Postmortem: Days of Wonder's Ticket to Ride Pocket

PRINT

By Gerald Guyomard,Yann Corno

When asked what business we're in at Days of Wonder, we tend to simply reply: "we do games".

"Computer games?"

"No, board games."

"Ah... You mean like Monopoly?"

(This question is usually followed by an embarrassed silence on the asker's part.)

"They still make those?"

"Well, we do both actually -- cardboard and digital board games. We make games that are just plain fun to play on any platform."

When we first embarked on this adventure some 10 years ago, launching Days of Wonder as a traditional board game publisher, what attracted us to the industry was its staid and steady pace.

Far from the boom-and-bust, hockey-stick cycles of video games and the rat race of Silicon Valley, this was an industry where the CEO of the dominant player (Hasbro) was a grandson of the company's founder, and where the top selling product, Monopoly, was the very same board game that was on top some 70 years ago!

The term "disruption" has become something of a buzzword in technology recently but, if there was ever a perfect target for disruption, we saw no better industry to train our sights on than the world of board games.

As the founders of a 3D graphics software publishing company called Ray Dream, we were guilty participants in the 3D graphics arms race that dominated the mid-1990s and contributed to the ruin of many a good video game. At the time, the hyper-focus of game developers on fancy graphics and the technology behind the games often caused industry participants to lose sight of the fact that it is often the simplest things that make a game a great game.

In an effort to atone for our sins, we set out to work on game designs that were simple enough to be played and adjudicated in the minds of mere mortal beings, not in the tricked-out silicon chips of their über-PCs.

When you design a board game, you have nowhere to hide: no dramatic sound track, no heart-thumping frame rate, no insanely cool particle system...nothing to distract you from the game's fundamental mechanics. What makes or breaks your game in the world of cardboard is the basic interaction of a simple, but not necessarily simplistic, set of rules with the minds of a few human beings sitting around a table.

As a result, the challenges of designing, publishing, and marketing a board game are of a different nature than any seen in the digital world. For example:

