unity engine simply takes the information of the hardware cursor and forwards it to the game. This doesn't work on the PlayStation 4, so we had to write a routine ourselves« says Immanuel explaining how hard it is to give the game information at all about where the player is pointing at or clicking. The result is the »overridden-MousePosition«:

```
public static LocationService location { get { return Input.location; } }
public static bool mousePresent { get { return Input.mousePresent; } }
public static int touchCount { get { return Input.touchCount; } }
public static Touch[] touches { get { return Input.touches; } }

public static Vector2 overriddenMousePosition = new Vector2(float.NaN, 0);
public static Vector2 mousePosition
{ get { return !float.IsNaN(overriddenMousePosition.x)
    ? overriddenMousePosition : (Vector2)Input.mousePosition; } }
public static void ResetOverriddenMousePosition()
{ overriddenMousePosition = new Vector2(float.NaN, 0); }

public static bool compensateSensors
{
    get { return Input.compensateSensors; }
    set { Input.compensateSensors = value; }
```



Technical restrictions

The coders of Realmforge had some trouble with the required »ahead-of-time compilation« on the PlayStation 4. The reason for this is rooted in the C# code Dungeons 2 was created with using the Unity engine. »Generic data classes, e.g. a list, are only compiled at runtime by the JIT Compiler on the PC platforms« explains Korbinian Abenthum, Technical Lead at Realmforge. The problem: For security reasons, the architecture doesn't allow for an application to create and execute program code during runtime on the PlayStation 4. Instead, all list versions must have been noted and compiled when the program starts (»ahead-of-time«). However, the PC version of Dungeons 2 relied heavily on the creation and usage of such data during runtime.

To find and adjust every single case proved to be an industrious piece of work as programmer Jörg Böhnelt knows: »The problem is that things like that aren't marked as an error during the compilation in the Unity editor. You only notice it when the game continues to crash at a certain moment while you play. And sometimes this can happen in very obscure cases, for example, when a single unit uses a certain spe-

cial skill in a very special way. If there's a crash then, the order is: »Analyze crashlog, patch and compile again.«

Complex optimization

As expected, the optimization was one of the biggest tasks when porting Dungeons 2 to the PlayStation 4. Even though the next-gen consoles are much more powerful than their predecessors, their special hardware architecture requires adjustments compared with the PC development. The CPU of the PlayStation 4 consists of eight cores divided into two clusters. Cores 1 to 4 make the first cluster, cores 5 to 8 make the second one. The challenge is to amend the game in a way so the individual calculations (like rendering, AI, GUI etc.) are optimally distributed onto the various cores, no operation has to wait for too long for another one, and the priorities of the respective threads are always selected correctly. The latter one is easier said than done since for stability reasons the Unity engine predefines the priority for a lot of operations.

»You could say we had to work around Unity in order to prioritize the threads in a way that would work best for Dungeons 2« says Jörg. The cluster architecture raises another problem: »If a core is busy with an operation, the PlayStation 4 tends to call on the corresponding core from the other cluster, which causes only a slight delay. However, if that happens several hundred times per second, it results in considerable performance issues. We therefore tried to distribute the operations to the cores of the first cluster as much as possible, so that the system wouldn't even start calling on the second cluster (see [image 2](#))« explains Julian Moschüring who's mainly in charge of the optimization of the PS4 version at Realmforge. He also has the thankless task to revise the update routines of Dungeons 2 for the console versions.

The basic settings of Unity specify that all objects in the game (»game objects«) are updated continuously. Even if they aren't used, game objects won't be deselected, which would save computing time. This isn't a problem for a lot of games, especially when mostly static objects are being used. However, in addition to the real-time strategy world above, Dungeons 2 also offers a completely expandable dungeon. Every single tile in this dungeon can be mined, built on or changed in any way. Each character can do various tasks and be in any given location. A lot of things like gold, mana and barrels can be carried by characters. In brief: Each dungeon consists of over 30,000 game objects which would have to be included in the calculations, even if they aren't being used or aren't even visible.

Julian and his colleagues therefore had to optimize Unity's update routines in a way that everything that wasn't needed at the moment would be deselected or only updated rudi-



The screenshots show the PC version (lower image) and how the user interface has changed in comparison on the PS4 (upper image).

mentary. This applies to »nav agents« for idle creatures just as it applies to particle systems and light sources which aren't currently visible. »In the last few months we were thus able to reduce the approximately 3,000 update calls per frame to 50« says Julian. The resulting increase in performance also benefits the PC version through patches.

The right engine and important preparations

Even if Unity isn't the one-click wonder in practice that many of us had hoped it would be, the engine still proved to be the right choice. The guys at Realmforge especially appreciated the direct communication with the developers of the engine. »The PS4 support of Unity through forums and tickets is just awesome«, says Jörg. And he also has a very basic tip for all future PlayStation programmers: »Get yourself a static IP at an early stage!« Because without one Sony won't even register you as a PS4 developer. And depending on the internet provider, this may take several months.

Christian Schlütter



The technology team at Realmforge

Senior Programmer Julian Moschüring, Jörg Böhnel and Immanuel Scholz (from left to right) as well as Technical Director Korbinian Abenthum (not pictured) are the technical backbone of Munich-based developer Realmforge. In the last few months, they ported Dungeons 2 from PC to PlayStation 4.

Korbinian is one of the founders of Realmforge Studios who once tinkered with a modified Ogre version as the sole programmer when times were hard. Luckily, these days are over, and nowadays he focuses on the big picture. After a long odyssey through various engines, Unity is now the light at the end of the tunnel.

Julian joined Realmforge in 2009 and has since been in charge of everything related to graphics and performance optimization.

The graphic designers at Realmforge honor him, so he keeps providing them with new features. Shaders, visual effects and multi-threading are precisely his thing.

Immanuel has been on board since 2010 and was already part of the development team of Dungeons 1. He loves to disappear into the depths of gameplay programming, but along the way he also takes care of the AI, the pathfinding or he implements new spells and skills into the game – if the game designers supply him with plenty of gummy bears and other goodies.

Jörg also joined in 2010. When he isn't busy working on the client-server architecture, studying the assembler code or grooming his impressive beard, he's always on the spot whenever there's an emergency.

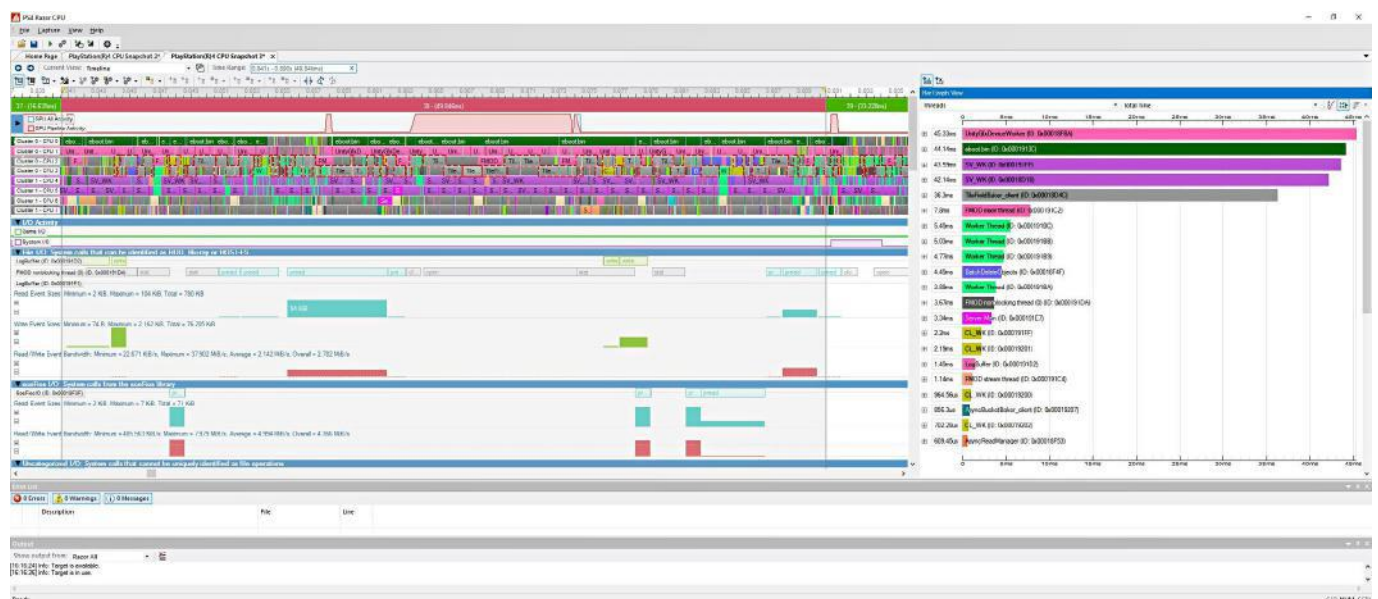


Image 2: Realmforge tried to distribute operations as much as possible onto the cores of the first cluster, so that the system wouldn't even attempt to call on the second cluster.



AN INDIE MAKING IT TO THE BIG CONSOLES

Black Forest Games describes how they published Giana Sisters for PS4 and Xbox One and which obstacles they had to overcome on the way.



Mischa Strecker
Product Manager
at Black Forest Games

Mischa is looking back on a long career in games journalism and product management with tenures at CyPress, Axel Springer, Yahoo and Web.de. Since 2012 he's been in charge of PR, marketing and community for Black Forest Games based in Offenburg, Germany.



New artworks or logos usually aren't needed for a console version...



... unless you release your game in China.

A brief look back: In 2012 we started our first crowdfunding on Kickstarter. Our goal was to bring Giana back to life – first for PC, but ultimately we were also planning a console release. It wasn't exactly a piece of cake; we had some experience with console games, however, we had never released one. Since early 2015 our baby is now available on Xbox One and PS4 – and we're working hard to repeat this success with Rogue Stormers.

A look at Steam shows that the initial hurdles are low. Even small teams without much experience can publish their games relatively easily. That was still different a few years ago – Steam put large obstacles in the way of developers and demanded working with a publisher. In the meantime the regulations have loosened, there is Green Light and Early Access, and the catalogue on Steam now includes the tremendous and overwhelming number of 30,000 games and apps.

Certification as a developer

It's a different picture on consoles, but access here has been made a lot easier as well. In order to get approved as a developer at all, you have to submit an application. For all platforms you have to start by filling in a form asking for basic information: Are you a registered company with your own secured office facilities, a secure network, and do you already have a portfolio of published games? Established studios with a track record have an advantage here.

Once these obstacles are removed, you still haven't reached the finishing line, but at least you may call yourself »Certified Developer« already. By now all manufacturers offer inexpensive programs to equip indies with the necessary dev kits. Microsoft provides developers with two free consoles via the ID@Xbox program, including a Unity license. Sony and Nintendo still sell the kits at a price of around

1,000 euros (used to be approx. 3,000 euros), but they also offer rental programs for indies.

Game approval

Just because the company has taken the first hurdle doesn't mean that their game will do the same. Consoles are still closed systems – the manufacturers put quality before quantity, and in order to guarantee this, they set clear rules.

Technical Requirements Certification

One of the biggest obstacles that needs to be overcome to release a game on a console are the Technical Requirements Certifications (TRC for Sony / XR for Microsoft). Every platform owner provides these in one way or the other and to different degrees. These documents include technical – but also gameplay-relevant – specifications that a game **MUST** meet. The content ranges from handling saved games and specifications for button assignments to player invitations to direct hardware access (spoiler: it's not allowed!). These documents need to be studied. Over and over again. Thoroughly. Ambiguities – which arise over some issues – can usually be eliminated through direct contact with the platform owners, but the dialog doesn't replace the familiarization with the matter.

The biggest bottleneck during a port or development of a game is still the compliance with the TRC – independent of the platform.

From PC to console – a game is being ported

The submission procedure of the individual manufacturers is a lot easier compared to what it used to be like, which has a positive effect mainly on the time that can be saved. One example: The port of Giana Sisters: Twisted Dreams to PS4 was completed within two months. Not even one year before, the port to PS3 took considerably more time.

Especially rookies in console development underestimate this point again and again. The compliance with the TRC can involve a

considerable work effort. This applies not only to obvious tasks like the optimization of the rendering engine which has to be adapted to the respective console; controller inputs, standardized button assignment as well as the user interface need to be adapted to the respective manufacturer specifications. Specifics like the output on the TV screen (overscan, anyone?) have to be taken into account and considered just like literally hundreds of other details.

We can't stress this enough: TRC are complex – and without any experience it can take months to have your project certified.

The submission procedure in general is almost identical for all manufacturers. The developer submits a submission package of the game, gets feedback in the form of a bug report (prioritized according to the seriousness and the specification or recommendation of a fix – for example »must fix« (»mf«) for bugs which absolutely must be fixed). The developer submits the fixed package again and waits for feedback. Wash – rinse – repeat.

The differences mainly involve the individual approach and the toolset. In case of Sony developers face another peculiarity: For each region you want to release your game in you go through the whole submission process all over again. Microsoft acts more »globally« in this respect. If a game has successfully passed the process, it can be published globally.

All manufacturers have in common that you should never assume that each submitted version will be tested with the same intensity: For example, the first feedback may only mention one »mf« bug while the next bug report unexpectedly includes eight new ones. That's something you have to be prepared for – usually, this can be avoided by sticking to the TRC documents.

It should also be mentioned that the QA by the platform owners can't replace your own QA. Although the submissions are usually free of charge, it may happen that you have to bear the costs when you exploit the system.

By the way, this process starts again from scratch with every new patch.

Communication: It's important! It helps!

As mentioned above, Microsoft, Sony and Nintendo care more about the developers these days – you usually get different contacts for each aspect of the release process. They help with questions during development, but also with regard to marketing or the collaboration with the local authorities in countries you want to develop your game in. It wants to take this opportunity to give kudos to SCE Asia: Without the intensive help of Liou, our local contact, who agreed to conduct negotiations with the authorities and saved us from dealing with the Han letters, we would have never been able to release Giana Sisters: Twisted Dreams in China.



The goal of the 2012a Kickstarter campaign was to bring Giana back to life. The ultimate goal from the beginning was a port from PC to console.

Why all that stress?

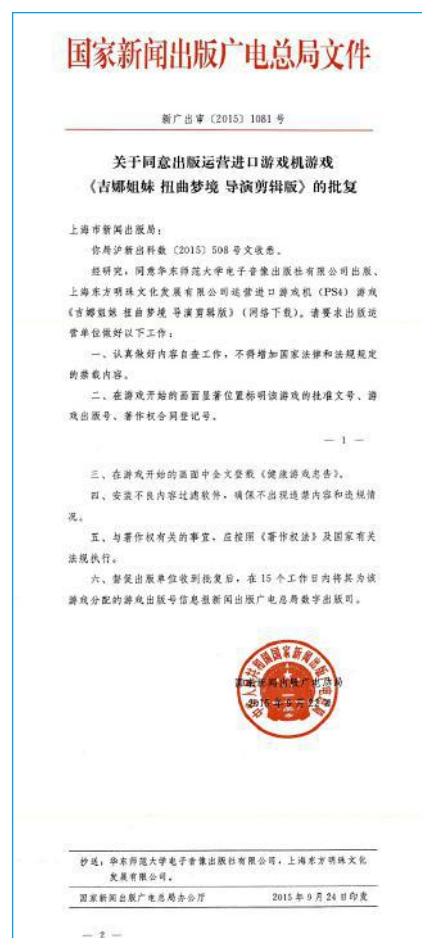
Apart from the technical aspect, indies have to deal with additional tasks like questions regarding age ratings, marketing and localization. Community work. Dealing with the press. Trade shows to present the game. If you can afford it, even working with agencies. The truth is: For every DayZ, Risk of Rain, Speedrunners or Minecraft there are many other games which go away empty-handed. Self-publishing is technically feasible – now even independent of the platform. It's also true that indie titles are popular. However, no matter how hard you work, knowing that »you have done everything right«, success is never guaranteed.

If the effort – and the risk – is that high and the whole adventure may eventually be doomed to fail, why then expose yourself to that kind of pressure?

Speaking for ourselves, all I can say is this: Because it can be worth it after all and because we love what we do. From a technical point of view, porting games has become a lot easier due to the similar architecture of PC and the current console generation. Some console manufacturers actively approach indies and have considerably lowered the initial hurdles. And: The platforms aren't as busy as, for example, Steam. Games on Xbox Marketplace, the PlayStation Store or Nintendo's eShop have a much higher visibility and thus better chances to be sold. Obviously, this also helps us increase the number of potential buyers and reach other groups of consumers.

Last but not least, as a developer you benefit from offering a broad range of products, both in terms of commissioned work and your own productions.

Mischa Strecker



At the end of a long bureaucratic process, the official document is finally issued by the Chinese censorship authorities, which grants the release.



The team of KING Art Games published their fantasy adventure »The Book of Unwritten Tales 2« on a total of ten platforms. What was still an unthinkable thing to do for a small studio a few years ago, is now indeed possible thanks to multiplatform engines like Unity - provided the production cycle is organized adequately and potential problems are known.



Jan Theysen
Game Director
at KING Art Games

Jan is Game Director and author of »The Book of Unwritten Tales 2« and has worked on various user interfaces and control systems. He plays on all sorts of platforms ranging from PC to consoles to mobile - pretty much whatever he can get his hands on.



Arne von Ohsen
is Programmer
at KING Art Games

Arne and his team are responsible for porting »The Book of Unwritten Tales« to ten platforms. He has built numerous versions and has sent terabytes of data across the world. At the moment he is busy developing KING Art's multiplatform toolkit.

When publishing a game on one platform only, all development costs, all the money for assets, animations, speech recordings and marketing has to be recouped with this one version of the game.

We have published »The Book of Unwritten Tales 2« on PC, Mac, Linux, Xbox 360, PS3, Wii U, Xbox One, PS4, Android and iOS (tablets & smartphones). Those are ten ways to appeal to existing fans on their preferred platform, to present the game to new players and to generate press coverage.

The nine ports of the game increased the development costs by less than 15 percent. That is because:

- From the beginning we designed the assets in a way so that they may be used on all systems.
- With regard to the features, controls and the UI we always had all platforms in mind.

- We focused on platform-independent technologies/formats and on a tried and tested multiplatform engine which is Unity.

We also benefited from the fact that we had already ported other projects for last-gen consoles and iOS/Android. The (partly painful) experience we gained from these projects helped us now for BoUT2.

Not every game is suitable for every platform

Of course, it's exceptional to publish a game on ten platforms. The release of BoUT 2 fell together with the change of the console generation, and there aren't a lot of games which are as suitable for a multiplatform release as point&click adventures.

- It must be possible to play the game equally well with mouse, controller or via touch screen.
- The game must run on a broad range of devices (from ten-year old Xbox 360 consoles

to feeble mobile phones to current PCs) and still look good.

- The user interface must work on a TV that's mounted on a wall just as well as on a tiny display that the player is holding in his hand.
- There must be a target audience for the game on every platform.

Porting a game doesn't have to cost a lot of money, and yet you should resist the temptation to get your game onto any available platform just because you can. For example, if a game simply isn't suitable for touch devices, you should spare your fans from creating a bad gaming experience and instead have confidence that they will buy one of the other versions.

One should also stay away from the idea of the »lowest common denominator«. A game which looks bad on PC or consoles just because it runs on mobile devices, too, won't be accepted by players, and rightfully so. The same goes for »pre-ported« games which feel like they don't belong to any system at all because the developer was constantly busy thinking about how to keep the porting effort as low as possible.

Three control systems

We all had to deal with badly ported controls and user interfaces on PC at some point. That's why we recommend to anyone who wants to develop a multiplatform game to plan with three individual control systems and three individual user interfaces right from the start. There is no control system or UI which feels ideal for all three input options on all three types of devices.

The effort to design individual controls and user interfaces isn't too bad, which is why we find it astonishing again and again that a lot of AAA games save expenses in these cases, of all things. After all, poor user comfort or awkward UIs can ruin a game!

The following considerations should be made:

- **What are the standards for games on the various platforms?** How do you »right-click« or display tooltips on tablets? Or how does a controller-navigated character behave when leaving a scene to the left and entering the next scene on the right (with the player still pressing left)?
- **What are the technical consequences of the different controls?** What does it mean for the animation system if a character is to smoothly turn around upon a mouse click on PC and the same character has to react directly to the player's input when navigated by a controller to avoid a »vague« feel to it?
- **What does that mean for the gameplay and the level design?** If your fingers are big and you need to select an item on an iPhone

screen – how big does the click area have to be at least? And will it be too big then for mouse controls on PC? How much space is needed at least between collision objects to allow for a comfortable navigation through them via controller?

- **Are there any device-specific input possibilities which you want to or have to use?** Does it make sense to use the motion sensors or the light in the PS4 controller? Or do you want to support voice control on Xbox One? The »back« button on Android devices should do something useful in the game...

It is necessary to think about these things early in the development process, ideally during pre-production. It's not necessarily required to implement all control systems at the begin-



While we were developing BoUT2, we always had consoles and tablets in mind, and yet we first finished the PC, Mac and Linux versions and only then started to seriously tackle the ports.



Since it's not possible to right-click on tablets and mobile phones, we integrated a context menu in the game for these systems. We also implemented a drag&drop functionality for the inventory because that's what iOS and Android players are used to.

ning, but you have to keep them in mind to avoid problems in the future.

To find out, for example, which controls work best with gamepads you have to run some tests. You hand the controller to players with different levels of experience and watch what they do with it. Feedback such as »When I do this, I would have expected the character to do that« is invaluable.

Three user interfaces

You should do yourself and the players a favor by implementing scalable user interfaces and fonts in the games.

Texts which have a certain size on PC have to be enlarged to 150 to 200 percent for wall-mounted TVs. Mobile devices, too, require larger (per-pixel) fonts due to the incredibly high resolutions.

On touch screens, buttons and other click areas have to be a lot larger than, e.g., on PC. Also a solution needs to be found for mouse-overs/ tooltips and right-clicks which are standard for mouse controls, but tricky for mobile applications.

A lot less elements fit a screen for a console UI, and the menu navigation, too, often differs a lot from PC. Again, you have to really familiarize yourself with other games of this genre and know the standards.

Some platform-specific elements like the controller button icons should be easily exchangeable and you should familiarize yourself with the specifications of the platform

providers – there are rules for the look of the circle, square, triangle and X buttons.

Apart from the different scaling and smaller adjustments the three user interfaces in BoUT2 are quite similar. This is mainly due to the fact that point&click adventures are rather simple games when it comes to user interfaces. The effort it takes for games of other genres to do the required adjustments should not be underestimated.

Quality vs. performance

From a quality point of view, ordinary smartphones and tablets are currently on the same level as the last console generation at max. They have more memory, but the graphics chips are usually weaker and need to display a lot more pixels than Xbox 360 or PS3.

Development is progressing fast in the mobile sector, yet we shouldn't expect the quality of the graphics to reach the level of the current console generation in the next few years.

If you want to offer your players the same high-quality look on all systems, you shouldn't focus on pure computing power, but rather take a smart approach. Games like »Don't Starve« or »FTL« thrive on their style and look the same on each platform – obviously since they don't put heavy strains on the systems from a technical point of view.

With BoUT 2 we took a similar approach and focused on a technology called »projection mapping« where pre-rendered and pre-lit high-poly geometry is mapped onto real-time low-poly geometry. The result are very detailed scenes with a beautiful depth effect, consisting of only a few thousand pre-lit polygons (compared to hundreds of thousands of dynamically lit polygons which would have been required for classic low-poly versions of the scenes).

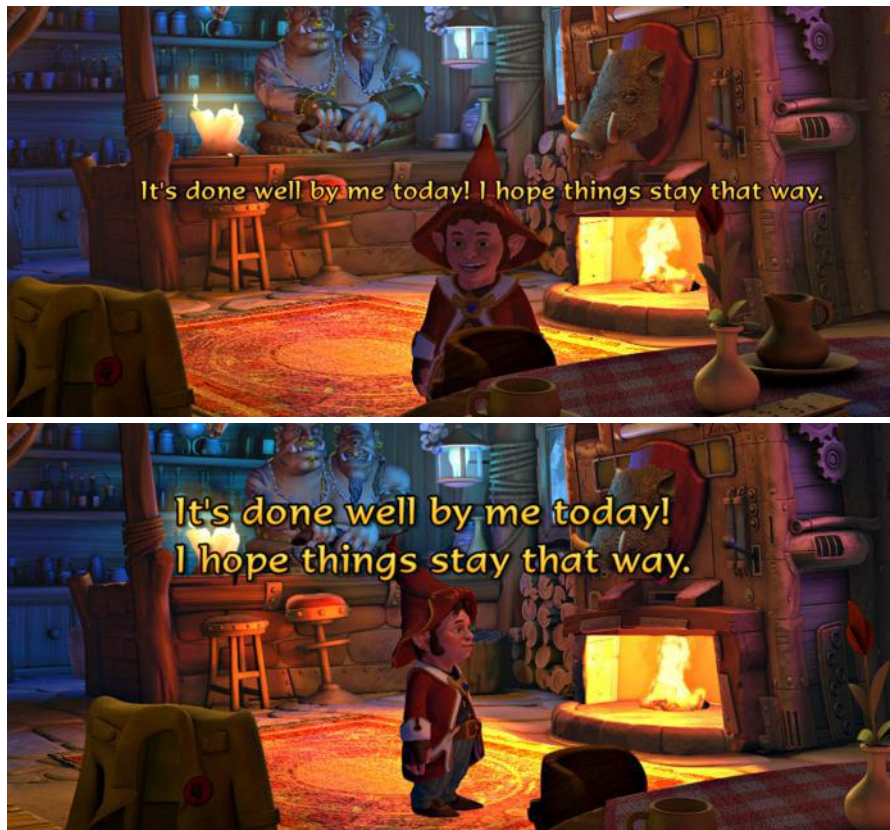
We invested the performance we saved from calculating the backgrounds into the characters, their real-time illumination, effects and shadows. The effort was worth it: BoUT 2 basically features the same (good) look on all platforms.

Herausforderungen

Challenges during the code port

If you look at Unity's multiplatform feature list, you quickly get the impression that all available platforms are only a mouse click away. It's true that Unity does a lot of the work for you, however, every new platform also requires more or less complex adjustments in the game itself.

On the desktop platforms, Unity uses mono runtime for the scripts written by the developer. Mono is almost identical to Microsoft's .NET framework and therefore offers developers a C# compiler with support for a lot of comfortable features like LINQ. The core of some of these features is the just-in-time compilation (JIT) where new code is generated during runtime.



Font sizes differ a lot between PC (top) and consoles (bottom). This becomes particularly relevant when space is very limited – unlike in this example.

Dev kits & resources

In order to be able to work on consoles, you need special dev kits from Sony, Microsoft and Nintendo. The chances for a professional developer to receive a dev kit aren't bad. However, it takes time. Some contacts at the Big Three are hard to reach, and several months can go by between the initial contact and the delivery of the dev kits.

After signing an NDA as a registered developer for a console you receive access to development resources, documentations and tools. We are not allowed to describe or show

any of them here. The NDA is also the reason why there isn't much publicly available information to be found when running into problems. A quick Google search is usually ineffective when it comes to console-specific problems.

Unity offers closed forums for registered developers to exchange experiences and information with other developers. The level of discussion is more professional in these forums than in public forums because usually it's only professionals who get access to dev kits.

On platforms where this isn't supported (e.g. iOS), the code is converted by Unity »ahead of time« (AOT) when building the application. If a JIT feature is used in an AOT environment, it will result in a runtime error and – in a lot of cases – a game crash.

only a few kilobytes of data, the fastest (and most comfortable) way is to do it in a blocking operation. Writing such small files is usually hardly noticeable. Unfortunately, these solutions cause problems on a lot of platforms.

<code>var lastItem = list.Last();</code>	<code>//will result (in some cases) in a crash</code>
<code>var lastItem = list[list.Count - 1];</code>	<code>//will run in all environments</code>

Finding these errors can quickly become very time-consuming since they don't occur in a desktop build or the Unity editor. Therefore, the game must be brought onto the target platform, which can take quite a long time depending on the scope of the project. That's why it pays off quickly to automate and accelerate these work steps by using, e.g., a small test project without huge assets.

There are comprehensive non-NDA documents about the dos and don'ts with regard to AOT online. It is recommended to always keep an eye to this issue during the development.

Asynchronicity and troubleshooting

When implementing a save game system on desktop platforms, which has to write

It's safer to assume the following scenario:

- All write, read and delete operations may take several seconds in worst case.
- These operations aren't always successful and the user then has to receive a clear error information.

Implementing a save game system in a way where all operations are carried out asynchronously and result in proper error information, if necessary, is the ideal foundation for a smooth port.

An asynchronous architecture is not just limited to the save game system though. If the game exceeds a certain data size, it may be required to split it up into several parts. These parts are being installed while the game is



For our strategy game »Battle Worlds: Kronos« we had to create a graphically leaner mobile version because we didn't give it enough consideration in advance – the mobile version was only added as a Kickstarter stretch goal.

A new platform – the first steps for programmers

1. Read documentation and familiarize yourself with the submission process
 - What are the technical requirements of the target platform for my game?
 - How does the certification process work? What is tested and how?
 - How do I get my game ready for the submission? Are there any specifications and limitations?
2. Familiarize yourself with the dev kits and create a test project
 - What do I have to do to make a Unity project run on the console?
 - What settings can I adjust and what effect do they have?
 - How do I receive error information?
3. Get the target game to run
 - Will there be problems with my game if it is compiled AOT?
 - Are there performance or memory issues?
 - Are there any shaders or effects which don't work on the target platform?
4. Set up build server, create automated tests (benchmark system)

already running, so it may happen that, e.g., a player wants to enter a level which hasn't been installed yet. In this case, the player needs to be notified and the game needs to wait for the level to be installed.

Also, events may occur which don't have anything to do with the actual game, but which the game needs to react to at any time. On Xbox One, for example, the current user can log off at any time. In an ideal case, the game would then be stopped and a message displayed.

Implementing the platform APIs

Platform-specific features like cloud save games, achievements or in-game purchases require the use of the respective platform API. These APIs are usually written in the native platform language and therefore can't be readily used with mono in Unity just like that.

Of course, there are set mono implementations for most of these functions – some of which are integrated directly in Unity, while others can be loaded via plugins. If you deviate from the standards and you want to use exotic features, you may be required to write your own native plugin.

Your own code calling on the platform API has to be implemented again for each platform. It's therefore recommendable to call on the API in a central exchangeable place and not to spread these calls throughout the entire code.

Memory optimization

Memory optimization plays a minor role on the current-generation consoles compared to the last-generation systems. Yet, it's worth investing some time in the memory optimization since all versions of the game benefit from lower memory usage and shorter load times.

- All assets present in a Unity scene are written in the delay line memory when the scene is being loaded. One should therefore make sure that only those assets that are actually needed are in the scene. Spoken text, music etc. should ideally be loaded »on the fly« – and then unloaded again.
- The import settings of all assets have to be correct. In order to get The Book of Unwritten Tales 2 to run on the last-generation consoles, it was necessary, for example, to halve the texture resolutions of all background textures. It is possible to set a maximum size per platform.
- Code stripping saves memory and accelerates the game's load times. If this Unity feature is activated, the code will be analyzed during building, and classes and methods which haven't been referenced will be removed. A dynamic request for removed methods (e.g. through reflection)



PlayStation 4 and Xbox One require that large games can be started already while the game is still being downloaded/installed in the background. You either take this into account at the beginning of the development or you will be facing a massive problem at the end...

after the code stripping leads to a runtime error though. That's why it is necessary to note these classes and methods in a blacklist before building.

- If a lot of new objects are instantiated in short intervals (e.g. through initiating objects in an update call), the mono heap increases. The heap has a fixed size in the main memory. If the size of the heap exceeds its currently allocated budget and the garbage collector can't free up enough memory, new memory will be allocated from the main memory. The size of the heap budget can only increase in Unity. Once allocated, there is no way to return a heap memory to the system again. This may cause problems which can be difficult to diagnose and may only arise late in the game.

Performance optimization

In *The Book of Unwritten Tales 2* we are offering four levels of quality. On platforms with pre-defined hardware the game is permanently set to one of the quality levels. On platforms with different hardware like the PC or Android devices the user can decide between the quality levels.

Depending on the chosen quality, post effects like depth of field-blur or antialiasing will then be deactivated or reduced. Also, depending on the quality, different shaders will be used for our backgrounds or characters.

In addition, we created the option to tag objects, effects and shaders for a certain quality level, based on which higher-quality particle effects or more detailed objects can be used.

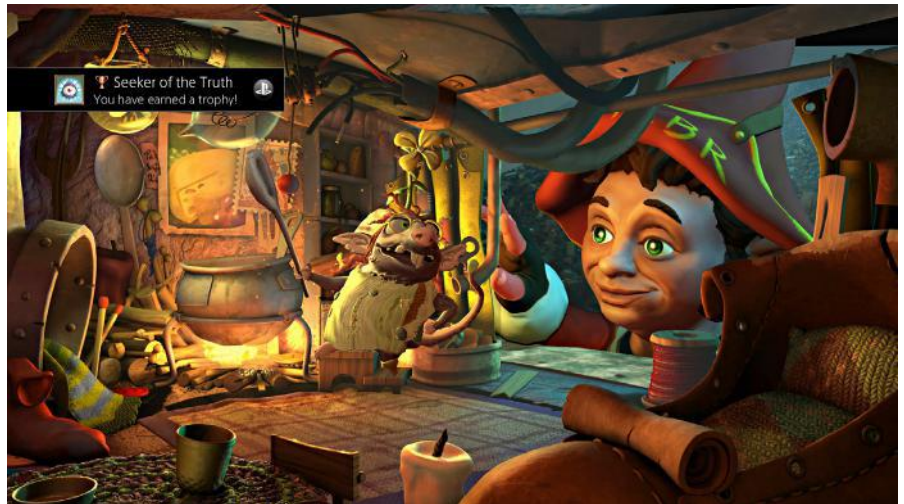
The final way to achieve better performance on low-end hardware is to reduce the resolution of the game. While this step leads to significant improvements especially on mobile devices, it also considerably reduces the quality of the game.

The performance and memory optimization of the current console generation for *BoUT 2* didn't take a whole lot of effort. Since we already had a high-end version in the PC version and a low-end version in the Xbox 360 version, all we had to do was to find ideal settings for PS4 and Xbox One. Our automated benchmark (see box-out) was of great help here.

Console submissions

A PC/Mac/Linux game is »ready« when the developer and/or the publisher says it's ready. A disc image will then be sent to the press plant and one version will be uploaded to Steam – that's it. This is completely different in regards to console games. They will first have to be thoroughly scrutinized in a certification process by the platform owner.

There is very detailed documentation for registered developers stating which tests will be carried out and how they work exactly. Depending on whether the game has an online/multiplayer functionality, there are



The game logic calls on a »grant achievement« method if the player has completed an achievement. The method then informs the achievement systems of Steam, Xbox 360, Xbox One, PlayStation 3, PlayStation 4 and iOS Game Center.



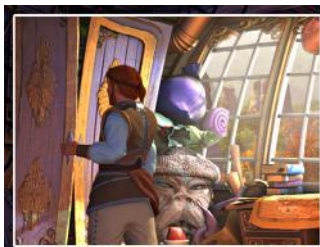
BoUT 2 doesn't load dialogs, animations, sound effects and music until they're actually needed - an optimization which was necessary for the memory-restricted last-generation consoles, but which also saves load times for PC and the current consoles.

over a hundred test cases which all have to be implemented and tested.

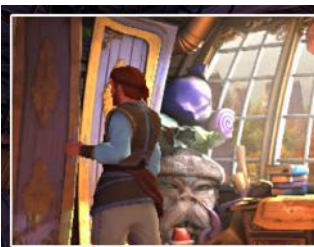
One should familiarize oneself with the certification process and know exactly what is required, both in terms of the game and the accompanying documents for the submission. Basically, it's the platform owners' intention to guarantee a constant level of quality and usability of the console games.

Depending on how busy the certification team is, a submission can take up to several weeks. If bugs are found, the certification process needs to be completed all over again. It's therefore necessary to allow enough time for this process, and it's realistic to expect your game to fail at least one submission.

During the console development of *The Book of Unwritten Tales 2* it clearly paid off using an external QA team. There are a number of companies that are registered with the console manufacturers, specializing in testing games with regard to the requirements of the certification process.



Current-generation consoles: High-resolution textures, smooth shadows, antialiasing, light and other effects, real-time reflections.



Lowest quality level setting for mobile: 50 percent texture size, mobile compression for textures, no antialiasing, no shadows, simpler light and other effects, simpler shaders.

Don't underestimate the administrative effort

Even though the technical effort for a game port to a new platform is not as high as it used to be, one shouldn't underestimate the extensive administrative effort.

Just like each additional language version or game edition means additional organizational effort, each additional platform, too, means more work.

- The effort it takes to test the game is a lot higher. If all versions of a game are to be released simultaneously, you need a lot more testers all of a sudden.
- Game versions and patches must potentially be built and tested simultaneously for numerous different game versions – the deadline, as always, being »yesterday«!
- Tons of gigabytes of data must be uploaded. This may sound like nothing special, but if you need to upload 60 GB of builds

overnight, you better have a fast and stable connection or the possibility to divide the upload between different locations.

- You have to communicate with numerous parties: With the platform owners (in the case of Sony even with Europe and North America separately), with digital sales platforms, with Unity, with the publisher, with the testers – each additional platform causes new needs for clarification. And since the partners are often in different time zones, you can't always expect a quick reply.
- More platforms also mean more PR work, more marketing materials and later more players with platform-specific problems or questions...

Lessons learned

If the decision is made to develop a multiplatform game, there are many new factors which need to be considered. If you are aware of this

KING Art Games

Founded 15 years ago, KING Art Games is one of the long-established game developers in Germany. The development team is approximately 35 people strong and is working on PC, console, mobile and browser games. »The Book of Unwritten Tales 2« – the fifth adventure of the Bremen-based company – won the German developer award for »Best console game« in 2015.



Each version of BoUT 2 was tested multiple times at QA provider Quantic Lab, overall we did more than 20 test rounds (compared to the two or three that we would schedule for a PC-only title).

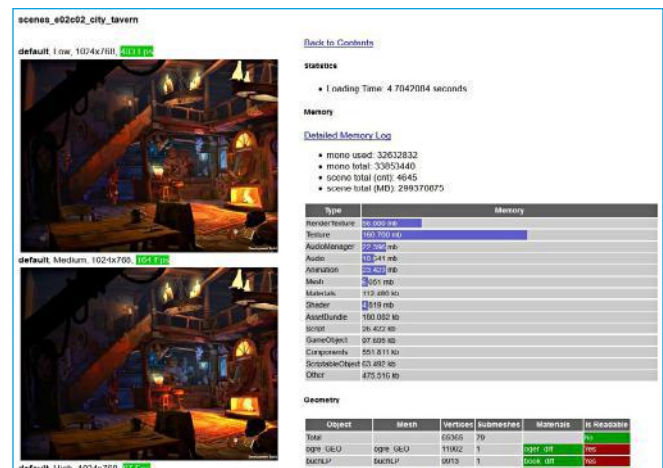
The lifesaver: Automated benchmarks

Unity allows to create good-looking scenes quickly and almost without any previous knowledge. What's great for graphic designers is often a problem for coders who have to keep an eye on the memory usage and performance. It happens regularly that too »expensive« effects or objects are implemented, performance-critical things are set incorrectly and features are destroyed unnoticed. In order to keep an eye on the current memory usage and the performance during the development, we outfitted our game with an automated benchmark mode. If the game is started with the corresponding command line argument, it begins to successively load all scenes that can occur in the game. Every scene is displayed in all quality levels and a screenshot is taken of each. This way, missing textures, broken shaders or lighting problems can be detected quickly.

Also, the frame rate is recorded and the memory usage of the scene is analyzed. For this analysis, an inquiry is issued about all objects that are currently in the Unity memory via »Resources.FindObjectsOfType(typeof(UnityEngine.Object));« which are then measured via »Profiler.GetRuntimeMemorySize(object);«.

All data recorded this way will then be summarized in an HTML report which is available to all team members. After each successful build of the project, the benchmark will automatically be started on a dedicated computer which roughly meets the minimum requirements of the game, which means that we can have a detailed report which isn't older than 1 or 2 hours about the memory usage and the performance of the game whenever we want.

The benchmark also helped a lot with the hardware compatibility test on Android devices. It was run simultaneously on dozens of devices, and without wasting too much time, we received detailed feedback about which scenes caused problems on which systems.



The comfortable HTML report of our automated benchmark: Pink-colored objects indicate a shader problem. There are many graphics chips which don't get on well with every shader for some reason, especially on mobile devices.

additional effort and bear in mind the potential sources of error, Unity offers you an engine which does a lot of the work for you in terms of reaching new platforms.

In our experience a game port to several platforms benefits all versions of the game. Everybody likes to have controller support, a scalable user interface, comprehensive memory, performance optimizations and reasonable error messages. Well. Error messages not so much. But you know what we mean.

To make it even easier in the future to develop games for several platforms, we summarized our experiences in a platform tool kit. This tool kit can be implemented in a Unity game and acts as a link to various target platforms. It frees developers with only little experience on the various platforms from a lot of concerns and reduces the number of potential bugs.

It seems we won't be releasing another game for ten platforms in the near future, but we still feel well-prepared for future developments for five to seven platforms!

Jan Theysen



Our current game, The Dwarves, won't be released for last-generation consoles or mobile devices, »just« on five or six platforms (let's see what Nintendo NX is capable of).



Making Games 02/2016

RELEASING AN INDIE GAME ON 3 CONSOLES AT ONCE AND FAILING FINANCIALLY

After releasing a former game jam project on Ouya, the team of Juicy Beast decided to port »Toto Temple Deluxe« to three consoles at the same time, which turned out to be a financial fail instead of a big success. Here is what the team learned after the release.



Yowan Langlais
is Co-Founder, Art Director
and UI Designer
at Juicy Beast.

Yowan is the co-founder, art director and UI designer at Juicy Beast (www.juicybeast.com), a small indie studio born in 2009 in Montreal, Canada. He specializes in UI, UX and game design, but has always been fascinated by minimalism, elegance and computer generated art in general.

[@wyel](https://twitter.com/wyel)
[@juicybeast](https://twitter.com/juicybeast)

In May 2013, we attended Toronto Game Jam and made a local-multiplayer game called »Toto Temple«. About a year later, we released a first version called »Toto Temple Deluxe« on Ouya. We kept improving the game afterwards and released the final version of Toto Temple Deluxe on PS4, Xbox One, Wii U and Steam on September 29th, 2015. The game didn't sell very well, and we think we know why. Here is what I will talk about on the following pages:

1. Why invest two years on such a small game?
2. Was the game at least fun?
3. Gameplay analysis
(how about those other popular games?)
4. Marketing experiments
5. Why didn't the game sell?
6. Next time

1. Why Invest Two Years on such a Small Game?

The truth is, we never actually planned to do anything with Toto Temple (the game jam version). We were presented with a chain of opportunities, and we simply decided to take each one of them as the development went on.

Year 1: Working with Ouya

The first opportunity we took after the Toronto Game Jam (where the game ended up being one of the favorites), was taking Ouya's money to make a full-fledged game out of Toto Temple. At the time, Matt Thorson's »Towerfall« was Ouya's killer app, and they wanted to cash in on that local-multiplayer trend (who wouldn't). We took the money and made a bigger version of the game, which we then called Toto Temple Deluxe.



The deal was simple, get a bigger game (and more experience) in our portfolio without spending our own money. The game went live on Ouya, people enjoyed the Deluxe version, we even received really nice feedback from the press at E3 (at Ouya's booth).

Here we were, a couple of weeks after the Ouya release, with a »console friendly« game in our hands, which technically didn't cost us anything to make. For us, this was something unusual: We used to make small Flash and mobile games, and we now had a »big« game that could be played in the living room with actual controllers. It felt like a big deal, as we could remember actually saying a couple years ago that »we would probably never be able to make console games«. At the time, we didn't have the skills to develop for consoles, and Unity wasn't really a thing, yet.

Year 2: Porting to Consoles

This is where we took the opportunity to get in touch with the three giants. We arranged meetings at GDC and pitched Toto Temple Deluxe to the three console owners.

They all liked the game and things moved forward pretty quickly after that. We officially became PS4, Xbox One and Wii U developers, which still felt like a really big deal. We were in the big leagues now.

Being total beginners in the console market, we quickly realized that we would never get an interesting exclusivity deal with any of three console owners, since the game was already out on Ouya.

Knowing that, we decided to release the game on all three consoles. The decision was mainly based on two points:

- Our thought process back then was that if we can't strike a good exclusivity deal, then let's do all three consoles for three times the revenue.
- We also thought that we probably wouldn't get the media's attention three times for the same game (the three releases), so let's release the game on all three consoles at the same time. That way, we would get decent attention one time and speak to all console owners at once.

2. Was the Game at least Fun?

We genuinely think Toto Temple Deluxe is a lot of fun, under the right circumstances. We designed the game to be as family friendly as possible, meaning that anyone should have a minimum amount of fun, no matter how skilled their opponents are (auto-balance system, etc.). It's also designed to be played with real people, in the same room, on the same couch. The more yelling, the better!

We also witnessed genuine fun and engagement from players who tried the game in public events, like at PAX and Comiccon.

3. Gameplay Analysis

We don't think the game didn't sell well because it wasn't fun. On the other hand, Toto Temple Deluxe is still quite different from those other popular local-multiplayer games.

Intuitive Controls

One of the first things you notice when you first play Toto Temple Deluxe is how tricky it can be to effectively move around in the game (by using the dash mechanic).

Even if we did our best to explain the controls in the most intuitive way we could think of, there's still a difficulty curve that you need to overcome to really feel at ease while moving around.

It definitely comes in the way of our goal to make the game family friendly. We should either have made the controls simpler, or aimed at a more »experienced« crowd from the start. Not both.

In Towerfall, for instance, everyone can be deadly by simply pressing the »shoot« button once. It's that simple. In Toto Temple Deluxe, you need to:

1. Get close to the goat carrier.
2. Use a combination of two buttons to dash towards the goat carrier.
3. Hope that you were aiming right.
4. If not, you need to turn around and start over.

It's definitely not a problem once you get the hang of it, but it's not making the game as easy to get into for new players.

About Toto Temple Deluxe

Toto Temple Deluxe is a fast-paced, local-multiplayer party game in which players must steal an egg-laying goat from their friends and try to keep it on their own head for as long as possible.

Dashing Gameplay

Infinitely dash in any direction to swiftly move around and powerfully head-butt the goat carrier to steal its four-legged booty. Then you have to escape your opponents.

Classic Mode

In Classic mode the player has to gather 3,000 points first by holding on to the goat, collecting coins hatched from goat eggs and harnessing the power of power-ups.

Bomb Mode

In Bomb Mode the player has to be the first to grab the explosive goat and keep it long enough for it to detonate near his/her rivals. If the goat is lost, the timer resets. This mode can be played as teams or in free-for-all.



Toto Temple Deluxe is a game that is meant to be played in local multiplayer. It was created during a game jam and was later developed into a full game.



The Show

Tricky controls can actually be a good thing if they're funny to watch. Our friends from Breakfall did just that with their game »Starwhal«. The game is pretty hard to control, but it's part of the show, as you watch your starwhal wiggle its way towards your opponent's heart.

In Toto Temple Deluxe, it's actually pretty hard to make sense of what's happening on the screen if you don't know the game already.

Toto Temple Deluxe doesn't really give a »good show« by default. You never see spectacular kills like in Towerfall or Starwhal, and not many actions are worth their own replay or slow motion sequence (there are, but they are harder to spot). It's not making the game less fun to play, obviously, but it's making it less fun to watch. It might not be that attractive to YouTubers and Twitchers for this reason.

Intensity Variations

In most competitive local-multiplayer games where you can »eliminate« players in some way (like Samurai Gunn, Nidhogg, Towerfall, Starwhal, etc.), you often find significant variations in intensity.

The tension goes up as you're fighting or trying to survive, but once you or the other player is eliminated, there is this drop in intensity that lets you breath (and celebrate) for a couple seconds, which feels right. On paper, it would probably look like in [figure 1](#).

In Toto Temple Deluxe, the intensity graph probably looks more like in [figure 2](#). You get micro rewards or satisfactions every time you steal the goat or strike a power-up, but the game doesn't stop nor celebrate it that much, it just keeps going. Your real reward is at the end, when you finally reach 3k points and win the match.

We don't think this kind of intensity is necessarily a bad thing, but it forced us to reduce the time of a match to a maximum of about two minutes. Anything passed that would really

drain you out mentally, and a single game was usually more than enough.

All of these elements are probably not responsible for the game not selling well, but they do have an effect on the actual feel and accessibility of the game.

4. Marketing Experiments

We tend to do everything in-house (like contacting media, official websites, video trailers, community management, etc.). We've never been fans of spending money on marketing for things we could do ourselves. We also try to come up with creative solutions to market our games to counter-balance our low marketing budgets.

For Toto Temple Deluxe, we really tried to do more than what we usually do. We went to PAX for the first time (remember, we used to make Flash games), and we also paid for PR for the first time as well. While going to gaming events and paying for PR services is pretty traditional, we also experimented a bit with other things like real goats and automatic Twitch raffles.

Paying for nothing PR at PAX East

Our first experience with PR wasn't super useful in the end, to be honest. Even if we managed to get those services for a reasonable price (being »new customers« to a super friendly PR company), the end results left us pretty unsatisfied.

The services they provided were good, don't get me wrong. We managed to get a couple interesting meetings scheduled at PAX (big websites), but the snowstorm that happened that week made some of them cancel. When we got back, the PR company helped us re-schedule new online meetings with the sites who canceled, but none of them answered our multiple emails.

One other big website that actually came to the meeting wrote an article about the upcoming game following the event, but they never bothered writing a review once the game was released. We contacted them multiple times with no luck, which is weird, considering they already showed an interest in the game.

Giving away \$17,000 worth of Toto Temple Deluxe at PAX East

To put things in perspective a bit, we originally designed our PAX demo so that matches would be really quick. That way, we'd have a better flow and have more players try the game. Since matches wouldn't last long enough for the crowd to feel any major engagement, or for hype to build up, we needed a better way of getting spectators more engaged in the game as they were looking at it.

Normally, only the four current players would be really engaged in the game. Maybe they were with a friend, and maybe that friend would get engaged and cheer a bit, but we felt like it wouldn't be enough to build enough engagement or hype overall.

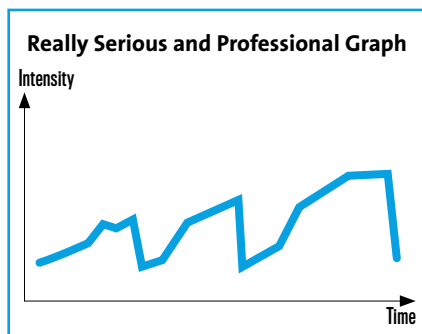


Figure 1: More variations in tension during gameplay means more time to breathe for the player.

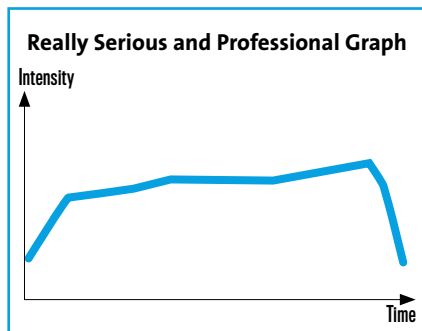


Figure 2: In Toto Temple Deluxe tension is more or less always on the same level and the real reward comes at the end if the player wins the match.



»Starwhal« is pretty hard to control, but it's part of the show, as you watch your starwhal wiggle its way towards your opponent's heart.

We then heard of a strategy that the guys from Vlambeer were doing at events for »Nuclear Throne«. They would have hourly challenges where everyone watching would get a free copy of the game if the current player managed to beat the game. That would normally encourage people to stay and look at the game. They would have something to win (or lose), so they should feel engaged.

We thought it was really clever and it would totally fit our needs, but we decided to push the idea a bit further. Instead of asking players to come back every hour to have a chance of winning a free copy of the game, we decided that every – single – match would be an opportunity to win a copy of the game.

With that in mind, we needed something to link the giveaways to. Ideally, something that wouldn't happen every match, but only once in a while so we don't end up giving away 50,000 copies. That's when we came up with the showdown system:

Toto Temple Deluxe has two main game modes. The Classic mode, where you fight for an egg-laying goat to make points, and the Bomb mode, where you fight for an explosive goat in order to kill your opponents with the blast.

Since we needed something to link the giveaways to, we figured we could use the bomb mode for that. Here's how the system works:

The winner of each match (in Classic mode) would be automatically thrown in a 1 vs. 1 bomb showdown against one bot (set at hard). If that player manages to kill the bot two times, everybody watching would get a free copy of the game.

It was also a good way of having players experience both modes and discover the game's content without having to come back to play a match in the other mode.

This particular experiment (the showdown system) was really successful at the event. We attracted big crowds and got noticed by a lot of players. The only problem is that we didn't get noticed by that much press, which would have helped promote the game later down the road.

The showdown system worked so well with players that, that we replicated it at future events (like the Montreal Comiccon), and received the same type of response (only on a smaller scale).

It's definitely something we would try to adapt to our future games, and we encourage anyone to remix this idea to fit their needs.

The Real Goat Bundle

When the game came out, we had a plan to join a good cause by promoting the »Oxfam Unwrapped program«. Oxfam are offering to buy real animals (like chickens, goats, donkeys, etc.), to provide communities in need around the world. It's a sort of donation, basically.

What we did is we committed ourselves to buy real goats in the name of anyone who would purchase the »Real Goat bundle«. The



The showdown system helped that the booth of Juicy Beast at PAX East was pretty much crowded.

buyers would also get four copies (Steam keys) of Toto Temple Deluxe to share with their friends.

Despite being really well adapted to the theme of the game (goats) and being original to some extent, the Real Goat bundle didn't sell at all. We think it's probably due to three factors:

1. The steep price of the bundle (\$60 USD), which basically just covers the price of the goat. We make no money with the bundle, which actually fits the »good cause« idea. We're giving away copies of our game in exchange for a donation to a charity organization.
2. We only felt comfortable giving away PC versions of the game. The game is more enjoyable and popular on consoles, but we're not allowed to distribute console codes that way.
3. The game doesn't support online-multiplayer, so four copies of a local-multiplayer game might feel less interesting a deal if it doesn't let you play with your friends online. We were aiming to give buyers an opportunity to simply gift the game to their friends, but we completely understand the ambiguity.

Bot Bet Battle

Following the release of Toto Temple Deluxe, we organized a one-day Twitch event called »Bot Bet Battle«.

We had the game's bots fight themselves in a classic tournament, and viewers could vote for the bot they thought would win. Everything was AI controlled, so we had no input over which bot would win.

The number of votes would update in real time (or almost), the names of the voters would pop on the screen for everyone to see, and the tournament would start when enough votes were entered to fill the »votes bar«.

At the end of the tournament, everyone who voted for the winning bot would automatically receive a Steam key in the chat system (as a whisper, obviously). You basically had a chance out of four to win a free copy of the game.

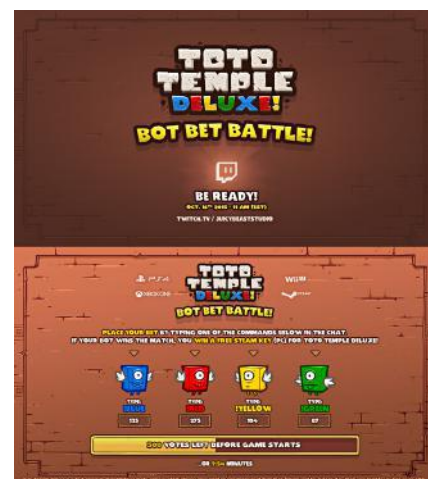
We would also ramp up the amount of necessary votes from round to round, so that viewers would invite their friends to accelerate the process.



A yellow warning sign like this which the team used at PAX East is a pretty good way of getting people's attention!



The »Real Goat bundle« consists of a donation of \$60 to Oxfam, who provide a real goat to communities in need and players get four steam keys for Toto Temple Deluxe in return.



In a Twitch event called »Bot Bet Battle« the matches were played by bots and spectators could bet on which one would win to win a steam key themselves.



When Toto Temple Deluxe was released some of the developers who were present in the game as cameos helped to promote the game via Twitter.

We gave away about 1,400 Steam keys during the event. Knowing that a single account couldn't win twice, and that a lot of viewers never won a key, we know we reached a lot more than 1,400 viewers.

We promoted the event on our social pages and a bunch of related subreddits, so there was probably a decent percentage of »new customers« in the lot.

Even though the event was fun, and that we probably managed to reach new followers, we barely noticed any bump in the game's sales.

Cameos

This one took us a lot of time and grew out of proportion pretty quickly. When we started working on the Deluxe version, we knew we wanted to include cameos from other games as a replacement for the goat (what you're fighting for). By other games, we mean games made by friends of ours and other developers we admire and respect. We also wanted to include people in there (not just game characters), like the developers themselves.

All of this was supposed to be a small and cute addition to the game. We were adding unlockable cameos from games we love, and everyone from the team started to pitch in. This is where everything exploded. We created around 150 cameos (and by we I mean JP, our artist), ranging from game characters to their developers themselves.

We reached out to every single developer to introduce our idea and get an approval if they liked it. Some of them were already our friends, which was easy, but a lot of them were big developers we never spoke with before. Sometimes it took a lot of effort and contacts to get the missing email addresses.

We wanted to do things in a clean way, so we wrote a personalized agreement for each of them, stating that the cameos were just cosmetic additions, and that we would never sell them or make money directly from them, etc.

The agreement itself let the devs change the name of the cameos, decide on which consoles they would appear, pick the secret code to unlock them, and if they agreed with the whole idea or not. A digital signature was required, and boom, everything was set (no PDF, no emails).

A couple of months later, on the actual release date, we've seen a lot of those devs share their cameos (and their secret code to unlock them) on Twitter and Facebook. Not all of them shared though. There was obviously no way to predict who would share and who wouldn't, but we still saw a LOT of tweets that day.

It's pretty hard to calculate how much visibility we gained through the cameo system, as we can't access other dev's Twitter stats, but we probably reached a lot of people who wouldn't have been aware of the game otherwise. What we can calculate though, is how many new connections we made with developers we admire, and we're pretty happy about it.

Newsletter

A newsletter system is nothing new, but we also toyed with it for the first time. Okay, maybe the second time. Remember that part where we gave out free Steam keys at events? Well they weren't really free. We actually asked people to subscribe to our newsletter in order to receive the key.

We even created a sweet little backend system (with the help of our good friend Devine Lu Linvega) that automatically distributes the keys without having us send a single email. We just need to upload the keys to a database, and the system handles everything.

1. Users need a unique code printed on the »key card« they won.
2. They join our newsletter by entering their email address and unique code in a Mailchimp form.
3. Mailchimp sends the info back to our backend system, where we validate the code and register the email if it's good.
4. Users automatically receive an email with their key in it.

We managed to gather 1,437 email addresses around Toto Temple Deluxe (not just with free keys), which all received a single email on the release date (we don't like spam). Out of these 1,437 emails we sent, 493 (34 percent) were opened, and 72 (5 percent) got clicked. Based on certain benchmarks, these numbers aren't that bad!

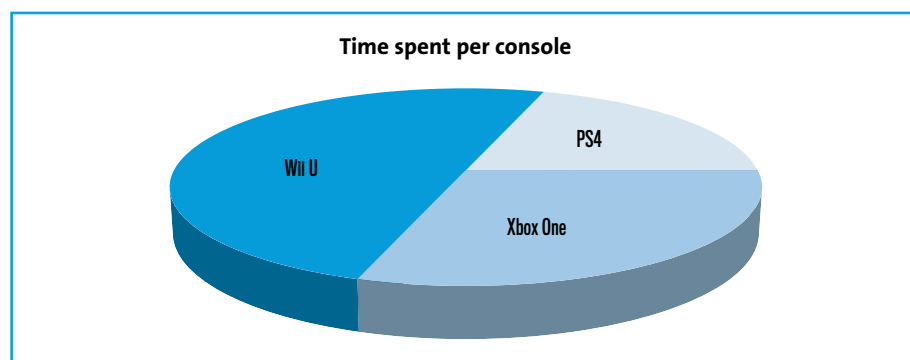


Figure 3: This chart shows how the team spent their time on the actual console ports.



Some examples of cameos that made it into the game.

5. So, why didn't the Game sell?

The game was fun and players were genuinely engaged at events. We reached out to a lot of press with the help of a PR company, but barely got any coverage. We tried different and original marketing tactics, but they didn't result in many changes in the sales. So, what happened?

There's a good possibility that the game didn't have good enough «hooks» to captivate people's attention. Sure, «fighting your friends to put an egg-laying goat on your head» sounds funny and weird, but it doesn't really give you a clear idea of what it's like to play Toto Temple Deluxe.

At the end of the day, we think the biggest factor is because it's a local-multiplayer game with no online play. The game is aiming at a pretty niche audience by requiring actual human friends to play, and we can't ignore the impact it has on sales. A quick look at comments on YouTube, Reddit and such, and it's obvious that a lot of people are simply not buying the game for this very reason (that along a lack of solo experience).

To confirm this idea, one of our contacts from a very well-known gaming site explained to us why they wouldn't write an article on Toto Temple Deluxe. Apparently, it's simply because local-multiplayer news doesn't generate enough clicks to be worth it. We totally appreciated the honesty!

On top of all this, we also think the game came out almost two years too late. Back in 2014, when the local-multiplayer boom was happening, Toto Temple Deluxe's development felt more logical.

Today, it feels a bit out of place, as we think a lot of players might have bought popular local-multiplayer games in the past, then realized that they would play them less and less frequently. We definitely can't blame them, since it's pretty much the same for us.

Obviously, these are all based on our current knowledge and experience, and probably don't apply to all local-multiplayer games. If you're

reading this and you're thinking we missed a point, or have a different opinion on the subject, we'd love to hear it!

Porting to 3 Consoles at Once

Releasing the game on the three consoles at the same time felt like a good idea at the time. We were only thinking about it from a revenues and media coverage standpoint, which was obviously a big mistake (for a local-multiplayer game released two years too late with no online mode, at least).

The Actual Ports

The worst part in all of this is probably that our only programmer, Alex, did all the heavy lifting by himself (the guy's a freaking machine). Making the game run smoothly on all three consoles took him a good amount of time and energy, especially during the certification process.

To give you an idea of the difference between the three consoles, [figure 3](#) shows how we spent our time on the actual ports over the second year of development.

Considering that we made about equal revenues on each consoles so far, you can see that one was definitely more expensive to port to. This might change in the future though, as this was our very first time working with consoles. We also received a lot of help from our friends from Chainsawesome Games for the Xbox One port, which saved us a lot of time and headaches!

Another factor that greatly extended the port process was having to jump back and forth between the different versions of the game. Yes, Unity makes it easier to port to multiple platforms, but the requirements for each console were so different that we ended up having three different Unity projects / branches to support them.

We lost a LOT of time patching little mistakes here and there, like details we added or fixed on one version but forgot to include in the other, etc. With only two team members skilled enough to handle the «technical stuff», you can surely imagine how this inefficient

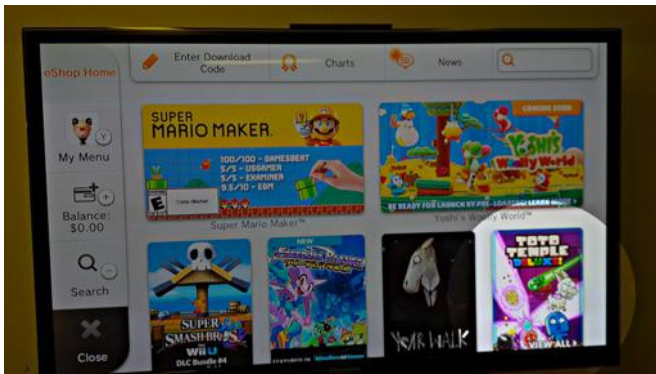
Juicy Beats



Juicy Beast was co-founded in 2009 by longtime friends Yowan Langlais and Dominique Lemire-Nault. Programmer Alexandre Dazé-Hill and illustrator-animator Jean-Philippe Côté joined the ranks shortly after the legal birth of the company. Fresh out of school and with no work experience whatsoever, the four friends took the big leap and released over ten Flash games in the following years, including the successful «Burrito Bison» series and «Knightmare Tower». After releasing Burrito Bison on iOS with the help of developer Ravenous Games, they slowly shifted their attention towards new markets, and replaced their old friend Flash with Unity. Now with the ability to export to almost every platform, they completely revamped Knightmare Tower and released it on Steam, Ouya, Android and iOS, where it got featured by Apple as one of the Best Games of 2013.

In May 2013, Juicy Beast participated in the Toronto Game Jam (TOJam) where they made a local-multiplayer party game called «Toto Temple». The jam game got Ouya's attention and they offered their financial help to develop a full-fledged version of Toto Temple. In July 2014, the new and bigger version now called Toto Temple Deluxe was released on the Ouya store.





Of the three console manufacturers, Nintendo was the only one which featured Toto Temple Deluxe in their store.

workflow became extremely messy during the certification process.

Burning Out

We spent two years, full time, working on Toto Temple Deluxe. The first year was about developing the game and its content, while the second year was about porting it to three different consoles at once. That second year is where we started to feel the burn.

We're a team of creative people, and we get our motivation and energy by exploiting that creativity. We like to create things, invent worlds and mechanics. That second year of development was all about paperwork, unusable console portals, technical problems, bugs and desperate searches for information (consoles and their documentation are not really easy to navigate).

Once we shipped the game on September 29th, we all went home and did nothing for at least a week. Porting Toto Temple Deluxe really drained us out, especially for Alex who did an inhuman job all by himself (seriously, a machine).

The process of porting the game was hard by itself, but following up a year of non-creative work with a launch that doesn't make much noise, nor sell a lot, was pretty hard on all of us. We're slowly getting back in shape for a new project (winning the prize of the most addicting game with »Right Click to Necromance« for Indie Speed Run shook things up a bit), but we can still taste the bitterness of the unsuccessful launch.

No Deal, no Help

Back in the days, when we were young and naive, we pictured console storefronts as big marketing machines with lots of users and traffic (remember, we were doing Flash games). We realized pretty quickly that it's not how it works!

Since we couldn't really strike an exclusivity deal with any of the consoles, with the Ouya version already available, and we were porting to all three consoles at the same time (no »semi-exclusivity«), Toto Temple Deluxe probably wasn't perceived as a really interesting deal for any of the consoles. Sure, they were getting a new game for their system, but that was pretty much it.

Because of that, we weren't really able to get a featured spot on any the storefronts, except for Nintendo who actually featured us on both the American and European front pages. Thanks, Nintendo!

What's also interesting is that being featured twice on the Wii-U-eShop actually ramped up the sales of the game to almost equal numbers with the other two.

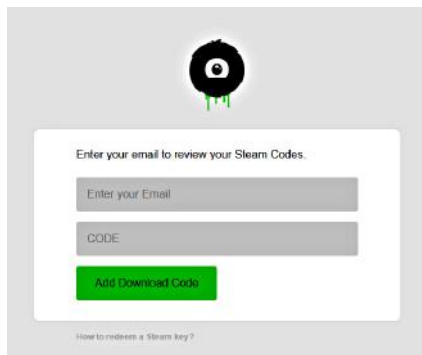
6. Next Time

We've learned a lot with Toto Temple Deluxe, and there's a lot of things we'd do differently (or not at all) for our next game:

1. **No more local-multiplayer:** They're easy to make in a game jam, but super hard to market if you don't add online play.
2. **Strike a deal first:** If we're to make another game on console, we'll make sure to strike an exclusivity deal first, so we can get a bit more help and visibility from the console owner.
3. **Only one console:** It goes without saying that we will only do one console if we strike an exclusivity deal, but even without that, we wouldn't port to three consoles at once ever again. It's a LOT of hard work that didn't really pay well in the end.
4. **Design from a marketing perspective:** Visibility is getting harder and harder to come by, so we will try to design our games with marketing in mind from day one. Having a solid hook and making sure that there's a potential audience for your game is super important, as Ryan Clark from Brace Yourself in his blog »What Makes an Indie Hit?: How to Choose the Right Design« explained.

I think it's easy to think of Toto Temple Deluxe as a failure. Even worst, a waste of time and resources. Sure, the game only generated a fraction of what we spent on it, but we've learned a lot. Now it's time to get up, dust off our shoulders and get the creative juice flowing again!

Yowan Langlais



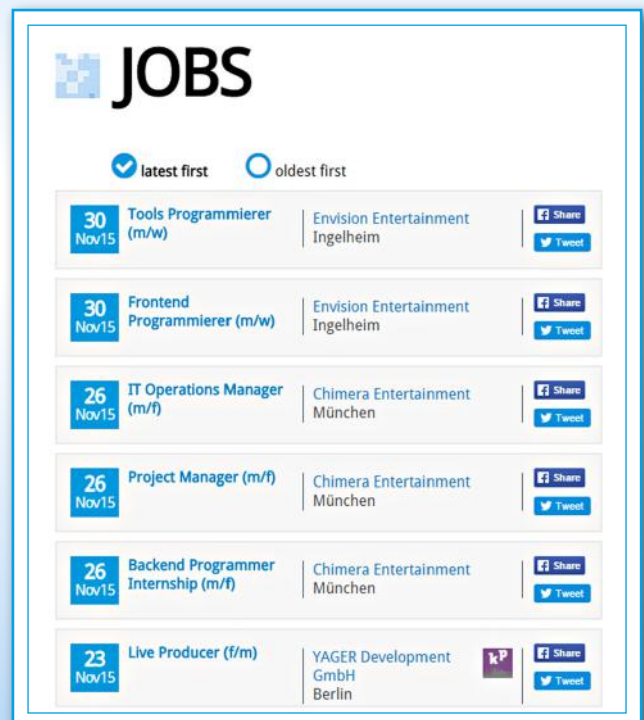
Anybody who won a steam key in all the different marketing actions had to register to Juicy Beast's newsletter, what was a good way to get in contact with new customers.



JOB EXCHANGE

Are you looking for new employees?
Do you want to hire the best talent in the industry?
Then the job exchange on makinggames.biz is the perfect match for you:

- Makinggames.biz is the **BIGGEST WEBSITE** on game development in Central Europe with the perfect target group for your jobs!
- High reach thanks to the **CROSSPROMOTION ON GAMESTAR.DE AND GAMEPRO.DE**, Germany's leading B2C-websites on videogames.
- **GREAT VALUE** – choose a single job posting for 95 Euros per month or a flatrate for 595 Euros per month, offering a subscription model with bonuses like free ads.
- A **FREE TRIAL OFFER** lets you post up to 3 jobs free of charge for a runtime of 14 days and guarantees a **RISK FREE BOOKING**.



For more information visit:
www.makinggames.biz/jobexchange



Or get in contact:
Sebastian Weber
Phone: +49 (0)89 / 244 136 650
E-Mail: sebastian.weber@webedia-group.com

underground

ACTUALLY FREE ON AMAZON UNDERGROUND BEYOND FREEMIUM AND IN-APP-PURCHASES

A new business model from Amazon, benefitting Android developers and customers alike as the players get apps for free while developers are getting paid on how long their apps are used. Carlos Lievano describes how the system works and how you can publish your apps on Amazon Underground.



Carlos Lievano
is Senior
Developer Marketing Manager
at Amazon Appstore.

Carlos Lievano is Senior Developer Marketing Manager at Amazon Appstore, engaging games and apps creators with the distribution and monetisation opportunities offered by Amazon. He is also an experienced entrepreneur, with projects including a Tech Ed business and a mobile games studio.

The Amazon Appstore allows developers to distribute and sell their Android and HTML5 web apps to millions of customers in 236 countries and territories. It is available on a multitude of devices, and is pre-loaded on Android phones on select carriers in the U.S. and Europe. And now the Amazon Appstore has over 500,000 apps globally.

Amazon has tools and services that help you monetize your apps and keep users engaged. It's easy to get started. You can test your app in under a minute and most Android apps just work on Amazon devices, with no additional development required. Developers can get started in minutes at developer.amazon.com/apps.

Amazon Underground

In August 2015, Amazon introduced the new Android shopping app Amazon Underground to customers in the U.S., UK, Germany, and France and just expanded to Spain, Italy, and 14 other European countries and territories at the beginning of December 2015. While Amazon Underground boasts all the well-known features and categories of the regular Amazon mobile shopping app, it also comes with an exciting subsection benefitting Android users: apps that are actually free and there's no premium subscription like Amazon Prime required to use Amazon Underground. More than 17,000 EUR in paid apps, games and in-app

items were available to customers on Amazon Underground at the time of the December expansion to additional European countries.

With Underground, Amazon wants to provide their customers with an experience to shop, watch, and play in one place. In addition to millions of physical goods they can discover and purchase, unlimited hours of their favorite movies and television shows they can stream, customers can also browse through entertaining audiobooks, useful apps and addictive games – now with the additional feature of apps and games that are 100 percent free – marked as actually free. All Android customers need to do is download the App directly from Amazon at amazon.de/underground, as Google doesn't allow an external app that offers Google or apps to be included in Google Play. Fire HD and HDX users don't have to do anything – Amazon Underground is already preloaded.

Being Actually Free

Customers won't have to pay a single cent to download popular premium apps like »Office Suite Professional 8« and »Monument Valley«. What's more, they can also download popular titles that come with In-App-Purchases, like »Frozen Free Fall«, »Star Wars Rebels: Recon Missions«, »Angry Birds Slingshot Stella«, and many more – except they won't be expected to pay for any of the items needed to get ahead in the game if they got the app from Amazon Underground. In times when fewer



»Colossatron« from developer Halfbrick is one of the actually free games in Amazon Underground and could see an increase of downloads and revenue since then.

users are willing to shell out for IAPs or higher priced premium apps without knowing if they will actually use it, our actually free model can be a game changer.

What does this mean to Developers?

This is made possible by introducing a brand new business model which will allow developers to turn 100 percent of their users into revenue generating customers. How? If developers agree to waive their regular download or in-app-fees, Amazon will pay them instead on a per-minute-played basis. Payments may vary in different regions. In Europe, the current rate is 0.0018 Euro. This means as soon as the users start playing the game or using the app, developers start getting paid for each minute their users spend with the Amazon Underground app. The decision whether to go premium or freemium is obsolete for developers who choose Amazon Underground. Instead of focusing on a monetisation model, they can put all their energy into creating a truly engaging mobile experience, either through an addictive game, a useful lifestyle app or kid-friendly and ad-free content. Developers no longer have to convince users to pay a premium price or wait around for them to convert to paying customers as they become fans of the game – as long as they keep the customers engaged, there is no need to sell them anything.

The fact that the selection of free apps, games, and in-app purchases on Amazon Underground have tripled since release to more than 1,500 apps by the launch in the additional European territories in December is proof that developers are adopting this new business model. Revenues are subject to increase for different reasons:

- You'll monetise all your users: If your game is free to play, instead of monetising a small percentage of users that convert via in-app purchasing, you'll get paid for every minute all of your users are playing.
- You'll increase user engagement: By taking away the frictions associated with payment walls and in-app purchasing, no user will leave your game at that point in which those mechanisms start to work against you.

- You'll get new users: If your game is premium, you're reducing the price barrier to all users that would have been interested, but aren't eager to pay upfront for their games. Moreover, the game mechanics that you had in place to influence users into spreading the word out, will become more effective, by becoming a game anyone can afford.

Real-Life Example: Halfbrick's Colossatron

The results for developers are already showing. For instance, developer Halfbrick, creator of blockbuster games like »Jetpack Joyride« and »Fruit Ninja«, with worldwide downloads of more than 500 million and 1 billion, respectively, across all platforms, has released many popular mobile games over the years using traditional monetisation models like premium, freemium, and ad-supported. According to Shaniel Deo, founder of Halfbrick, »We had varying degrees of success. [Amazon Underground] really piqued my interest when I heard about it. This is something that I think is innovative, and allows us to reach a different audience, while at the same time generate revenue from that audience.«

»Colossatron« was the first game Halfbrick published to Amazon Underground, and Deo



Amazon automatically adds Amazon Underground branding to any assets uploaded through the dev portal so you don't have to do this manually



This figure shows how downloads and revenue changed after Halfbrick published »Colossatron« on Amazon Underground.

has been pleased with the customer response. »Session lengths went up, and the overall hours that we saw in the Underground version (compared to the regular version of the game) were incredible – hundreds of percent higher. It was really amazing to see. All of that translates into more hours played in total, and ultimately, revenue.« In fact, monthly downloads nearly tripled after the game became available through Amazon Underground, and total revenue rose by more than 750 percent.

Deo believes one of the keys to the success of Colossatron makes it well-suited to Amazon Underground. »It's one of those games that has a real »stickiness« about it; people engage with it and play it for a long time.« Still, even Deo was surprised by the bump in revenue, though he's not complaining. »We wouldn't have been able to do that without this unique business model. I've been very happy with the performance.«

Now several games from Halfbrick's catalog are available on Amazon Underground, and Deo believes there are opportunities to get even more from the platform. »There's a lot of scope to really look at the strength of the model: the fact that you can monetise your entire audience. I would definitely recommend getting your apps onto Amazon Under-

ground; it's another viable platform and it's incremental revenue.«

How to turn an App into an Amazon Underground App

Amazon Underground represents an easy opportunity to convert non-paying users into paying ones through deeper engagement and longer sessions spent in the app.

The four basic steps to convert a freemium app for Amazon Underground are:

1. Make free all your IAP items.

You have two options to execute this step. Those options are:

Option A – Re-create and re-submit to the Amazon Appstore using new SKUs for the Amazon Underground version of your app. Set each re-created item to its original IAP price; Amazon will apply strikethrough pricing to your items to show that they are now free. You will also need to update your code base to use the new SKUs. Keep in mind that testing for your app will not start, unless you have successfully submitted your IAP items before submitting your app. To create and submit the new SKUs, you need to follow these steps on the Amazon Apps & Games Developer Portal:

1. Sign in to your account on the Amazon Apps & Games Developer Portal
2. Go to the My Apps view
3. Locate your app and choose the Manage in-app items option from the app drop down menu
4. Choose the in-app item you want to create (Consumable, Entitlement, or Subscription)
5. Configure the specifics around the purchasable item
6. Specify the SKU
7. Click Save
8. Repeat steps 4 to 7 for each purchasable item you want to create

After you have created all the new SKUs, you will be ready to submit your application.

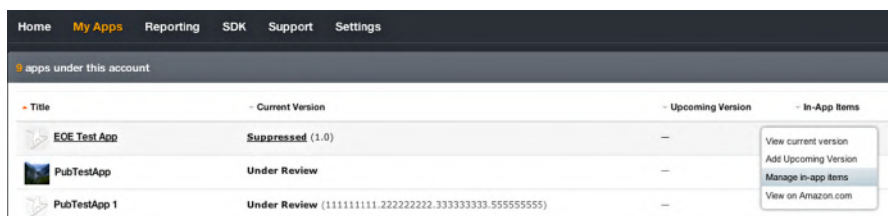
Once submitted, it is subject to the flow chart depicted in [figure 1](#). The following considerations are important regarding the different application status:

- You can cancel your app submission while your app is in SUBMITTED status by clicking the Cancel review button.
- If an issue is encountered with your app or a purchasable item defined for your app, your app status will change to PENDING, and you will be messaged with an explanation of the issue encountered and additional information as appropriate. Additional testing will be suspended until you submit an updated version of the app or purchasable items.
- Once your app has been approved for publication, the status will change to APPROVED and your app will be live in Amazon Appstore.

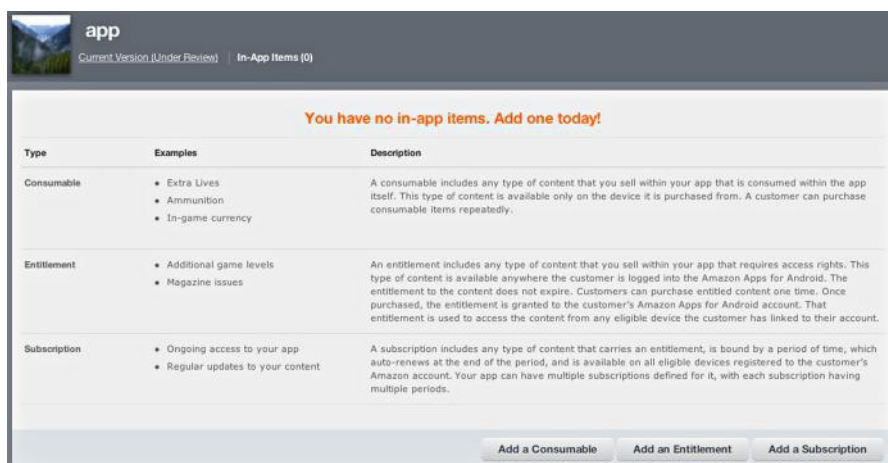
Note that if you are not using Amazon's IAP API because your app uses a different in-app



Halfbrick published several games from their portfolio on Amazon Underground.



Step 3: Managing in-app items for your app



Step 4: Choosing the in-app item you want to create.

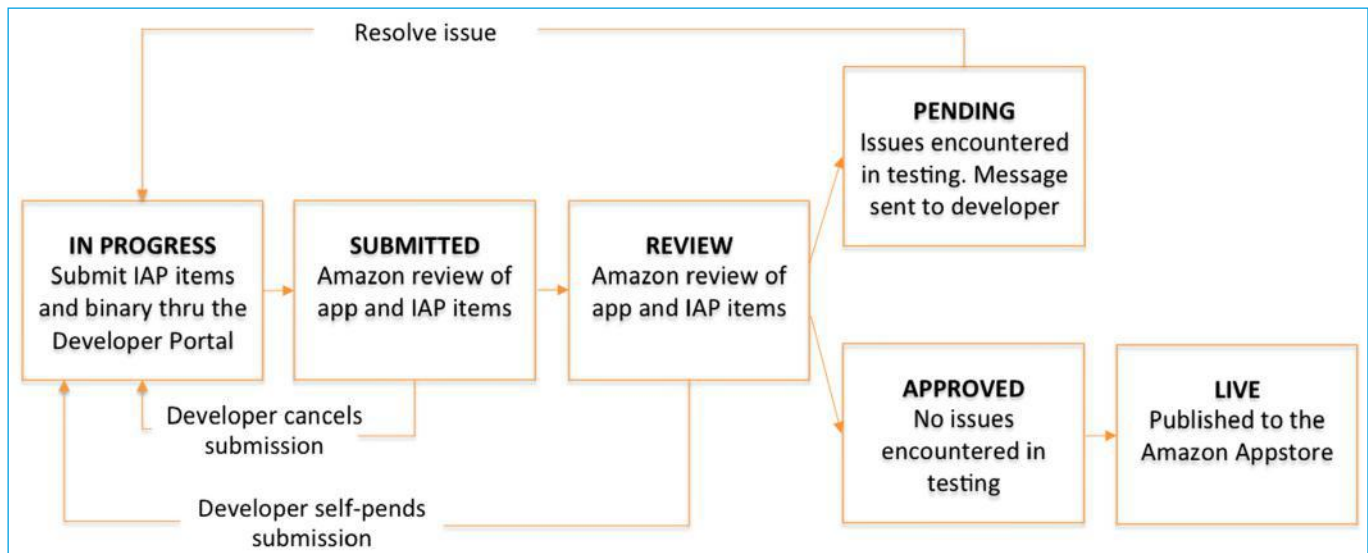


Figure 1: The App Submission Flow Chart shows the status your app will go through in the submission process.

purchasing API, such as Google's In-App Billing (IAB), you will need to implement Amazon's IAP API to use this option. This implementation will also be necessary if you want your app to be made available to run on Amazon devices, such as Fire tablets, as Google Play's IAB will not work on Amazon devices. It's an opportunity to reach new users and be discovered by all customers using Amazon Appstore.

If your app uses Google's IAB, and you wish to port your code to the Amazon IAP API for the first time, you will need to follow this steps:

Step 1: Configure your `AndroidManifest.xml` file to support Amazon's IAP. Configure your `AndroidManifest.xml` file to define the IAP Response Receiver for your app. The IAP Response Receiver ensures that the intent communication from the Amazon Client is intercepted by your app. Note that if you are supporting both Google Play's IAB and Amazon's IAP, you do not need to remove elements related to IAB; they will simply be ignored by IAP. In your `AndroidManifest.xml` file, add the appropriate `<receiver>` tag for the IAP ResponseReceiver:

```

1 <application>
2 ...
3 <receiver android:name="com.amazon.inapp.iap.ResponseReceiver">
4   <intent-filter>
5     <action android:name="com.amazon.inapp.purchasing.NOTIFY"
6     android:permission="com.amazon.inapp.purchasing.Permission.NOTIFY">
7     </action></intent-filter>
8   </receiver>
9 ...
10 </application>
  
```

Step 2: In your app, implement logic to mediate between IAB and IAP. You can use the same code base for your app, regardless of where your app is hosted. Just add logic to your app to

determine whether the app was downloaded from the Amazon Appstore or from Google Play. Depending on where the user downloaded the app, execute the appropriate methods for either IAP or IAB.

The following example code determines whether a package was installed from Google Play or the Amazon Appstore:

```

1 PackageManager pkgManager = context.getPackageManager();
2
3 String installerPackageName =
4   pkgManager.getInstallerPackageName(context.getPackageName());
5
6 if(installerPackageName.startsWith("com.amazon")) {
7   // Amazon
8 } else if ("com.android.vending".equals(installerPackageName)) {
9   // Google Play
10 }
  
```

Step 3: Add and implement the Amazon IAP API. For the most part, Amazon's IAP works similarly to Google Play's IAB. When you create the path in your code to implement the IAP API, you should be able to use a similar logic

flow to IAB, but will need to account for different class and method names.

The following table maps the most frequently used IAP methods to their IAB equivalents:

PurchasingService method	PurchasingListener callback	Response object	Equivalent IAB call
getUserData()	onUserDataResponse()	UserDataResponse	N/A
getPurchaseUpdates()	onPurchaseUpdatesResponse()	PurchaseUpdatesResponse	getPurchases()
getProductData()	onProductDataResponse()	ProductDataResponse	getSkuDetails()
purchase()	onPurchaseResponse()	PurchaseResponse	getBuyIntent()
notifyFulfillment()	N/A	N/A	consumePurchase()

Note that Amazon's `notifyFulfillment()` call and Google Play's `consumePurchase()` call correlate with each other somewhat. Calling `consumePurchase()` on a managed item for IAB will return the item's state to »unpurchased«, essentially converting it to a consumable. After that, the receipt will no longer appear in the response from subsequent calls to `getPurchases()`. Amazon will return the receipts for consumable items in calls to `getPurchaseUpdates()` until the app calls `notifyFulfillment()`.

Step 4: Test your app. To test the API integration, download and install the App Tester tool provided by Amazon in their developer portal. This tool can also test Mobile Associates APIs, if your app uses those, as well. After testing your app locally, you can use the Live App Testing service, also provided by Amazon, to beta test your app in a live production environment with a select group of users.

Option B: Remove the IAP integration from your app, but provide an alternate mechanism for users to obtain IAP items for free. For example, build a »store« that does not actually make any IAP calls or has any IAP items on the backend but whose items can be purchased through non-monetary ways. Your users could then »buy« items from your store, just as they normally would do for IAP items, but the »currency« for the store could be game experience, time played, or something else. When developing ways to unlock IAP items, keep in mind that you will not be permitted to require users to watch ads to unlock IAP items.

Implementing Option B involves the following high-level tasks:

- Remove the IAP SDK from your code.
- Modify your purchase execution call to fulfill former IAP items to your users for free: Add your own logic for what each former IAP button does, such as a »Buy« button. (This logic is handled by the IAP SDK for non-Amazon Underground apps.) Fulfill the item to the customer.

We encourage you to update the icons and strings for your IAP items for Amazon Underground to emphasize to users that they are now receiving these items for free, which will help to communicate to your users the value of the items that they are now receiving for free.

2. Repackage your existing app

In order for your game to work with Amazon Underground you will need to create a new APK for your app. This APK will need a unique package name that includes appending the

string ».underground« to the original package name for the app. For example, »com.company.example.underground« would be the package name for the Amazon Underground version of »com.company.example.«

3. Ease migration of existing customers (optional)

While not required, it would be a good user experience if you ensured that customers of your current app can keep and use any saved data in the new Amazon Underground version as well. These are some of the data and items that users generally expect to be available when migrating to a different version of your app:

- Game progress (levels achieved, high scores, rewards gained)
- Individual or character progress (a level 20 knight vs. a level 1 knight)
- In-app currency
- Competitive progress (leaderboard standing)
- Settings and preferences (colors, control layouts, defaults, sound options)

Now, we will present you with some of the tools that enable you to achieve this, as well as a suggested workflow for the migration between versions. Keep in mind that this isn't something you will only want to do to migrate a user into Underground, but also to ensure that users do not lose their progress achieved while your app was part of Amazon Underground.

There are different tools – cloud-based sync solutions – you can use to migrate the user progress across different version of your app. You may find that the best solution for your app is to use a combination of the following options:

Amazon Solutions:

- Amazon GameCircle with Whispersync for Games: Use GameCircle to share game achievements and leaderboards across the non-Underground and Amazon Underground versions of your title. Use Whispersync for Games to sync user data, including settings, preferences, and pure game state data.
- Amazon Web Services S3 and Cognito: Use Amazon Web Services (AWS) S3 and the Data Transfer API from the AWS Mobile SDK to create your own cloud solution, or use Amazon Cognito to sync user identity and mobile app data in the AWS Cloud.
- Amazon Cloud Drive: Amazon Cloud Drive is available as a paid service that can store data, similarly to DropBox or Google Drive.
- Login with Amazon: Use Login with Amazon to manage user identity support between app versions.

Identity Services: Identity services generally work in similar ways, so we will use Login with Facebook as an example: Use the Facebook SDK to provide identity support. This solution requires you create two separate Android App IDs, one for the non-Underground version, which you already have if you are using Facebook Login, and a new one for the Amazon Underground version. Each of these IDs uses the same Facebook Hash Key but will need separate package names. To locate your Facebook Hash Key, select any application from the developer console and navigate to the Binary File(s) tab. In the Appstore Certificate Hashes section both SHA-1 and MD5 hashes can be found.

Proprietary Solutions: If you have your own proprietary cloud save solution, be aware that using the package name at run-time to access data may not work for sharing data between the different Amazon versions of your app. You might need to modify your app's code to transform the Amazon Underground version of the package name to match the non-Underground version, or you may need to modify your cloud solution to accept an alternate identifier for identifying the app and version.

Android app preferences, settings, and files created by an app are stored locally in Shared Preferences, Internal Storage or SQLite databases. A new version of an app, such as an Amazon Underground version, does not have visibility to these data stores. To share this data, the non-Underground version of your app must store data in external storage – in the Android sense – so that the Amazon Underground version of your app can also access these items.

The following process is a suggested workflow for migrating locally stored data and files from a non-Underground version of your app to an Amazon Underground version. Remember to follow general security best practices when working with externally-stored files. To migrate these data and files:

1. Update the non-Underground version of your app so that each time that the state is changed in private app storage, the appropriate data is also stored in external storage.
2. In the Amazon Underground version of your app, on startup, check external storage for shared data, and import this data into the private storage of the app. If this data import is lengthy, communicate to the user that the app is importing the previous app state.
3. After importing the data into the Amazon Underground version of your app, delete the data from external storage.

Note that the techniques described here also apply to Amazon devices, such as Amazon Fire tablets.

4. Create new marketing material for the Underground version

You will need to upload new screenshots and a new app description to the developer portal.

It's important to explain the value you're giving customers and call out any additional features of your app as part of the Amazon Underground program. Also be watchful of changes that involved changes to the in-game experience shown in the screenshots, especially if one is showing the in-app store. Amazon automatically adds Amazon Underground branding to any assets uploaded through the dev portal.

5. Upload the new APK

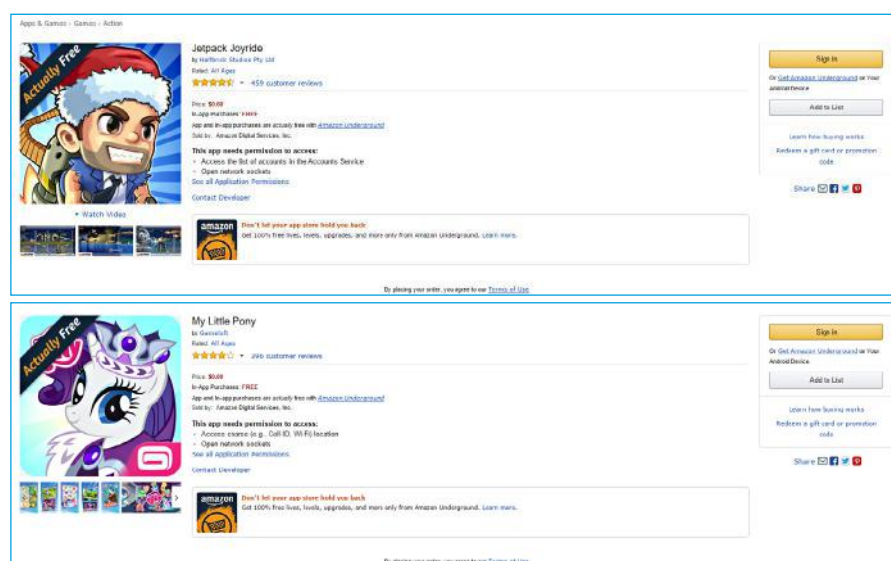
Simply upload your new APK, description, and screenshots, and submit for approval through Amazon's developer portal (developer.amazon.com/apps). Once the app passes testing, your app will be live to all Amazon Underground customers.

For any premium apps, you will be following the same steps, only skipping the first one. Getting an existing premium app converted for Amazon Underground is quite easy – developers won't even have to change the game's code to get it ready for Amazon Underground – though of course they are free to do so, as long as the Underground version is similar or better than the regular premium app.

Conclusion

With Amazon Underground, Amazon has created a one-stop-shop for all their customer's needs – from physical products to TV shows, films, audio books to apps and games. Moreover, with the new business model for apps and games, customers receive actual value – for free. A release on Amazon Underground increases customer loyalty and engagement, promotes discoverability, and supplements revenues. Preparing your Android and Fire apps for Underground is just as simple as covered here. And since Amazon's developer program is free to register, you can benefit from this new opportunity now.

Carlos Lievano



All games on Amazon Underground are discoverable on the Amazon website and are identifiable as actually free.

LOCALISING COMEDY

Comedy is one of the biggest localisation challenges. Mark Estdale explains the importance of game assets, context and the necessity of changes to avoid a weak game localisation.



Mark Estdale
is CEO
at OMUK.

Mark is the pioneer and evangelist of Game Immersive Voice Recording and specialist on the adaptation of acting for espionage. He is a voice director turned audio engineer, audio technology and tools expert. He joined the industry in 1994 as a consultant to build the first dedicated voice studio for games in the UK. Many studios and over 600 titles later Mark runs OMUK, one of the most successful international voice production teams in London and Los Angeles.



The example of »Zero Wing« for Sega Genesis shows why Japanese to English translations are infamous in the industry.

Getting comedy right in localisation requires a game asset rich approach that provides the freedom to deviate from the original language. If this is not understood and resourced then the localisation will be weak.

In his book »Is that a fish in your ear?« Professor David Bellos discusses what he calls the »myth of literal translation«. He argues that the key to translating comedy »lies NOT in preserving the literal text of the original but instead ABANDON the idea of perfect fidelity and try to find humour that rings some of the same bells as the original.« Finding what »rings bells« needs intimate knowledge of the context of the game.

Context is King

The best solution therefore is to translate whilst playing the game. But doing this takes time and there is a distinct possibility the game won't be ready when translation is required to start. Without context you will create bugs. You need to decide between the cost (including time) of preparing assets to give rich context for the start of the localisation process and fixing localisation bugs in testing. There is a clear statistical correlation between quality of contextual information and the number of bugs.

If the game isn't ready, the next best thing is to use raw game assets for context: character

art, object art, scene art; and any audio assets: recorded voice, music, and sound effects. It's also a good idea to write contextual notes – especially when wordplay is involved.

Localising Verbal Humour

Localising verbal humour is difficult, and in some instances, impossible. The humour in puns or wordplay comes mainly from exploiting the multiple meanings of similar sounding words. As these similarities frequently don't exist between languages, it is important to remember that »perfect fidelity« isn't the aim, capturing the essence of a moment is. This is the core difference between pure translation and localisation. Translation focusses on the words and localisation on the objective and momentum.

A developer who has worked on a game for two years will know the context intimately, a translator coming in fresh won't. The more context you can give the better. Also having a short and fast chain of communication between the translators and the developer is a big advantage for dealing with questions effectively.

Changes are necessary

Making changes to enable a scene to work in a different language needs to be understood by the whole team. Humour is full of cultural references so adapting the scene to make the

humour work in a different market is an essential ingredient in localisation requiring trust between the creative team and the localiser.

Notorious in the industry are Japanese to English localisations. The most notable reason for cringe-worthy localisations is perhaps the fact that historically Japanese publishers and developers have retained the last word of localisation veto. They often over-control the creative input of the localisation partner, consequently constraining and losing out on the creative cultural input of the localiser. Freedom to localise is freedom to adapt. Constrain it at your peril.

Escape the Literal

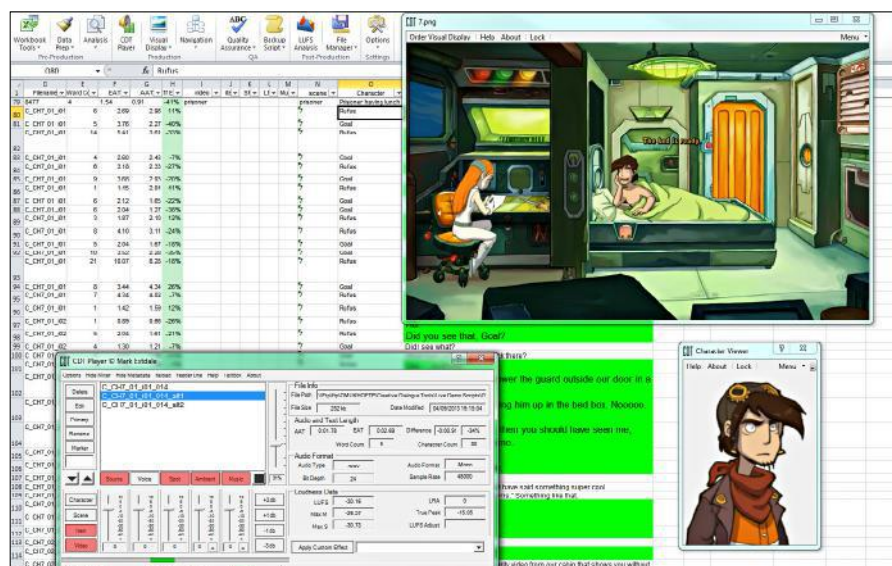
When recording the English version of »The Book of Unwritten Tales 2«, a translation bug was flagged by the developer in a response to the character Wilbur making a comment »It doesn't look like ice«. In German the response to Wilbur was »Weil es hier unten völlig windstill ist! Ohne Wellen und Blasen im Eis sieht gefrorenes Wasser aus wie ... Wasser.« The localisation in English was »So?« This is a translation bug but NOT a localisation bug. If the German was translated literally the scene would not have worked as well in English. Focussing on the intention, and NOT the literal meaning, is about preserving the momentum and tension in a scene above all else. The change would not have been possible without knowing the context of the scene. The context brings meaning to the moment and makes it possible to escape the literal.

Scene: A little ladder stands next to a large step-ladder and says to the step-ladder: »you're not my real father.« This is a wordplay in English that ties step-ladder with step-father. How would you make it work in German? The choice is to find a different joke that works or to remove the joke. A literal translation would not work.

Daedelic's »Deponia« series of games written by Jan Müller-Michaelis are full of German language puns and wordplays that sadly don't translate between German and English. So if you play the English versions you will find either completely different puns or no pun at all. I was very lucky to have Jan join me in the studio for recording the English versions of the games. His input was priceless as he could explain his sometimes obscure humorous intent. We had time to experiment with the English. The whole studio team including the actors joined in the process. The collaborative effort was extremely valuable. Without knowledge of the context and intent it would have been impossible to get the results we did.

The Monkey Wrench

A while ago I was speaking with a non-English adventure game developer about some really odd puzzles he had in his game. Like most adventure gamers he was influenced by LucasArts.



For the localisation of Daedalic's »Deponia« series, writer Jan Müller-Michaelis gave valuable input.

Whilst defending his puzzles he referred to »Monkey Island 2«. He said »If you think my puzzles are weird, what about Guybrush having to have sex with a monkey to stop a waterfall.« At the time I had no idea what he was talking about so I checked it out.

The funny thing is he thought the puzzle really cool even though he completely missed the simple American wordplay. If you're unfamiliar with the game, Guybrush needs to close a valve to turn off the waterfall. In Britain a spanner does the job, but in the US it's a Monkey Wrench, hence the monkey. A classic wordplay. While the intentional humour in the writing had been missed, it didn't really matter that much as the playing experience wasn't lost. The original joke was completely lost in translation but it didn't matter for the gamer. The essential thing is to keep the momentum of the playing experience.

I spoke to some of the original Monkey Island team about the monkey animation and they admitted that the sex animation was done that way on purpose. It just happened to slip by the censor.

Fidelity to the Intention

The keys to successfully localising comedy lie in putting fidelity to the intention of a scene above fidelity to the words that create it. Attention to detail will help you faithfully capture the essence and intention of a scene. As developers, share as much as you can as this helps everyone understand the context and make the writing and performance work. This may mean adapting your pipelines but the impact will be significant. It is context that makes it possible to escape from a literal translation. Without it you have only the words to work from. Finally, it's worth noting that this holds true for all aspects of localisation and voice recording, not just localising comedy.

Mark Estdale



Knowing the context of each scene in »The Book of Unwritten Tales 2« was extremely important for the localisation process.



The original joke of the monkey wrench in »Monkey Island 2« was lost in translation.

7 WAYS TO RETAIN PLAYERS

No matter how good a game is – if it fails to captivate the user, it fails to reach its essential goal. Gunther Rehfeld, professor at the HAW Hamburg (Hamburg University of Applied Sciences), provides tips on how players can be motivated to stay on board.



Alexander Karl
ist Redakteur
für nextMedia-Hamburg.

Alexander Karl, born in 1989, writes for nextMedia.Hamburg, a joint initiative for the media and digital industry, based in Hamburg and carried by the Free and Hanseatic City of Hamburg, the Hamburg Business Development Corporation, the Hamburg@work association as well as committed companies. The initiative's goal is to secure Hamburg's top position as a media capital and to improve the general conditions for local companies. One of the initiative's projects is gamecity:Hamburg, the biggest regional network of the German games industry with over 2,000 players.

It's every game's ultimate goal: to captivate the users and to attract their attention for as long as possible. It's not only computer games that have this goal: In ancient Greece, they were already wondering how they could entertain people with their texts and stories. Although things have changed quite a bit since then, especially in terms of technology, the mechanics of the attention economy still bear striking similarities. Hamburg Professor Gunther Rehfeld knows which elements it takes for video games to capture their players' attention. For nextMedia.Hamburg, the joint initiative for the media and digital industry, he commented on seven ways on how to maintain a user's fascination by a game.

1. The story

»What's it about?« is the typical question you get when telling your friends about a new game. Often, the more complex a title is, the more difficult it is to answer this question. However, there is a broad spread of opinions about which elements make a good story. One of the most popular schemes for constructing a

story is the hero's journey which the mythology expert Joseph Campbell created from a multitude of old myths. The hero's journey involves twelve stations and starts with a task for the hero; it takes him on an adventure, and provided with new knowledge, he is finally taken back to his old world.

»The principle can be found especially in narrative action RPGs« explains Gunther Rehfeld. »One example are the ›BioShock‹ games by 2K, where the protagonist has to face a number of tasks. But the hero's journey also applies to a lot of other titles since Campbell defined his stations of a hero's journey in wide terms.« With regard to cinematic scripts, Christopher Vogler wrote a handbook about »The Writer's Journey«, which also offers a guidance worth reading for game developers about the process and structure of a hero's journey as well as a description of the core archetypes. »I would also recommend learning about the Asian narrative structures though as they appear in the Chinese story ›Journey to the West‹ where the concept of the hero is influenced by Buddhism and Taoism.« Those who want to reach an international audience, should therefore think outside the Western box.

2. The characters

Speaking of hero: Every game features one or more protagonists, unless we're talking about »Tetris«. But the bigger, more complex productions put their focus on one or more characters who lead through the story. The past millennia saw the rise of certain types which Carl Gustav Jung summarized as archetypes. »The spectrum ranges from the classic hero to the shadow which is either the villain or the inner discontent. Thus, archetypes vary in terms of their character traits on the one hand and their function for the plot on the other hand. Often, the hero is the character leading through the game while the shadow gets in his way as the villain. However, there isn't just good and bad; there are also ambivalent characters. Even an



The characters in a game can be an important aspect for the user to engage in the game on a long-term basis. They don't necessarily have to be classified as good and evil; anti-heroes as in »GTA 5« can have a certain appeal, too.



The quests a player has to complete don't just have the goal to keep the player occupied; they may also have a captivating effect. A good example is »City of Heroes« where players can even develop their own tasks based on the core quest mechanics.



Some games focus on mysteries in the background of the actual plot, which help the player to immerse himself even deeper in the story - like in »Lost Ember« where the player is to discover the mysteries of a forgotten culture by assuming the forms of various animals, among other things.

anti-hero like Rufus in »Deponia« can win some sympathies« explains Professor Rehfeld. »The criminals in »GTA«, too, are typical anti-heroes which the users don't mind embodying from time to time.«

3. The game objective

»The players should always know exactly where they are and where they want to go. Where am I? Where did I come from? Where do I want to go at all? These are the central questions that always need to be answered« says Rehfeld. However, that doesn't mean that there can't be any puzzles; provided there is clarity again eventually, for which it takes good UI design and a clear structure of the game elements. One possibility is a symbolic charge of elements such as swords with a narrative background – comparable to the shield of Achilles in Homer's Iliad. The motivation of the player is decisive especially in narrative titles, and just like in real life one is driven by goals of which there usually are a few in games to appeal to the diversified audience. »That goes for MMOs, but also for »Clash of Clans«. Seemingly pointless arranging of blocks in casual games in order to beat one's personal highscore can be a game target, too, though.«

4. The quests

There is no rule of thumb for good quests, however, there are different varieties, from the fetch quest to the story quest to the side

quest, all of which offer different possibilities to provide tasks. »Fetch quests especially where you have to collect things like spiders or rats are easy to create; however, in most cases they are of little narrative relevance. On the other hand, story quests like getting a car from A to B in »GTA« define a clear goal. The side quests embedded in the regional quests of »Guild Wars 2« were nicely implemented« says Rehfeld. An interesting approach, be it on a kind of meta-level, was offered by »City of Heroes«. »The quest generator was integrated into the game« explains Professor Rehfeld. Based on the core quest mechanics, the players of »City of Heroes« were able to develop their own quests and evaluate those of other players. And yet, most of the quest texts aren't read by the players, but simply clicked away since the really important thing is the actual task and the resulting reward.

5. The reward

»The reward plays a vital role in the game for the player to progress. In the game this can be the level progression or the beating of a highscore.« But even rewards like new swords for the hero can be an important driving force. »Blizzard Entertainment, for example, offers creative reward systems in their games, as do Hamburg-based browser game developers Goodgame Studios or InnoGames who smartly apply the highscore concept« adds Rehfeld. There is an unlimited number of rewards

 nextMedia.
Hamburg

nextMedia.Hamburg is a joint initiative for the media and digital industry, based in Hamburg, carried by the Free and Hanseatic City of Hamburg, the Hamburg Business Development Corporation, the Hamburg@work association as well as committed companies. The initiative's goal is to secure Hamburg's top position as a media capital and to improve the general conditions for local companies. One of the initiative's projects is game-city:Hamburg, the biggest regional network of the German games industry with over 2,000 players.

 next
MEDIA.HAMBURG



An active community can also be essential for the player retention. In case of »World of Warcraft« Blizzard even recruited some employees from their community.



Gunther Rehfeld
Professor of Graphics
and Image Processing
at HAW Hamburg

Gunther Rehfeld, born in 1961, studied Arts at the HfBK Hamburg (Hamburg University of Fine Arts). After his graduation he started out as a freelancer with a focus on new media. Following a professorship for Media Communication and Media Design at the FH Kaiserslautern (Kaiserslautern University of Applied Sciences), he moved to Hamburg in 2001 where he's been teaching as a Professor of Graphics and Digital Image Processing at HAW Hamburg to this day. His main focus is on identity formation through in-game narration and through the conception of living environments. In 2007, Gamecitylab was established in cooperation with the IT Faculty, followed by the partial MA program »Games« in 2010, initially funded by gamecity:Hamburg. Additionally, he teaches in the Games Master program (www.gamesmaster-hamburg.de). His book »Game Design und Produktion: Grundlagen, Anwendungen und Beispiele« (Game Design and Production: Basics, Applications, and Examples) was published by Carl Hanser Verlag in 2013.

which appeal to different types of users, like finding items, reaching self-appointed goals etc., which is then again enhanced by gaining items, gold, social acceptance, glory, honor, achievements, better rankings, praise by fellow players, improving one's own skills etc. Biologically speaking, the release of dopamine is of great importance, among other things, when the player reaches certain of these milestones in the game.

6. The mysteries

Some games focus on mysteries in the background of the actual plot, which help the player to immerse himself even deeper in the story – the Hamburg-based start-up Mooneye Studios is currently taking this approach in »Lost Ember«. In this game, the player is to discover the mysteries of a forgotten culture by assuming the forms of various animals, among other things. And those who have played »Age of Mythology« may wonder what Zeus, Odin or Isis were all about. However, Gunther Rehfeld thinks it's quite unlikely that players suddenly start reading collections of ancient myths by Gustav Schwab. »The media shift to books is rather unlikely, but with a bit of luck we may achieve a shift to Wikipedia.« However, those

who plan to develop a game with lots of references to the classic antiquity or the Middle Ages don't have to worry anyway: »It can work very well if the myths are used appropriately. Even if I don't think that all players will find it interesting.« It's therefore rather the icing on the cake if users get lost in those background mysteries, provided they already took a liking to the story in the first place.

7. The community

»The integration of the community is very important for the development and marketing of the games« explains Rehfeld. This worked out particularly well for »World of Warcraft«. »They took community leaders and popular gamers on board to develop the MMO. That way, they were able to pinpoint their audience even more. But even when releasing an unfinished version – clue: Steam Greenlight – it's important to listen very carefully to what the community has to say.« In that case, all communication channels should be considered, from social media to relevant forums. Obviously, it's essential to filter the input and check it for relevance. But the tie to the customer shouldn't be severed after the release either: »The more you accommodate and care for the customers' wishes and problems, the better you can develop your product. Those who are deeply involved in the game, will promote it – or may just as well harm the game's image.« This applies to both the real-world peer groups and social networks.

All these factors lead to what is the core element of every game: fun. »But whatever we think is fun, is different for every user and with every game principle« states Rehfeld.

Gunther Rehfeld



References to the classic antiquity like in »Age of Mythology« or to the Middle Ages may fascinate the players due to the real background and cause them to engage deeper in the matter.

OPEN WORLD ON MOBILE WITH UNITY

TruckSimulation 16 is a truck simulation for mobile devices in which players can roam around in a massive open game world. Christoph Eder of developer kunst-stoff explains how his team met this technical challenge with Unity and which problems they had to solve along the way.



For a truck simulation like »TruckSimulation 16« the gameplay is clearly defined: The player will be driving. For that reason the surroundings should be appealing and not be repetitive. The defined playground is the motorway network of Central Europe. Every street was to offer a certain level of reality in terms of course, landscaping and distinctive landmarks. Such a big, non-procedural game world requires considerably more space than the memory of a smartphone has to offer. That's why it is divided into smaller bits that are dynamically loaded as required during the game.

Dynamic loading

In an ideal case, these landscape patches are completely created in a 3D modeling program. That way you have full control of the degree of detail and the visibility of the various areas and you can, for example, create tidy break-off edges at road ditches. Unfortunately, this approach is too time-consuming for the size of our world and thus too costly. One should also

keep in mind that the Level Designers have to be familiar with a 3D modeling program.

Despite justified performance warnings we decided for the terrain that's integrated in Unity. These terrains are defined through a heightmap and can be quickly created, amended and automated through scripts in the editor. Besides, the render quality of geometry (pixelerror) and textures (basedistance) can be easily adjusted to the performance of the device. The compromise between quality and memory usage meant a resolution of one data point per unit and meter.

The world with its landscape, houses, streets etc. was divided into square tiles sized 1024 x 1024 meters (image 1). A square kilometer is big enough to comfortably accommodate a huge city or a motorway junction, and not too small to make life hard for the Level Designers. In hindsight though slightly smaller tiles would have been better to reduce the memory usage and the load times. In an open game world it's common to load and display three or eight adjacent tiles. After running a few tests



Christoph Eder
Technical Director
at kunst-stoff GmbH

Chris has been working at kunst-stoff since 2011 (with interruptions) and was the technical lead for TruckSimulation 16. Being a big fan of racing games, he especially enjoys driving race trucks. After his studies of computer science with a focus on computer graphics, he moved to Berlin where he's been working at local game developers for eleven years. He can look back on a track record of 25 game titles. www.xing.com/profile/Christoph_Eder2



TruckSimulation 16 includes an open game world based on Central and Western Europe with over 20 cities. To implement such a huge territory on mobile devices with Unity wasn't exactly an easy thing to do.

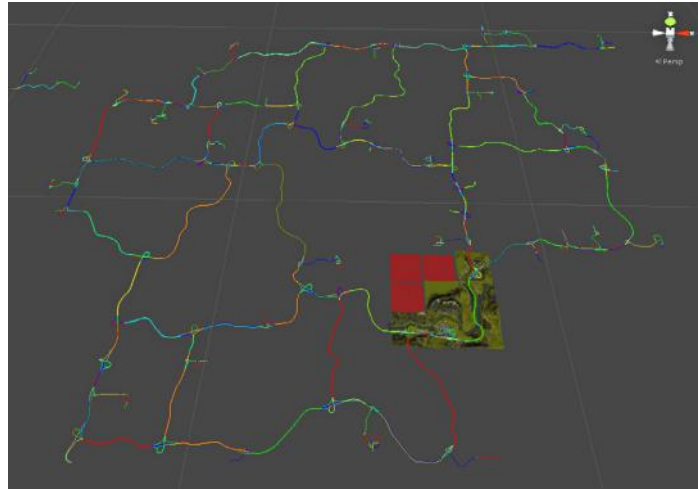
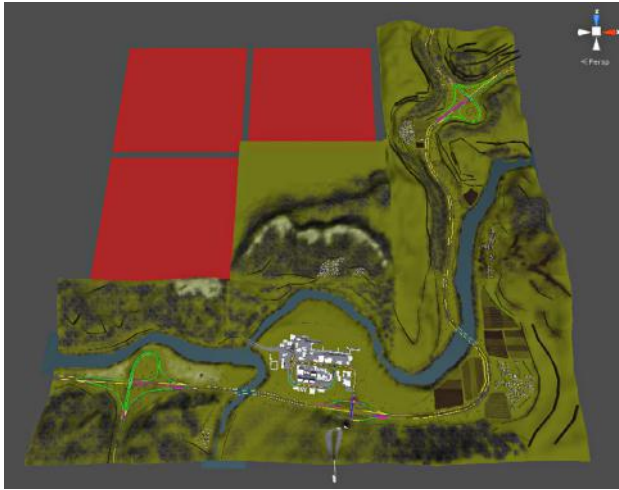


Image 1: The terrain of several landscape scenes can be edited at the same time.

though, we confined ourselves to only one adjacent tile. This is possible since we can limit the camera position through our road course. The player won't notice it in practice, and it saves a lot of effort and memory. We set the visibility range in the highest visual quality to 600 meters, which is quite remarkable for a mobile game.

To drive through a landscape tile the player only needs about 20 seconds using the 200 km/h (124 mph) racetruck. A mobile device of a lower performance class (like the iPhone 4s or Samsung S3) can only just manage to load and display the next tile in the background in time. If there is a massive loss of performance due to other apps, the truck would drop through the map. In order to not interrupt the gameflow, tiles must be loaded in the background – asynchronously. Unity supports this by saving each tile in a scene of its own. In each of these scenes there is only one root node to easily unload it. Alternatively, we could also work with a pool of instantiated objects and a separate load system. Unfortunately, when setting a heightmap of a terrain, a Unity application always stops for a few milliseconds. That's why slower devices experience choppy playback from time to time when loading scenes. And we forgot to factor in »Mr. Physics«. Most scenes include hundreds, if not thousands of static colliders for guardrails, street lanterns, houses etc. The internal integration into the room partitioning procedure of PhysX takes roughly about several milliseconds.

The landscape

Unfortunately, it's slightly cumbersome to edit the terrain; fortunately, smaller tools from the Asset Store can help here. We tried out various terrain generators like »Terrain Composer«, but none of them were convincing. The floor consists of a mixture of small, repeatable textures (splat maps). The use of satellite photos is technically feasible and

was also tested, however, it would take a lot of memory, even more media storage and way too much time to load.

In the course of the project we discovered several flaws of Unity Terrain:

- The virtual camera in Unity Editor has a fixed setting based on which the terrain renders artifacts which make an exact placement of objects more difficult.
- The built-in support of trees isn't usable on mobile devices. If such a tree is placed on a terrain, the frame rate on the device is halved when rotating the camera.
- Only a few floor textures per tile can be used. When there are more than three, the performance drops.
- On some Android devices with Mali chips the floor texture is rendered incorrectly as a rough chessboard pattern. The workaround is to only use odd splat map indices. We moved the grass texture that's used practically everywhere to second position.
- The terrain offers a performant collision detection which prevents the truck from dropping through the ground. In rare cases, this became only active after a few frames on mobile devices, so that the truck dropped down nonetheless.

As a test, the finished Unity Terrain was transformed to static geometry in a complex automated process (Image 2). The number of polygons was considerably reduced using the mesh simplification tool »Cruncher«; for optimization purposes, we tried to leave the triangles close to the street untouched and removed any triangles below streets. Altogether, the rendering of this approach was more efficient, but it also increased the compressed size of the app from approximately 2 MB per tile to 3 MB. In an uncompressed state, this corresponds to 5 MB and 7.5 MB. With further development work, one could thin the polygon mass through occlusion culling in a more targeted manner in order to make this approach the preferred

kunst-stoff
GmbH

kunst-stoff is an independent game developer based in Berlin. The team creates titles for PC and Mac as well as for consoles and mobile platforms. Apart from »TruckSimulation 16«, they have released titles like »Lapuco«, »Galaxy Racers« and the »The Great Jitters« series.

The studio also implements interactive exhibits and apps for customers from the industry and culture sectors, for example, the educational game »Isungur – Save Your Viking Village«, the museum app »Vikings – The Treasure Quiz« or two interactive installations for Volkswagen's Autostadt.

www.kunst-stoff.de



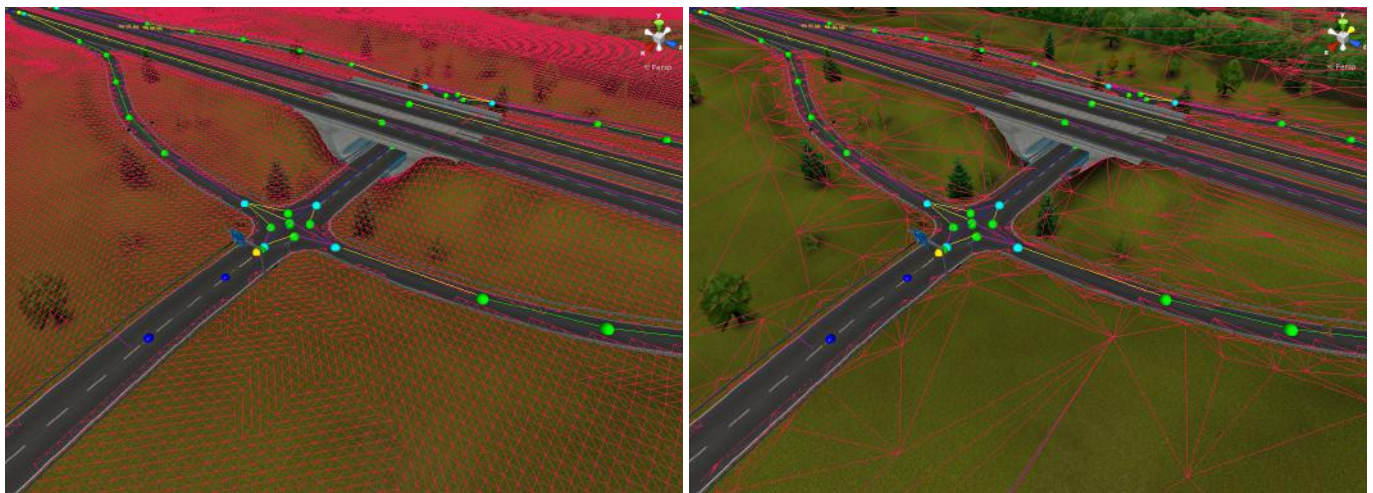


Image 2: The Unity Terrain as a heightmap can be automatically converted to a polygon-reduced mesh.

procedure. Unfortunately, the Cruncher API doesn't have enough impact on the process. In the distance, Unity displays its Terrain with increasingly less triangles (dynamic level of detail, ROAM), which may result in penetration flaws with the street. That's not the case with a static mesh.

For tunnel entrances, no visual or physical holes can be drilled into the Unity Terrain, even if some assets from the store say so. In order to compensate for that, we drew the Unity Terrain in the tunnel below the street and replaced the area above the street with the terrain converted to the mesh. Meshes in Unity can comfortably be edited with plugins like »Mesh Toolkit«.

The street geometry lies 4 cm above the terrain. It's the lowest value where no penetrations and no Z-buffer fighting occur on the mobile device. It turned out that the adjustment of the terrain to the street takes a long time and could not be performed manually with the same precision. A lot of effort was put into developing a tool which ensures a nice adjustment of the roadside as well (image 5).

Terrains in general take up a big part of the computing power on mobile devices and therefore have to be considered thoroughly.

Automate

In a six-minute driving mission with an average speed of 100 km/h the player covers ten tiles. As a long-term motivation we created a road network with 1,300 streets on 180 tiles, which took several man-years to do! I would like to take this opportunity to thank the Level Designers for courageously hanging in there during this almost endless task. A manual correction, for example exchanging a certain type of tree, would be prone to errors and would take days if not weeks! That's why scripts are required to work through all tiles. One example is the above-mentioned conversion of the Unity Terrain into static geometry. A set internal

structure of the tile scenes is always helpful for scripts. By the way, only a few dozen scenes can be loaded into the editor at a time because the maximum of approximately 64,000 physics objects in Unity 4 is rapidly reached.

Unfortunately, Unity 4 can only edit one terrain at a time. A tool has been written for comfortably and simultaneously displaying and working with several tiles (image 1 and 3). The common boundary of two terrains was smoothed semi-automatically (stitching). When saving tiles a lot of integration tests are run, for example, to check if streets are connected correctly. It's important to detect potential errors immediately. To completely drive and test all routes in both directions takes at least one day. Unity 5 now supports multiscene-editing.

Light and shadow

A real-time shadow requires considerable performance from mobile devices. In terms of a driving simulation, the limited depictive space of the shadows can be noticed, for example, if in a parkway only the next two trees cast shadows, but not the other ones that are further away. Lightmaps for the terrains and all objects look great, but they require creation effort, computing power and memory. These two known shading techniques prevent Draw Call Batching (see below). That's why we used some tricks here.

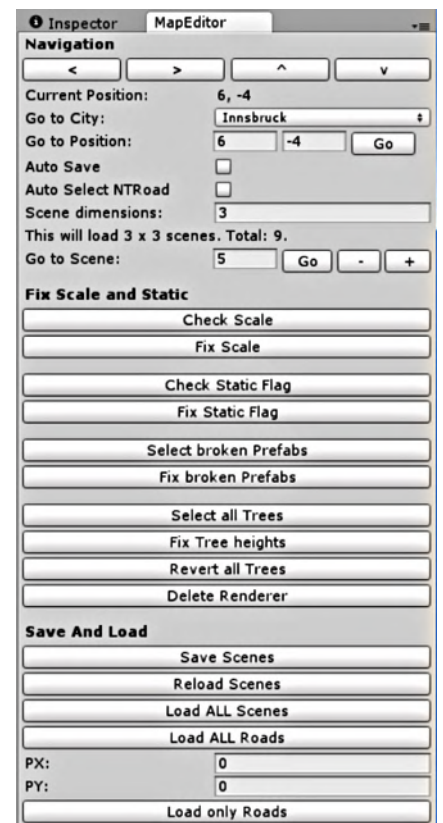
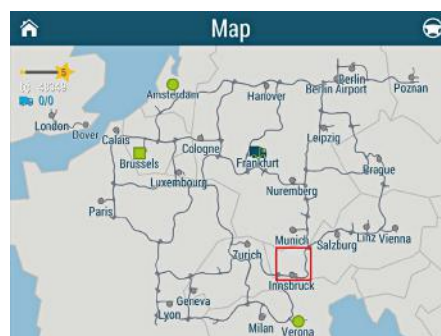


Image 3: The landscape scenes are drawn into the vectorized map of the game (left) and the internal Tool Map Editor offers comfortable management of all landscape scenes (above).



Image 4: Even without tiling, a lot of different houses can be textured by only one texture.

In the country, the terrain under each tree was darkened. A script or a Level Designer increased the weighting for the darkest floor texture around this spot. The measurement is free of charge; it requires no extra service. The result is similar to the simple blobs under the vehicles. A lot of houses and other objects in the countryside are often far away and require no shadows.

For the »City« scenario, a shadow through interpolation of the floor textures (blending) or multiplication with a factor per geometry node (vertex coloring) of the terrain isn't sufficient. The one-meter resolution of the terrain limits the resolution of the shadow. The terrain is covered by the street, and bridges and houses etc. also cast shadows onto the street. For each city a lightmap was calculated and placed on a big transparent layer in the middle of the city (image 5). Even details like lanterns cast a correct shadow onto the floor. They don't leave shadows on walls, but this situation doesn't occur anyway. The lightmap is like an IGeL service and involves extra costs, e. g. one frame on slow mobile devices. One restriction is that the sun can't be too low and can't move around, and cities must largely be built on one layer. All this won't really be noticed by the player. In terms of the terrain version with static geometry, we calculated the shading (not shadows) offline and wrote them into the geometry nodes in order to relieve the actual terrain shader.

One way to implement shadows, which is useful for barren landscapes but rarely applied, is the use of pre-calculated shadow flags. Since objects can be placed facing any given direction, the shadow of e.g. a house can be pre-calculated in eight directions and

projected on transparent geometry. In many cases, a number of generic shadow forms is enough, and through rotation and scaling of the geometry the shadow can be fine-tuned. However, the shadow flag approach already failed due to the creation effort, selection and arrangement and especially due to the positioning in regards to the mountains. Altogether we're happy with the shadow, even if our approach doesn't take special cases like tunnels into account.

Streets and traffic

There have been some assets and software for the simulation of car traffic; however, none of them were designed for a massive game world and dynamic reloading. Programming dynamic and credible car traffic took several months and should not be underestimated! The implementation of the road network and the car AI was the subject of numerous discussions and meetings.

Streets are defined through control points which – in the correct order – form a curved line or space curve (spline) and thus constitute the center line of the street. This logic information later becomes important for the car traffic, the map and the path finding. For each individual lane a space curve is derived with a distance of a few meters (image 5).

The original idea to create geometry and crash barriers via control points whilst driving soon turned out to be impracticable. The integration of the »EasyRoad« asset for the static creation of streets and their geometry proved to be a catastrophe only after production had already started. We fixed bugs in hard-to-edit source code (obfuscated code) and had to live with bugs in the libraries (dll).

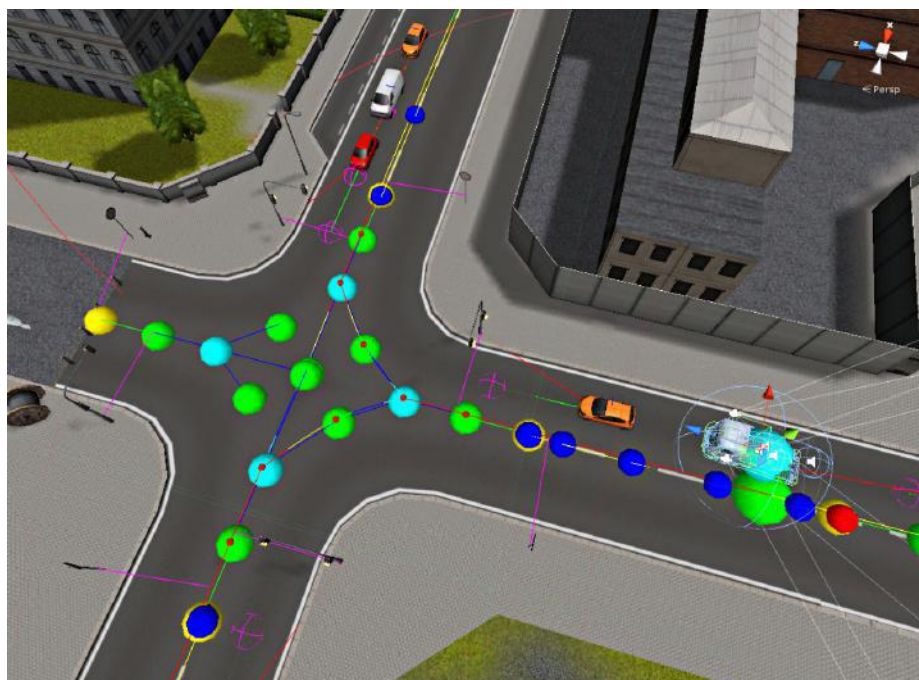


Image 5: An intersection with traffic lights and car AI.

In hindsight we would write the required functions for the street creation, the adjustment of the terrain, the creation of crash barriers, noise barriers, lanterns etc. ourselves. Due to their number of polygons and colliders, the crash barriers proved to be a real performance killer, especially in curves. Another idea that was tested and rejected was to replace the many colliders by an invisible strap that would pull the truck to the center of the street. The result were very funny physical effects which, however, didn't do justice to the term »simulation«.

Intersections and motorway exits were built using a 3D modeling tool. In order to avoid having to form the terrain for bridges in a way that it fits exactly the streets, the terrain surrounding the bridge was created as a geometry. To place it, a script would press the Unity Terrain into the form from below, just like making a sand cake, and the form would then be removed.

The first implementation of the car AI was physically realistic, but it overstrained the mobile processor. Even when a car slipped into the ditch, it often managed to make it back onto the street. For the second implementation, the cars were pulled by an imaginary string, but it failed due to one little detail: In each frame for every vehicle the position is calculated on a parameterized space curve (spline interpolation), which considerably strains the CPU. We therefore had to limit the number of cars to 25. A linear interpolation between pre-calculated close control points on the curve would be an improvement. The constant assigning of the road section that's closest to the truck strains the processor, too. It is required both for the cars to brake in time and for the path finding. Algorithms from scientific articles were analyzed and programmed for the optimal calculation of the shortest distance of a point to several space curves.

What's really disadvantageous is the often demanded option to nest prefabs in Unity. With regard to the »Intersection« prefab, one would love to set up the traffic lights as well as the actual lights as prefabs. In my opinion, that's the only real disadvantage of Unity Engine.

Performance

A tile has 500 to 5,000 objects on average. The system for display accuracy depending on the camera distance (level of detail) included in Unity 4 is too slow in this case. Objects that were close to each other were logically grouped in a particular system and switched on or off together in regular intervals, depending on the camera distance. A minor disadvantage is that the objects' pop-up distance slightly varies. Unity 5 has improved this system.

The multitude of game objects has a negative impact on the performance, especially in cities.

Unity can pre-calculate object occlusions for different camera locations offline (occlusion culling), which almost doubles the frame rate in cities! Much to our regret, this function can't be applied to asynchronously loaded scenes. The Asset Store offers several systems for occlusion culling, which, regrettably, aren't sufficient for our purpose.

The models and textures were created with foresight. All houses in the country, in industrial areas and in the cities only use a relatively small texture atlas (image 4). This allows for fast rendering and keeping the memory usage low. Some texture areas are used by various objects. Depending on the area the player is currently in, ideally only one to two of these atlases are in the cache at any given time. In addition, there is an atlas for all traffic signs. A high-rise could easily be coated with small tileable textures. Such textures were only exceptionally used for crash barriers and in order to keep the number of textures to a minimum. For performance reasons, native or complex shaders were only used for vehicles and water surfaces.

For performance optimization reasons, Unity can combine the geometry of several objects in one big object (vertex buffer combination) offline (draw call batching), distinguishing between two variants: The first one where objects aren't moved during runtime or amended in visibility status (static batching), and the second one where this doesn't apply (dynamic batching). The first one is recommended for the largely unchangeable objects of a landscape, however, only under certain conditions by Unity: All objects in a batch must use the same material and mustn't have more than 300 vertices. The graphic designers weren't quite fond of this threshold; yet it's important. If necessary, a complex object must be divided into two small objects. Transparent objects practically aren't batched and should generally be used sparingly on mobile devices. Further conditions are stated in the documentation. Activated »static batching« increases the frame rate significantly. One step further can be taken when combining the geometry offline. This would increase the frame rate even more, but counters the selection of objects located outside the camera (frustrum culling) and interferes with the mentioned adjustment to the camera distance and device performance. A performance factor of 4 lies between the slowest and the fastest supported smartphone. That's why we offer the player three performance settings.

TruckSimulation 16 is not only just a driving simulation. What wasn't addressed is that the players can build their own logistics company by establishing branches and hiring drivers.

In conclusion it can be said that open game worlds on mobile devices are still an adventure and increase the development time significantly.

Christoph Ender

Truck-Simulation 16

In TruckSimulation 16, the player takes on the role of a logistics operator. Players can do deliveries themselves or later, in a management part, hire employees who also do deliveries. The game includes seven trucks and major parts of Central and Western Europe with over 20 cities.

- **Platforms:** iOS and Android
- **First publication:** November 25, 2015
- **Publisher:** Astragon
- **Facebook:** /TruckSimulationMobile
- **Website:** www.trucksimulation16.de



BODY PARTS AND MOTION SICKNESS IN VR

MITIGATING MOTION SICKNESS WITH LEGS

A lot of VR users complain about motion sickness and similar discomforts. In a scientific study Dr. Matthew White tried to find out if missing body parts in games might be the reason for those troubles.



Dr. Matthew White
is Data Scientist
at Deep Silver Volition.

Dr. Matthew White is Data Scientist at Deep Silver Volition. A former academic, Dr. White helps Volition embed telemetry in titles, mine player data, perform quantitative analysis of playtests, perform A/B tests, and apply statistics wherever they may be needed. Dr. White has previously chaired content for the IGDA GURsig's yearly summit in 2015, a conference at which he has presented many times. He's a member of the user research SIG, and just recently spoke at #GAMESUR London, an IGDA event. He developed Penn State University's game development program, and was previously an IGDA Scholar (2009). Before Penn State, Dr. White ran two small startups in Canada, and did private usability consulting during his Ph.D., which he obtained from Memorial University of Newfoundland.

In the last few years, we have seen a pretty hefty increase in the number of VR systems, solutions, and peripherals that are available to consumers and even the most junior of developers. Starting with the meteoric popularity of the Oculus Rift, a bevy of followers has emerged. Unfortunately though, for a section of the population, none of the research and development into these cool new toys makes much of a difference.

I'm talking of course about those who are made horribly ill by the phenomenon known as simulator sickness. Simulator sickness, a form of motion sickness, is a sort of no-fun phenomenon in which individuals using a simulation or VR device get sick in a variety of unpleasant and often chunky ways. Some of the symptoms include nausea, vomiting, dizziness, lightheadedness, headaches, fainting, existential ennui, forgetting to call your mother, and more. Needless to say, these kind of hangover-esque phenomena keep people from playing games, using simulations, and so on. For this reason, my Penn State student Dylan Poljak (www.yellowdragon.net) and I were very interested in looking for a computationally inexpensive solution to what has become a pervasive problem. This study and the simple software

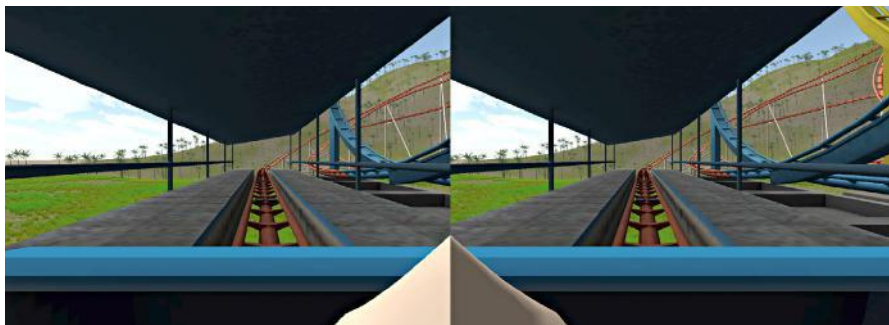
solution it presents were born out of observations of students in my lab, who after putting on the Oculus consistently looked up, then directly down. Our thought process was something like: »What if they're looking for legs?«

Background

There are myriad theories regarding the underlying cause of motion sickness, some of which involve the eyes and a triggering of the vagus nerve. In fact, many of the symptoms of vasovagal syncope (e.g. fainting) are very similar to those felt in simulation sickness – although the syncope symptoms usually do not indicate that precipitous vomiting is imminent – rather, they indicate that you are about to be more horizontal than you may have intended when donning the VR device. A few other studies have made a concerted effort at mitigating the symptoms associated with this illness with varying degrees of success.

One of the studies that inspired our thinking the most was Nasum Virtualis, a study from Purdue University that investigated the role that a virtual nose could play in delaying simulator sickness onset in players¹. The study found that adding a virtual nose to a simulator significantly reduced, or at least delayed simulator sickness in players. The study reported that individuals were able to maintain play without simulation sickness symptoms longer in a situation in which a simple nose was placed in front of the players' faces while they used both the Tuscany Demo packaged with the Oculus Rift, and a simulated roller coaster. This got us thinking – if a nose in the simulator significantly reduced player perceptions of sickness – what might other body parts do?

We started reflecting on our experiences with other games, like »Half-Life«, where for some darned reason, you just don't have legs. One of the real fourth-wall-breakers for me has always been looking down at my feet and just having no lower body to speak of. We made an



One of the studies that inspired the team the most was »Nasum Virtualis«, a study from Purdue University that investigated the role that a virtual nose could play in delaying simulator sickness onset in players.

educated guess (read: guess) that the sickness may be elicited by a vision-perception disconnection. That is to say, motion is seen (e.g. the game visualizes motion), but the game breaks the perceptive immersion by not presenting arms, legs, etc. to the player. Additionally, the visualized motion is not felt by the player in any appreciable way – resulting in a disconnection that has led to sickness in the past².

We decided we'd cook up a study and investigate how the presence of legs affected simulator sickness, if at all, knowing that this was the easier problem to fix. Simulating the feeling of motion is, after all, a bit difficult for two people.

Study and Software Design

We decided to design a traditional random study. During the process, we used Unity to create a 3D environment through which we would expect players to navigate. The objective was to find a ball at the end of an exploratory environment. Sure, not the best game ever, but we were more concerned with players' urge to barf than their enjoyment of the game.

The majority of the environment was created with Asset Store parts, and the assembly took about a week. We also decided that we would employ the same questionnaire used in previous studies, the Simulator Sickness Questionnaire by Kennedy et al³. This instrument has been used extensively and is a fairly sensitive and standard instrument for these types of tests.

Methodology

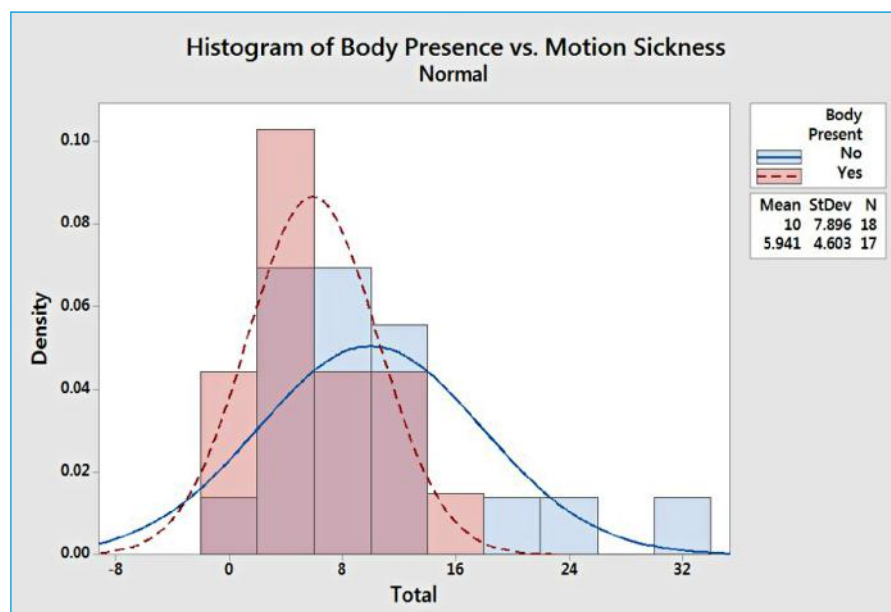
We recruited as many participants as a student and a professor reasonably could in a few weeks, and decided we should randomly split them into two groups – one group would get a body at which they could look (e.g. torso, legs etc.), and one would not. After playing for a bit, we'd give the participants the questionnaire, and review their responses to see if there were any meaningful differences.

Thankfully, the effect size was small enough to require the instrument. That's academic speak for: nobody outright barfed, so we needed a survey to tell if there was any difference or not. It wouldn't require much investigative work if 100 percent of people without legs shot projectile vomit on the wall, now would it?

Findings

As we originally expected, the group with legs had significantly less sickness than the group without legs. We're smart like that. Sometimes. In any case, the effect size was relatively large (about 30 per cent). Like most academic findings, however, we should be careful what we do with this data. The sample size was small, and the sample was collected from individuals who are already familiar with games and technology. As a result, these individuals might be faster technology adopters, and so on.

Nonetheless, this stands to further previous research indicating that the presence of body



As originally expected, the group with legs had significantly less sickness than the group without legs.

parts has a mitigating effect on simulator sickness, significantly reducing number of chunks per hour. This potentially disabling condition makes it nearly impossible for people to use VR, joking aside.

For nerds: the correlation between body presence and sickness was 0.306 using Pearson's r with a p-value of 0.074 (yes, not quite 0.05, but I'm comfortable with that given the preliminary nature of the study, get off my case, ok?). We had 35 participants, and these results seem to be replicable.

Conversation

While the effect size here is considerable, it's worth noting that we had very few people in the study (35). That said, the study was somewhat significant for the size, and we're confident that repeated tests and more sensitive tests that use a wider demographic would continue to validate that point. As with all things academic, further study is needed. In the meantime, the inclusion of a body part in the game seems to further the findings of Nasum's Virtualis: namely, that including elements of a person's real body seems to slow or mitigate the effects of motion sickness. We'd like to run this again using Nasum's methodology to check on time to onset of symptoms when presented with a virtual body.

In the meantime, developers can potentially reduce player churn by including simple fixes. Chances are the player model has been sculpted to some degree for use in cutscenes, cinematics etc., in a three-dimensional game. Where possible, developers should spend the extra resources to provide visible limbs and other body parts in games designed for virtual reality devices, if only to keep them free of sweat, tears, and food particulates.

Dr. Matthew White



The majority of the test level environment was created with Asset Store parts in Unity, and the assembly took about a week.

References

If you want to read more about the references that are mentioned in the text, please see the following publications and links:

1. 'Virtual nose' may reduce simulator sickness in video games by Emil Venere, David Whittinghill
www.makinggames.biz/VirtualNose
2. So, R.H.Y. and Ujike, H. (2010) Visually induced motion sickness, visual stress and photosensitive epileptic seizures: what do they have in common? – Preface to the special issue. *Applied Ergonomics*, 41(4), pp.491-393.
3. Simulator Sickness Questionnaire, Kennedy, Lane, Berbaum, & Lilienthal (1993)
www.makinggames.biz/Simulator-Sickness

FREE-SCALE BACKEND-ARCHITECTURE BASED ON ACTOR MODEL AND SERVICE FABRIC

The team behind OnlineFootballManager set out to look for an adequate backend architecture for their new game. In this article, they explain which technology is best suited for which areas of application.



Kevin Daniel Setiono
Senior Backend Developer
at OnlineFussballManager GmbH

Kevin has been working as a full-stack developer and consultant in various industries for almost ten years. As a consultant he has extensive experience with the Azure Cloud and the creation of concepts for highly scalable backend applications. In 2015 he started working at OnlineFussballManager GmbH. Within the games industry, he's in charge of the conception and implication of the backend systems for the project »Football Empire«.



Andreas Pohl
Game Developer and
Scrum Master
at OnlineFussballManager GmbH

Andreas graduated from the Trier University of Applied Sciences with a Bachelor of Science in Digital Media and Games, specializing in game coding. During his studies he was a project manager in the 4Players league - the eSports platform of 4Players GmbH. Andreas started working at OnlineFussballManager GmbH in 2014 as a backend developer for OFM. He's now responsible for the planning and organization of the backend for their new project »Football Empire«.

We kicked off the production of our new game in the middle of last year. Our goal is to develop a classic football manager with a builder-strategy theme.

In contrast to our existing product OnlineFootballManager (OFM) which started as a browser game twelve years ago, our new project named »Football Empire« is now focused on Android, iOS and Windows Phone (and potentially Windows 10) with Unity.

Thanks to our ongoing long-term development of the browser game OFM we have certain competencies in terms of conception and implementation of backend architectures. This experience helped us find a solid solution for the backend architecture of Football Empire.

In this article, we would like to present the experience and knowledge we have gained from our intensive research, and the solution we eventually went for.

Issues to be addressed

When developing a new backend architecture and infrastructure, there are a lot of issues that need to be addressed. Information about potential users and the resulting strain are particularly crucial for the architecture of the backend. After all, who wants their own game to suddenly go offline after it was featured in a TV spot on a major channel and all of a sudden a bunch of new users want to join the game?

Obviously, such events don't occur at random and providers can take preparatory actions. Yet, the backend needs to be scalable to an extent where even such temporary peaks can be managed. Also, it can be assumed that in an agile environment the requirements of a project are subject to change during development, which makes it very hard to reach a compromise between cost, complexity and availability. For this reason and also due to lacking flexibility, new backend systems are often designed and optimized based on expected peaks. In most cases this procedure isn't cost-efficient since you always pay for the peaks instead of the actual load. The other extreme are luxury problems such as Rocket League where the developers weren't expecting such a massive success and the backend simply couldn't cope with the enormous amount of users. But even if you are theoretically able to scale, in most cases this has to be done manually by a company person, usually DevOps or system administrators. And since we don't have a dedicated DevOps engineer in our team, our solution had to take this into consideration.

Goal

With Football Empire we have the rare opportunity to develop the backend from scratch, thus solving the usual issues through a modern environment. During development, we particularly focus on the following acceptance criteria:

- Scalability of the infrastructure (100,000 – 10,000,000)
- Cost minimization of the infrastructure
- Minimization of the maintenance effort due to limited DevOps resources
- Guaranteeing fast development
- Players should be able to make changes to their teams while the game is running (e.g. substituting a player during a match)

Since we only have limited resources, it's important to us that the development can be carried out within a reasonable time frame and by a limited amount of people only. In order to guarantee this, we need to keep the tasks the team has to deal with as small as possible. This is a slight contradiction to the requirement of a highly scalable infrastructure which is usually very complex and high-maintenance.

Infrastructure overview

We started out the development by evaluating classic architecture solutions. This way we could develop a monolith and scale it horizontally simply through more hardware. We took a similar approach for OFM, and therefore we know that our requirements are hard to meet if at all. The reason for this is the lacking ability to scale individual parts of the monolith. Only the entire monolith can be scaled at once, which usually results in a bottleneck in the persistence layer. That's why a monolith can't be considered given our requirements.

Separating the architecture into different layers like queues, worker, cache and persistence would keep the system more flexible and would be acceptable as a service. With this solution you would be able to scale only certain workers or queues, making the system more flexible. In worst case though it would mean a number of different systems to be understood and maintained. Based on the experience we gained with the browser game OFM we know that such an architecture also involves a considerable maintenance effort. In order to guarantee a solid service, we would have to expand our team by a dedicated DevOps engineer, so this solution isn't ideal for us either.

Modern infrastructure

The first question to face is this: Should you ever doubt a proven solution? Well, computer science especially is an area which is changing quickly and continuously. Even if new solutions aren't necessarily better than established concepts, we were wondering whether there weren't more efficient technologies to develop our backend.

When talking about the development of distributed or highly scalable systems these days, you can hardly avoid mentioning microservices. Especially when you read success stories about the likes of Spotify, Netflix etc.,

the use of microservices seems to be the new modern holy grail.

Microservices

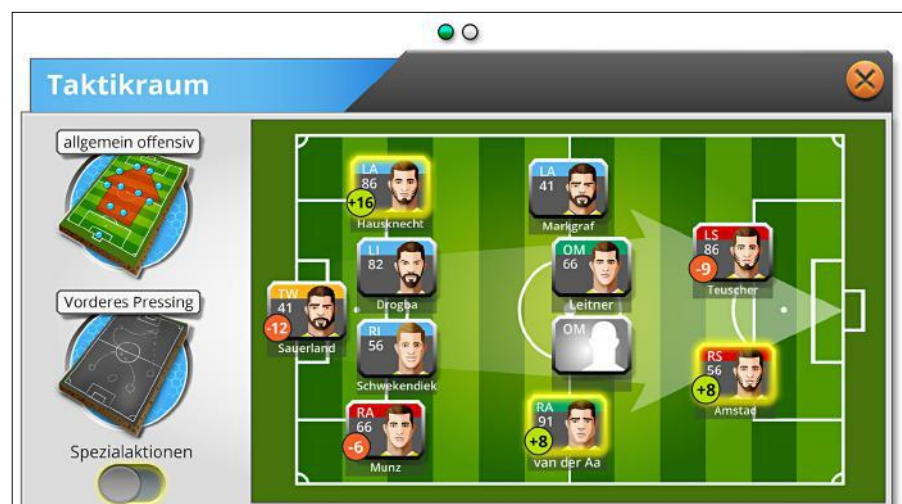
When we started with the development, we asked ourselves whether microservices were really the answer to our challenges and in which areas they would give us a relevant advantage in our case.

In the simplest case, a microservice links logic and data in order to provide an external interface, allowing each microservice to have its own database including the data it works with. That way, high scalability can be attained.

If we approach microservices from a software architecture point of view, one would wish for strong module boundaries in an ideal case. A microservice architecture supports this principle in that a service has no access to the data of another service except via the service interface itself. Those firm boundaries provide an advantage particularly for large teams. That way, microservices force an inherently clean architecture. That's also absolutely possible in a monolith, but requires a considerably stricter discipline from the team. In contrast to a monolith, however, the microservice approach increases the likelihood of a higher modularity of the application.

Since microservices are usually comparably small and always have to manage only one specific task as well as their own data, they can be published fast and independently of one another, which benefits the continuous integration/deployment idea. Additionally, services communicate with each other only via their interfaces, which is why every service can be coded in a different programming language, providing higher flexibility.

Even though microservices provide some advantages, they are still a distributed and thus an inherently very complex system. In a system based on microservices, the services communicate with each other through remote calls which are always linked to costs and the risk of failure.



The team behind OnlineFootballManager is working on a combination of a football manager and a builder-strategy game. For the first time though the project called Football Empire won't be a browser game, but is to be released on mobile devices.

This means if a service is no longer available, the system needs to restore itself and run as usual. This is usually the developer's responsibility.

In order to compensate for the latency resulting from remote calls, a high priority is set on asynchronous execution processes in most distributed systems, making error analyses a lot harder.

In a microservice-based system, we also have to deal with eventual consistency due to the separation of data since it's hardly possible to generate strong consistency in a distributed system. This may become a problem since a distributed system always takes some time to replicate data.

Service or microservice

So far there is no standard definition of what exactly a microservice is or what it's supposed to look like. We would classify the concept of microservices as a subset of service-oriented architecture (SOA) in order to get a definition. According to that, microservices should meet at least the following criteria:

- Clear application boundaries
- Services are autonomous
- Services share scheme and contracts, no classes
- Service compatibility is based on policy

According to Martin Fowler, microservices differ from SOA in that they can be updated independently from other services and each business requirement is displayed in a service of its own.

In practice it looks like this: Let's assume there's a team service responsible for managing a team of football players. Let's also assume a kicker can never exist without a team. If we take the SOA approach, this can be visualized like in [image 1](#).

In [image 2](#) we display each area of our logic in a separate service. That way, the kicker service is responsible itself to generate a new player, providing only the corresponding interfaces.

When should microservices be used?

In summary, it can be said that scalability can be achieved through an increased complexity of a distributed system. Therefore, we had to decide whether to take the risk of a higher complexity of the system. After all, this solution only provides an actual advantage if the complexity of a monolith is very high. A separation into microservices isn't justified for a trivial monolith because the complexity overhead would be too big. Besides, it needs to be clear from the beginning which stress requirements there are in order to develop efficient microservices.

If you ask yourself this question, it usually indicates that a separation into microservices isn't advisable for the time being.

The solution?

We know now that we don't necessarily want to accept the higher complexity of distributed systems, but at the same time we want to meet our scalability requirements. We also know that a high scalability is indeed possible with the established approaches, yet not trivial. Our Lead Game Designer and product owner, Guido Schmidt, correctly put it this way: »We have the competence, but not the experience with such systems.«

Therefore, we need a system that offers the benefits of distributed systems while reducing the complexity of such systems at the same time by using practical standard procedures.

Through thorough analyses, we eventually became acquainted with the so-called »actor model« which might solve our problems.

Actor model

The actor model is a model for the development of concurrent software. The concurrency or parallelism of software is becoming more and more important to guarantee a fast application. The execution of a software became faster and faster as the hardware was becoming faster and faster in the last few years. At the moment though it seems a physical limit has been reached, e.g. with regard to

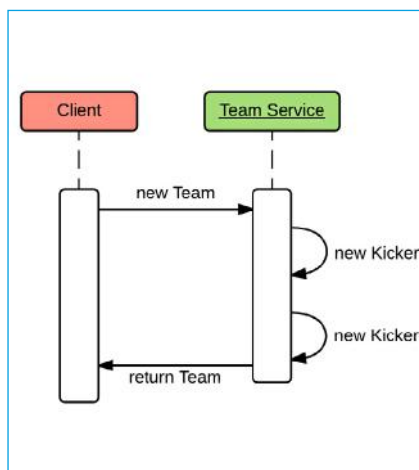


Image 1: Example of an SOA-based team service.

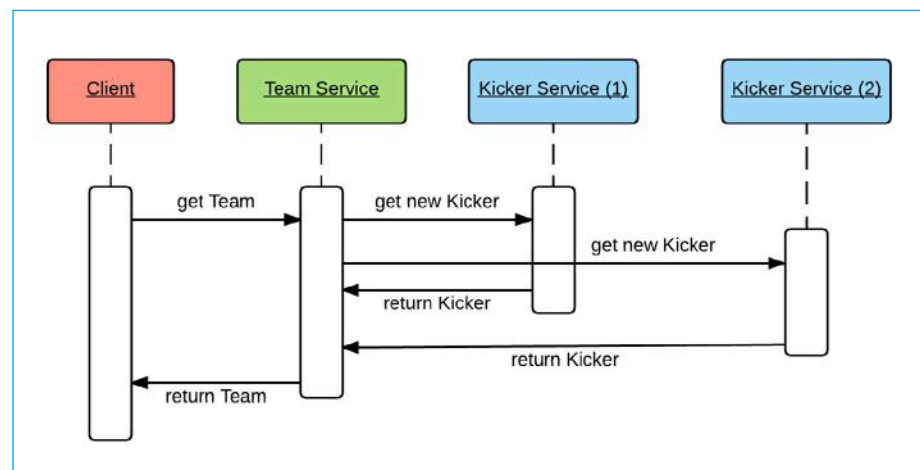


Image 2: Example of a microservice-based team service.

the CPU, where it's not possible any more to increase the clock speed. In order to accelerate the software, the processes are parallelized through multi-threading, which can be executed simultaneously through several cores in the CPU. In order to exchange data between the threads, a shared state is used, which bears the following challenge: If a thread accesses this state »in writing«, another thread has to wait until the first one is released again. In computer science, this shared state is called »mutable shared state«. Waiting for the access rights of the mutable shared state is achieved through a lock and a critical zone. Only one process or thread may be in a critical zone at a time. This can be achieved through a lock in the source code (C# for example). In **image 3** we compare a »counter« class which is »thread-safe« (several threads may access the class) with the same functionality as actor in **image 4**:

What is an actor?

The actor model adopts the philosophy »Everything is an actor« which is analog to »Everything is an object« in object-oriented programming (OOP). This allows software developers an easy entry into the development of actor-based systems.

An actor is a unit for calculations, which manages its own state. In order to exchange data between different actors, immutable messages are sent. The messages are sent asynchronously and temporarily stored in a mailbox. The mailbox serves as a waiting queue for applications submitted to the actor. A message in the mailbox is analog to a function call in programming and can be programmed in a similar way. Upon receipt of a message an actor can carry out the following actions:

- send a finite number of messages to other actors

```
namespace Ofm.Example
{
    /// <summary>
    /// Counter class
    /// </summary>
    public class Counter
    {
        /// <summary>
        /// The lock object
        /// </summary>
        public object lockObject = new object();

        /// <summary>
        /// The counter
        /// </summary>
        private int counter;

        /// <summary>
        /// Adds one to the counter.
        /// </summary>
        public void AddCounter()
        {
            lock (this.lockObject)
            {
                this.counter++;
            }
        }

        /// <summary>
        /// Gets the counter.
        /// </summary>
        public int GetCounter()
        {
            lock (this.lockObject)
            {
                return this.counter;
            }
        }
    }
}
```

Image 3: A class for the management of a counter (thread-safe).

```
namespace Ofm.Backend.ActorBackend.actors
{
    /// <summary>
    /// Counter Actor
    /// </summary>
    public class CounterActor : StatefulActor<CounterState>, ICounterActor
    {
        /// <summary>
        /// Adds one to the counter.
        /// </summary>
        public Task AddCounter()
        {
            this.State.Counter++;

            return Task.FromResult(true);
        }

        /// <summary>
        /// Gets the counter.
        /// </summary>
        [ReadOnly]
        public Task<int> GetCounter()
        {
            return Task.FromResult(this.State.Counter);
        }

        /// <summary>
        /// Called when [activate async] is called, initializes the State.
        /// </summary>
        /// <returns></returns>
        protected override Task OnActivateAsync()
        {
            if (this.State == null)
            {
                this.State = new CounterState();
            }

            return base.OnActivateAsync();
        }
    }
}
```

Image 4: An actor for the management of a counter (thread-safe).

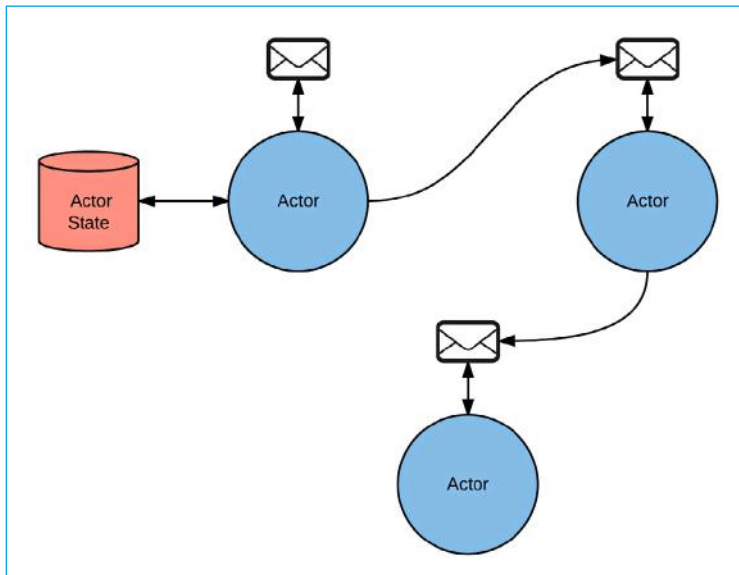


Image 5: A visualization of how actor systems look like.

- generate a finite number of new actors
- manage its own state

Frameworks for the actor model

There are many frameworks for the development of a distributed system based on the actor model. The most popular are Akka, Orleans, Service Fabric and Orbit. We have evaluated various frameworks for our distributed system, of which we will present just a few.

Akka

Akka (<http://akka.io>) is an open-source project based on the Apache 2 license, simplifying the creation of a concurrent software in a distributed system. Akka is designed for the Java Virtual Machine, although there is a .NET port as well. The main focus of Akka is on delivering a framework that allows the software developer to change everything in the system. That means the framework doesn't hide the complexity of distributed systems in order to offer the software developer all sorts of possibilities. According to our analysis, the DevOps effort of an Akka system is rather high.

- **Actor creation & load balancing:** The actors are created by their parent object, which means that all actors must be started and stopped explicitly. This also gives the creator the possibility to equip the actor with parameters during its creation. It has to be specified explicitly where the newly created actor is located, although this behavior can also be changed through the configuration. As a result, the software developer needs to pay attention to the load balancing.
- **Persistence:** The state of an actor is persisted via event sourcing where messages the actor has received are being stored. If the actor has to restore its state, the messages will be released again. Due to performance reasons there is the possibility

to use a snapshot where only messages after the snapshot are released. There are plugins available for Akka, which simplify the behavior.

- **Timers and reminders:** Akka supports timers for an actor only as long as the actor is active. If the actor is currently not used, the timer won't be executed either. The counterpart of timers are reminders. These are also executed if the actor is no longer active, and will activate the actor at certain times. Reminders are currently not supported by Akka.

Orleans

Orleans (<https://github.com/dotnet/orleans>) is an open-source project which is developed further by the community. It offers non-specialists the chance to develop efficient, scalable and highly available distributed systems. Orleans implements a virtual actor model and is meant to be used in the cloud. The virtual actor model assumes that an actor always exists. If an actor doesn't exist, it will be created automatically. Unlike Akka, the system of Orleans saves you a great deal of work.

- **Actor creation & load balancing:** An actor is virtually in Orleans, which means that the system makes sure that a necessary actor always exists. The actors are created automatically and distributed via the cluster, hence the system offers a good distribution via the cluster from the get-go, even if the software developer isn't familiar with topics like load balancing.
- **Persistence:** The state of an actor in Orleans is persisted through snapshots. The developer needs to specify exactly when the state is to be persisted. The state will then be filed as a snapshot and be reloaded.
- **Timers and reminders:** Orleans supports timers and reminders. Timers are only active as long as the actor is active. However, reminders wake up an actor, even if it was previously removed from the memory. Compared to Akka, Orleans offers the advantage of reminders here.

Service Fabric

Microsoft has recently released Service Fabric (<https://azure.microsoft.com/en-us/services/service-fabric/>) as a »public preview«. Service Fabric has many similarities to Orleans: For example, Service Fabric is a virtual actor model, too. Besides actor development, Service Fabric also offers the development of reliable services. These services are efficient in terms of managing lists, or they can also be used for configurations.

Unlike Orleans and Akka, Service Fabric isn't open source. On the other hand, the integration of Service Fabric into the Azure cloud or on the Windows server (cluster) is more intuitive.

- **Actor creation & load balancing:** Service Fabric also uses a virtual actor model and is

similar to Orleans in terms of the actor creation and load balancing behavior. However, custom metrics may be specified for the orchestration as well, so that, for example, CPU-intensive actors will only be created on the more efficient node. Apart from that, actors are always regarded as reliable and available.

- **Persistence:** In Service Fabric, the state of an actor is persisted almost after every message. There is the possibility to declare a message as »read-only«, so the state won't be persisted again. It's advisable to use read-only messages to increase the performance. The software developer can choose between a »volatile state provider« and a »key-value store state provider«. There is also the chance to create your own state provider, however, there are no useful examples for that yet. The »volatile state provider« only keeps the state of an actor in the memory and will lose it when the actor becomes inactive. In contrast to that, the »key-value store state provider« persists the state on the hard drive, keeping the state, even when an actor has become inactive. The state will be executed after processing a message (function call), and the software developer has the chance again to declare a message as »read-only«.
- **Timers and reminders:** Service Fabric supports timers and reminders analog to Orleans. Timers are only active as long as the actor is active. However, reminders wake up an actor, even if it was previously removed from the memory.

Which framework do we use?

In our team we have competencies in the DevOps department, even if the experience in that area isn't ideal. Compared to Orleans and Service Fabric, Akka offers the most possibilities, but also the biggest responsibility to exploit them. The porting of Akka with Akka.NET hadn't been stable yet, so we decided against Akka.

Orleans is a very stable and »battle-tested« framework with less DevOps obstacles than Akka. Yet we went for »newer« framework Service Fabric. We had a great experience with Microsoft's support with regard to Service Fabric and Azure Cloud. Christian Geuer-Pollmann, Thorsten Rendel and Damir Dobric provided their support when we made our decision for Service Fabric. Besides, Service Fabric offers reasonable standard behavior for most cases, which relieves us of a lot of the worries we had. The upgrade/downgrade mechanisms of software in the Service Fabric cluster can easily be carried out through the deployment tools. The bottom line is that we consider Service Fabric to be the best overall solution.

We have our Service Fabric cluster hosted in the Azure cloud. The Azure cloud linked with Service Fabric allows the nodes to scale

automatically. At the moment, the automatic addition and removal of nodes in the cluster isn't working yet. The following shows our daily work with Service Fabric.

Working with Service Fabric

Service Fabric is a simple system that every software developer of object-oriented programming is able to work with. We started working with Service Fabric when the examples and documentation hadn't even been quite finalized yet. We present non-trivial challenges with the system and show the elegant solutions with Service Fabric. The complexity of a distributed system is rooted in Service Fabric. In order to fully exploit the framework, one must put emphasis on good load balancing and the reduction of waiting times.

Load balancing and waiting times

A distributed system should reduce waiting times and distribute the load in an ideal way.

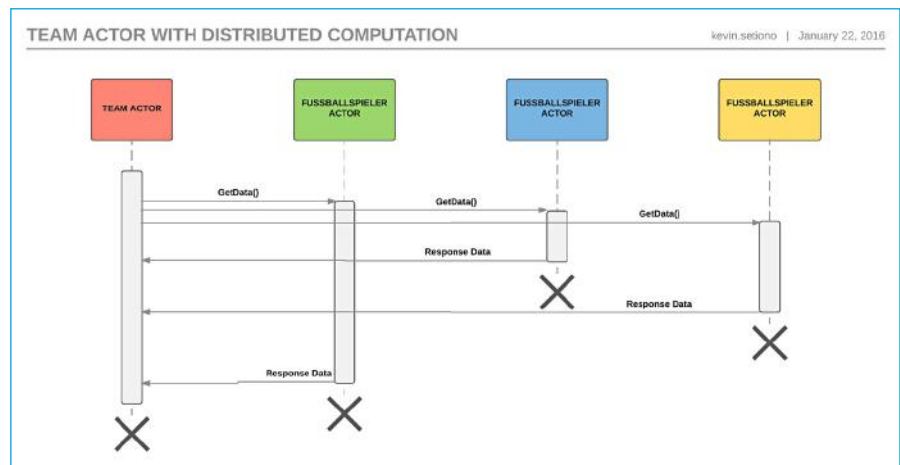


Image 6: A team actor with distributed computation.

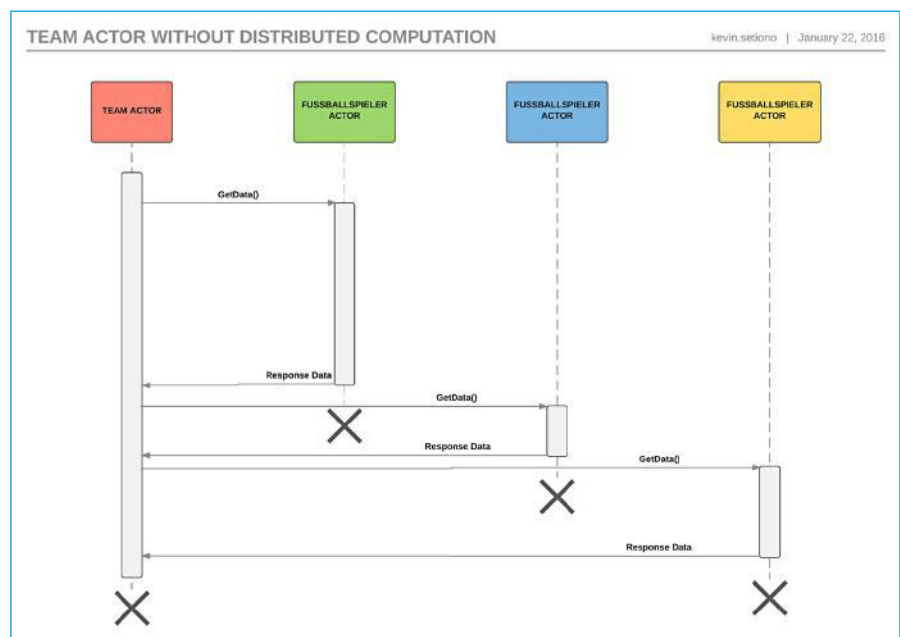


Image 7: A team actor without distributed computation.

The software developer must support the framework to reduce the waiting times. In our game, for example, every football player has

Request ID	Method	Requests	Response Times
/MatchmakingActor.GenerateNewGame	Microsoft.ServiceFabric-Actors/ActorMethod/Start	30 requests	2,123.139ms, 9 requests, 3,109.587ms, 5,914.585ms, 5,914.585ms
/UserActor.CreateDefault	Microsoft.ServiceFabric-Actors/ActorMethod/Start	68 requests	20.606ms, 18 requests, 3,812.852ms, 9,634.888ms, 10,116.632ms
/UserActor.PickEnemy	Microsoft.ServiceFabric-Actors/ActorMethod/Start	25 requests	1,584.319ms, 7 requests, 3,812.852ms, 5,109.587ms, 5,109.587ms
/UserActor.GetTeam	Microsoft.ServiceFabric-Actors/ActorMethod/Start	82 requests	1,522.492ms, 5 requests, 3,458.369ms, 5,109.587ms, 6,521.271ms

Image 8: Service profiler summary page.

Activity	Duration
Activity ActorMethod[/V/1714/./methodName=UserActor.GetTeam]	3,487.728
Thread CPU=376ms	0.435
Activity ClientOperation(f1e77e97-96ab-4f13-b8f2-19bcd70dd11f/Actio...	3,487.293
Thread CPU=376ms	2,403.767
OTHER <<RtlUserThreadStart>>	2,403.641
IUserActorMethodDispatcher.OnDispatchAsync	2,403.641
UserActor.GetTeam	2,403.641
UserActor.ProcessTraining	2,403.04
ITrainingsablaufActor.ProcessTraining	1,201.502
OTHER <<ActorProxy.ContinueWithResult>>	0.042
OTHER <<Task.Wait>>	1,201.495
CPU TIME [CPU_TIME]	0.116
WAITING [BLOCKED_TIME]	1,201.379
TeamHelper.GetTeam	0.601

Image 9: Service profiler detail get team page.

their own actor. This actor manages training points, experience points etc. To select a football player, the client wants to display a list of all football players on the team. In order to keep the waiting times to a minimum, we distribute requests for the football player actors simultaneously and wait for all football player actors instead of processing one after the other. In other words: We fill the mailbox of all football player actors on our team and wait for the reply of all of them (see image 6). The software pattern is called »distributed computation«. In a sequence chart you can see the theoretic benefit. If we keep processing one actor after the other, it will take more time to do the same task (see image 7).

Service Fabric offers everything it takes to develop efficient software. But every so often it's »easy to use, but hard to master«. In order to detect bottlenecks in the application, for example through non-ideal load balancing due to excessive waiting times, Service Fabric offers the necessary tooling – the service profiler.

Service profiler

The service profiler shows the performance of requests and calculations within in the system. It can currently be used by invitation only (»invite-only preview«). It's hard to replicate performance problems during a live operation since they only occur in rare constellations. The overhead on the overall system can be configured in the service profiler. Only a small agent is used for analyzing certain events. The developer doesn't have to make any changes within the application in order to get a solid base tracking of the performance. The service profiler lists all requests to the system (see image 8). There is no detailed information available for all requests since the service profiler tries to display only relevant information. If all information was always saved, the service profiler would cause excessive system load. It is possible though to display detailed information of requests. In image 9 we can see a request for all football players from a team. Our system carries out the team training and then sends back the

```
<!-- Code package is your service executable. -->
<CodePackage Name="Code" Version="1.0.0">
  <EntryPoint>
    <ExeHost>
      <Program>ServiceBackend.exe</Program>
    </ExeHost>
  </EntryPoint>
</CodePackage>

<!-- Config is the contents of the Config directory under PackageRoot that contains an independently-updateable and versioned set of custom configuration settings for your service. -->
<ConfigPackage Name="Config" Version="1.0.0" />
<DataPackage Name="Data" Version="1.0.0.0" />
```

Image 10: Segment of the XML file Service Manifest.

updated data of the football players. The data in the service profiler shows that the training takes up half of the time. The display »WAITING [AWAIT_TIME]« (see [image 9](#)) is, to a large part, the communications overhead. If an actor requires the reply of another actor or service, it will wait. This is where we could optimize if necessary.

Configuration of the application

There are configurations in our game and most other applications. Good load balancing and short waiting times are also useful when accessing configurations. Service Fabric allows updates for data (configurations for the application) without distributing a new code. All it takes is creating a folder and indicating it in the configuration. The folder needs to be created in a special location. A service – in this case no actor – can access the configuration during the initialization. If the configuration is renewed, the service will receive an event and be able to renew its internal cache. The service is stateless because it keeps its own cache only in the memory and can therefore distribute the load freely to the cluster. An actor can use the service to read values from the configuration. For more detailed information about the data package handling in Service Fabric we provided a segment of the source code in [image 11](#).

Matchmaking with the actor model

The actor model isn't ideal for tasks which require the data of a lot of actors, for example, the team size of all team actors. In order to avoid large lists, we chose an intelligent router (see [image 12](#)).

The matchmaking routing service has no state of its own (stateless), which is why it scales unrestrictedly. The router manages its own cache through buckets. Each bucket actor includes the teams with the corresponding strength, for example, team strength 5 includes all teams with the team strength 5. A bucket actor is always limited to the last 10,000 teams. This limitation allows the system to quickly find an opponent. During the game the team strength changes over time, so that the player doesn't notice that limitation.

- **My team has become stronger:** If the strength of a team changes, the team actor sends a message to the matchmaking routing service which adds the team to the bucket actor and removes it from the old bucket actor if it was listed there.
- **We want an opponent:** We would like an opponent assigned for a friendly match, so we send a request to the matchmaking service which again sends a request to the matchmaking routing service, stating who we are, including our team strength. The matchmaking then decides randomly and through balancing values which bucket actor has been selected for us and starts looking for an opponent there. Special cases like »The bucket is empty«, »There is no opponent

```
/// <summary>
/// Gets the Data Package
/// </summary>
/// <returns>Data Package Version</returns>
public Task<int> GetVersion()
{
    if(!this.isReady || this.Data == null)
    {
        throw new ServiceNotReadyException(ActorConfiguration.DataServiceTypeName);
    }

    return Task.FromResult(this.Data.Version);
}

/// <summary>
/// Initializes Data Packages Changes
/// </summary>
/// <param name="cancellationToken">Cancellation Token</param>
/// <returns>Task for Service Fabric</returns>
protected override Task RunAsync(Cancellation token cancellationToken)
{
    IList<string> packageName = this.ServiceInitializationParameters.CodePackageActivationContext.GetDataPackageNames();

    foreach (string name in packageNames)
    {
        System.Fabric.DataPackage dataPackage = this.ServiceInitializationParameters.CodePackageActivationContext.GetDataPackageObject(name);
        this.ProcessDataPackage(dataPackage);
    }

    this.ServiceInitializationParameters.CodePackageActivationContext.DataPackageModifiedEvent += (sender, e) =>
    {
        this.ProcessDataPackage(e.NewPackage);
    };

    return base.RunAsync(cancellationToken);
}

/// <summary>
/// Process Data Package
/// </summary>
/// <param name="dataPackage">New Data Package</param>
private void ProcessDataPackage(System.Fabric.DataPackage dataPackage)
{
    string filename = Path.Combine(dataPackage.Path, "DataPackage.json");

    if (!File.Exists(filename))
    {
        return;
    }

    this.isReady = false;
    try
    {
        using (StreamReader reader = new StreamReader(filename))
        {
            this.DataJson = reader.ReadToEnd();
            this.Data = JsonConvert.DeserializeObject<Shared.Entities.DataPackage>(this.DataJson);
        }

        this.isReady = true;
    }
    catch (Exception e)
    {
        ServiceEventSource.Current.ServiceMessage(this, e.ToString());
    }
}
```

Image 11: Source code segment of a data service for the data package handling in Service Fabric.

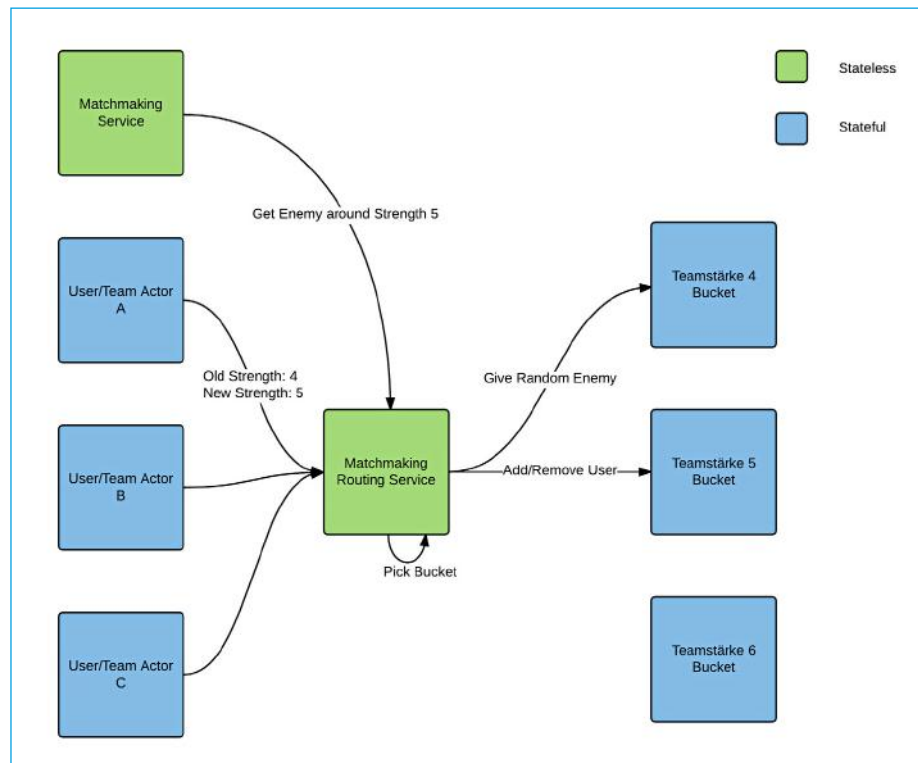


Image 12 : Architecture of our matchmaking system in Football Empire based on the actor model.

based on the balancing« etc. are handled by the matchmaking routing service. The service reports an exception if the search is taking too long.

- **Advantages:** The long-term task to look for an opponent is carried out in a stateless service. To put it in other words: The service can be carried out by any node in the cluster, scaling unrestrictedly. Calculations are independent upon receipt of the message, and the buckets serve as an optimized cache. The system passed our test and always found an opponent for our team quickly and comprehensibly.

Conclusion

At the start of the project we were literally overrun with potential approaches for our backend. The number of infrastructure providers is massive, and the features aren't always immediately apparent. After our comprehensive analysis phase we realized that architecture and infrastructure should ideally always be viewed together because you won't always achieve the best result if you commit to an architecture and just choose the infrastructure provider afterwards.

During our evaluation we talked to some providers and industry colleagues and discussed the state of the art. The feedback we received from these discussions led to thinking about non-established systems as well.

One example would be the actor model which can be implemented and run in a lot of different ways. It would also have been possible, for example, to host Orleans with Google

or Softlayer. However, in that case our team would be responsible for the infrastructure, which would be a bit problematic due to the lack of DevOps resources.

We have a strict schedule for our project, which is why we want to act as efficiently as possible and focus on programming the game rather than managing the infrastructure. And this is where we get support from the Service Fabric with the Azure cloud since it offers a low-maintenance runtime environment for the actor model.

Even if the actor model can't solve all issues, it offers a useful abstraction of a distributed system. This abstraction allows the software developer to solve the challenges in the familiar paradigm of the object orientation.

Thanks to this decision we have reached all of our goals. The development time of the backend logic took less time than expected, so we had more time to spend on the quality assurance. **Andreas Pohl, Kevin Daniel Setiono**

TOOLS

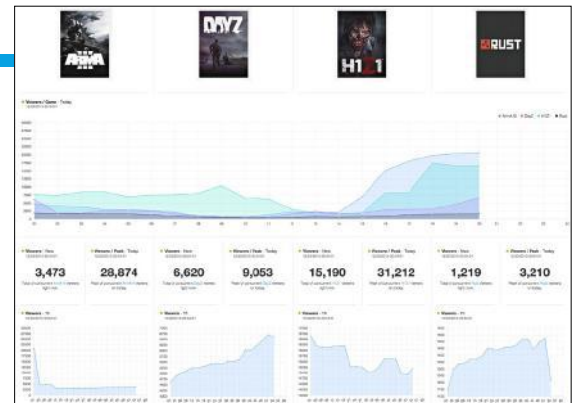
You need great tools in order to build great games. Experts from the industry share their view on recent updates as well as new software solutions and reveal their favorite little helpers.

LOOTILITY WWW.LOOTILITY.COM



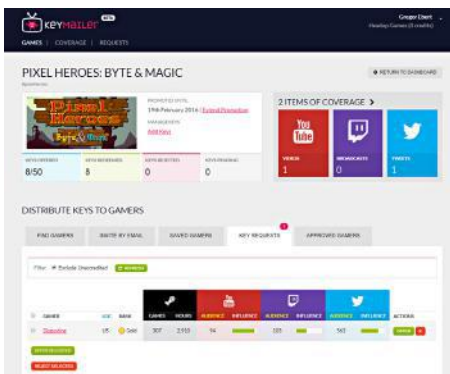
Gunnar Lott
Owner of PR and text agency
Visibility Communications

Lootility is an analysis tool for Twitch. It was developed by Berlin-based team »Fish in the Woods« as part of another application and is currently in beta phase. Lootility shows searchable live data from Twitch like the number of channels or spectators per game, with options to compare, and even offers insights into the channels themselves. If you want to know the percentage of League of Legends spectators that are currently following »C9Sneaky« or whether World of Tanks has more spectators than Darkest Dungeon, look no further. Lootility is still in beta phase, but the guys in Berlin generously send out invite codes. Note that in the current version historical data isn't accessible yet.



Lootility is still in the beta phase, but can already be used for analyzing the traffic of Twitch.

KEYMAILER WWW.KEYMAIL.CO



Who is to receive a review key? Keymailer provides a detailed overview of the user numbers of the various platforms.

Keymailer is a useful system for the quick distribution of review keys between social media influencers and licensors. Twitchers and YouTubers who register will be checked and can then be either contacted directly or request a key themselves. Setting up the keys on the publisher/developer side is extremely easy, and the system is even soon to be extended to include Xbox One and PS4 codes as well. What's nice is that with each request the respective user numbers of Twitch, YouTube and Twitter are displayed as well, which allows for quick decision making. If a title is mentioned by a registered user or if a user creates a video, the link appears automatically in a coverage report. Keymailer is currently still in beta phase (which is free of charge); targeted evaluation tools or marketing measures in the tool itself will later be subject to charge.



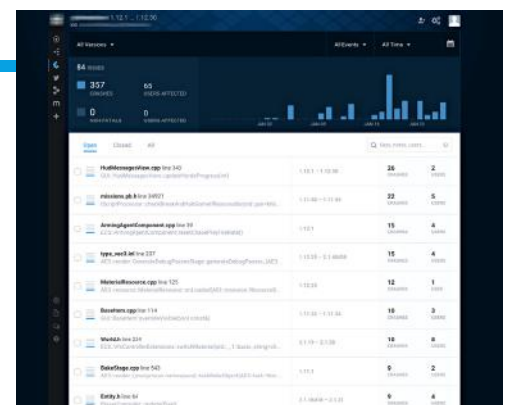
Gregor Ebert
PR Manager
at Headup Games

CRASHLYTICS WWW.CRASHLYTICS.COM



Max Röhrbein-Kling
Engine Programmer
at Deep Silver FISHLABS

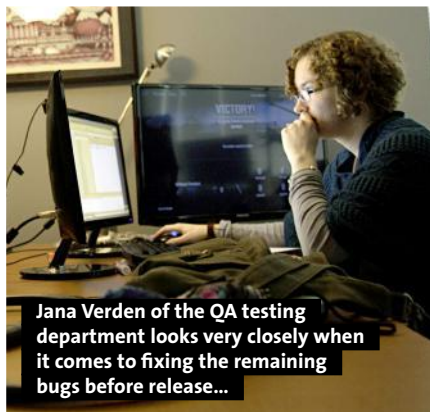
Crashlytics which is now part of Twitter's Fabric Framework is used by developers to evaluate and analyze crashes of iOS or Android apps. Crashlytics collects the data of the crashes as so-called crash logs and makes them available on a website that's accessible to developers who can then, for example, analyze the frequency of individual problems or receive detailed information on the stack trace of a cause. In addition to the crash logs, the developers can provide further log notifications in order to make the analysis easier. The collected crash logs are grouped based on the time the crash occurred and on the version of the app. This allows conclusions to be drawn on the time a bug crept in. Crashlytics is free of charge, very easy to integrate (in Unity3D, e.g., it's available as a plugin) and is a great help to develop a stable product.



Crashlytics provides developers with a constant overview of current issues in their app.

A day in the Life of...

WARGAMING CHICAGO-BALTIMORE



Jana Verden of the QA testing department looks very closely when it comes to fixing the remaining bugs before release...



The mistress of numbers – Business Intelligence analyst Erica Absetz never loses her good mood, no matter how complicated the numbers are.



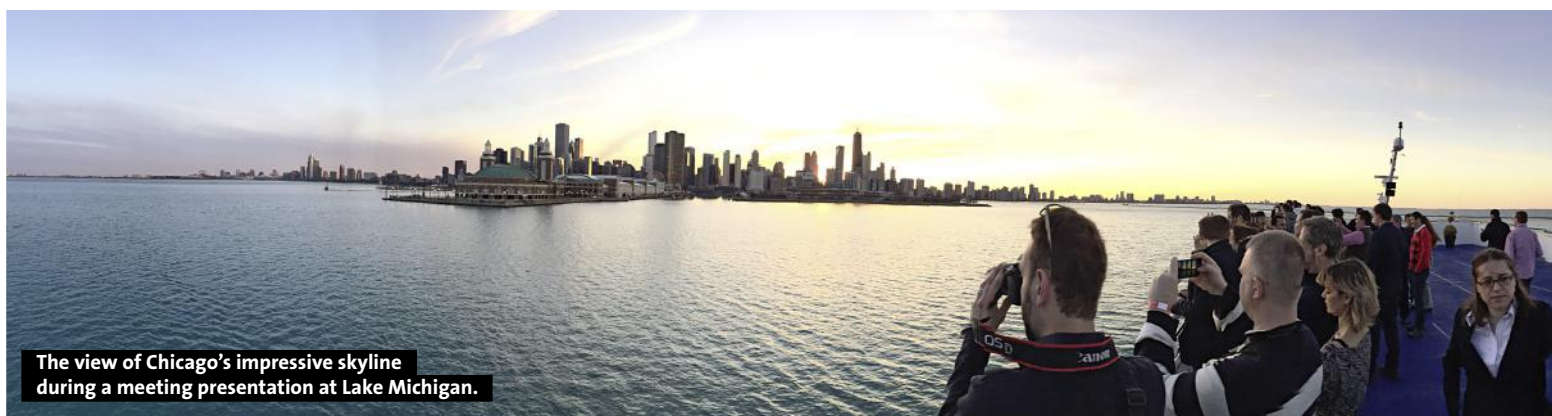
As Senior Sound Designer Brendan Blewett is not only responsible for the music in »World of Tanks« but also for the blaring tank noises of chains and gunshots.



Lead Game Designer Jeff Gregg (in the center wearing a base-cap) during the daily four fixe with his team in a casual atmosphere.



Associate Producer Jonathan Harper is pleased with the smooth start of »World of Tanks« on PlayStation 4 on January 19, 2016.



The view of Chicago's impressive skyline during a meeting presentation at Lake Michigan.



European guests are always happily welcomed by the development team in Chicago – pictured here are Senior Communication Coordinator EU East Ingo Horn and PR Manager Poland Konrad Rawinski at a press event.



The gate that takes guests of Wargaming Chicago-Baltimore into the world of the console versions of »World of Tanks« is an »inconspicuous« elevator.



Associate Producer Jonathan Harper is checking the DualShock controller support in an early version.

The Wargaming studio in Chicago is mainly in charge of bringing the successful free2play titles of the Belarussian developer to consoles - like they just did in January with World of Tanks for PlayStation 4. In the following, the team grants us insight into their everyday work.

Has every reason to smile thanks to uncomplicated processes at Greenlight meetings: Live Publishing Producer Darold Higa.



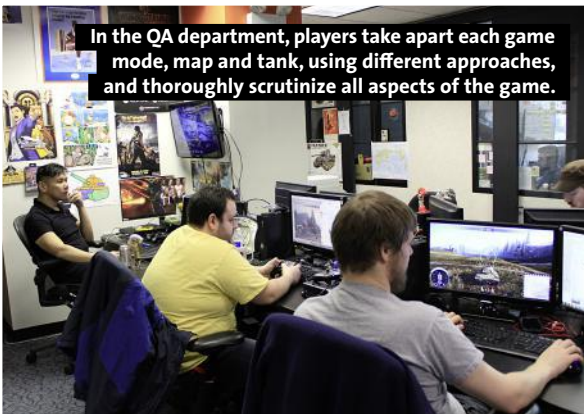
The yearly »Chili Cook Off« claims its yearly victims overestimating their heat tolerance...



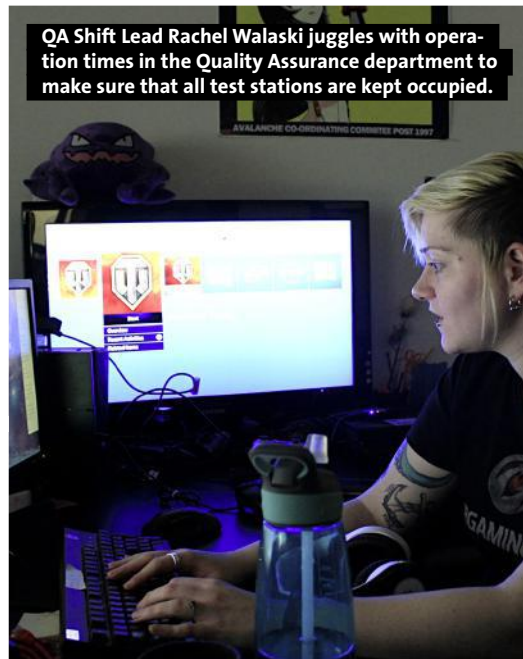
More than 50,000 polygons per tank require respective textures that the graphics department shows its full committed to.



In the QA department, players take apart each game mode, map and tank, using different approaches, and thoroughly scrutinize all aspects of the game.



QA Shift Lead Rachel Walaski juggles with operation times in the Quality Assurance department to make sure that all test stations are kept occupied.



Producer Matt Monforte (wearing a blue t-shirt) and UX QA Lead Jared Pace during some of their heated discussions about the user interface - which is why they have a fan to cool them down (not only on hot days).



The regular meetings of the graphic designers make sure that the PS4 version of »World of Tanks«, too, will soon feature the full number of over 350 tanks.



A huge amount of work was put into the comprehensive fleet of the game, so that the models have a detailed and realistic look on screen.



Great emphasis is put on team play - here are some of the not so camera-shy creative minds behind the newly released PlayStation 4 version of »World of Tanks«.



MAKING GAMES

THE NEXT ISSUE

03/2016
MAKING GAMES
WILL BE AVAILABLE ON APRIL 22ND, 2016

Culture Shock

Steam, various App Stores as well as PlayStation Network and Xbox Live are easy toolsets for developers to publish their games worldwide, opening up markets like China, Russia or Brasil for example. Those were not reachable without distribution or publishing partners in the old days, but this new freedom brings along a lot of problems. Our experts will give advice on which countries prefer what kind of genre or which platform to play on, they will explain how games have to be adapted for different territories or which legal restrictions there might be as well as how to find the right partner in an unknown culture.

Other topics

Traversal Level Design

We take a look at how traversal levels like in »Tomb Raider« or »Assassin's Creed« are created and how to make them fun.

Post Mortem on Bastion for PS Vita

The team of BlitWorks takes a look back at the port of Bastion for PS Vita and explains how they switched the game from C# to C++.

Music and Sound in Horror Games

Sounds and music are very important to create an atmosphere in games – especially in horror games. Our expert explains how to do that right.

IMPRINT

Webedia Gaming GmbH
Redaktion Making Games
Ridlerstraße 55
80339 München
Tel.: 089 / 244 136-0, Fax: 089 / 244 136-118
General managers: Nicolas John
Editor-in-chief: Heiko Klinge (v. i. S. d. § 55 Abs. 2 RStV) - see address of the publishing house
GameStar GmbH is a 100% subsidiary of
Webedia Gaming GmbH
Ridlerstraße 55, 80339 München
Tel.: +49 (89) 244 136-600, Fax: -89600
General Manager: Nicolas John
Register court: District court of Munich
Commercial Register Number: HR B 218 859
USt-ID Nr.: DE 300 33 09 61
Shareholder of Webedia Gaming GmbH are the majority shareholders Webedia GmbH (District court of Charlottenburg, HR B 114531 B), as well as PietSmiet Holding GmbH (District court of Cologne, HR B 84294) and Connected 4 Media GmbH (District court of Munich, HR B 218234)

DISTRIBUTION TRADE CIRCULATION

MZV GmbH & Co. KG
Ohmstraße 1, 85716 Unterschleißheim
Telefon: 089 / 319 06-0, Fax: -113
E-Mail: info@mzv.de
Web: www.mzv.de

EDITORIAL TEAM

Chefredakteur

Heiko Klinge

Managing Editor

Sebastian Weber

Editors

Yassin Chakhchoukh

Patricia Geiger

Markus Schwerdtel

Design

Sigrun Rüb (ltd.)

Manfred Aumaier

Alexander Wagner

Eva Zechmeister

Anita Blockinger

(Freelancer)

Editing (Freelancer)

Marion Schneider

Translation

Bettina Wilding

Contributors:

Daniel Dumont, Christoph Ender, Mark Estdale, Balázs Farkas, Alexander Karl, Jan Klose, Yowan Langlais, Carlos Lievano, Andreas Pohl, Christian Schlütter, Kevin Daniel Setiono, Mischa Streckler, Jan Theysen, Arne von Ohlsen, Timm Walter, Dr. Matthew White

We thank our interview partners:

Stefan Marcinek

Office Management:

Isa Stamp, Anita Thiel, Annie Weissenberger

CONTACT US

Advertising Sales:

Ralf Sattelberger 089 / 244 136-730
ralf.sattelberger@webedia-group.com

Director Sales:

Ralf Sattelberger 089 / 244 136-730

Article Suggestions and Comments:

editor@makinggames.biz

Recruiting & Projects:

projects@makinggames.biz

Address changes and Subscriptions:

kundenservice@makinggames.de

Changes company register:

info@makinggames.biz

Betreff: Company register

Ad approval for print issue:

Manfred Aumaier, 089 / 244 136-668
anzeigendisprint@gamestar.de

Ad approval Online:

werbemitteonline@idg-eas.de

Media rates:

www.makinggames.biz/mediadaten

Printing:

SDV Direct World GmbH
01159 Dresden

CUSTOMER SUPPORT

Subscriptions and single issue orders:

Making Games Kundenservice
ZENIT Pressevertrieb GmbH
Postfach 81 05 80, 70522 Stuttgart
Tel.: +49 (0) 711 / 7252-275
Fax: +49 (0) 711 / 7252-377

Austria: Tel.: 01 / 219 55 60

Switzerland: Tel.: 071 / 314 06-15

E-Mail: kundenservice@makinggames.de

Web: www.makinggames.biz/shop

Prices (print):

Single issue: 6,90 Euro

Subscription: 35,40 Euro per year
(includes 6 issues)

Please contact us for foreign prices and bulk discounts

Payment:

Postbank Stuttgart, BLZ 600 100 70
Konto-Nr.: 311 704

How to reach the team of Making Games:

Webedia Gaming GmbH
Redaktion Making Games
Ridlerstraße 55
80339 München
Tel.: 089 / 244 136-0, Fax: 089 / 244 136-118

LIABILITY INFORMATION

© Copyright Webedia Gaming GmbH

Liability: Despite review neither the editorial team nor the publisher are liable for the validity of any publication. Any paper in Making Games is published without consideration of possible patent protections. Any names of products are used without warranty of free utilization.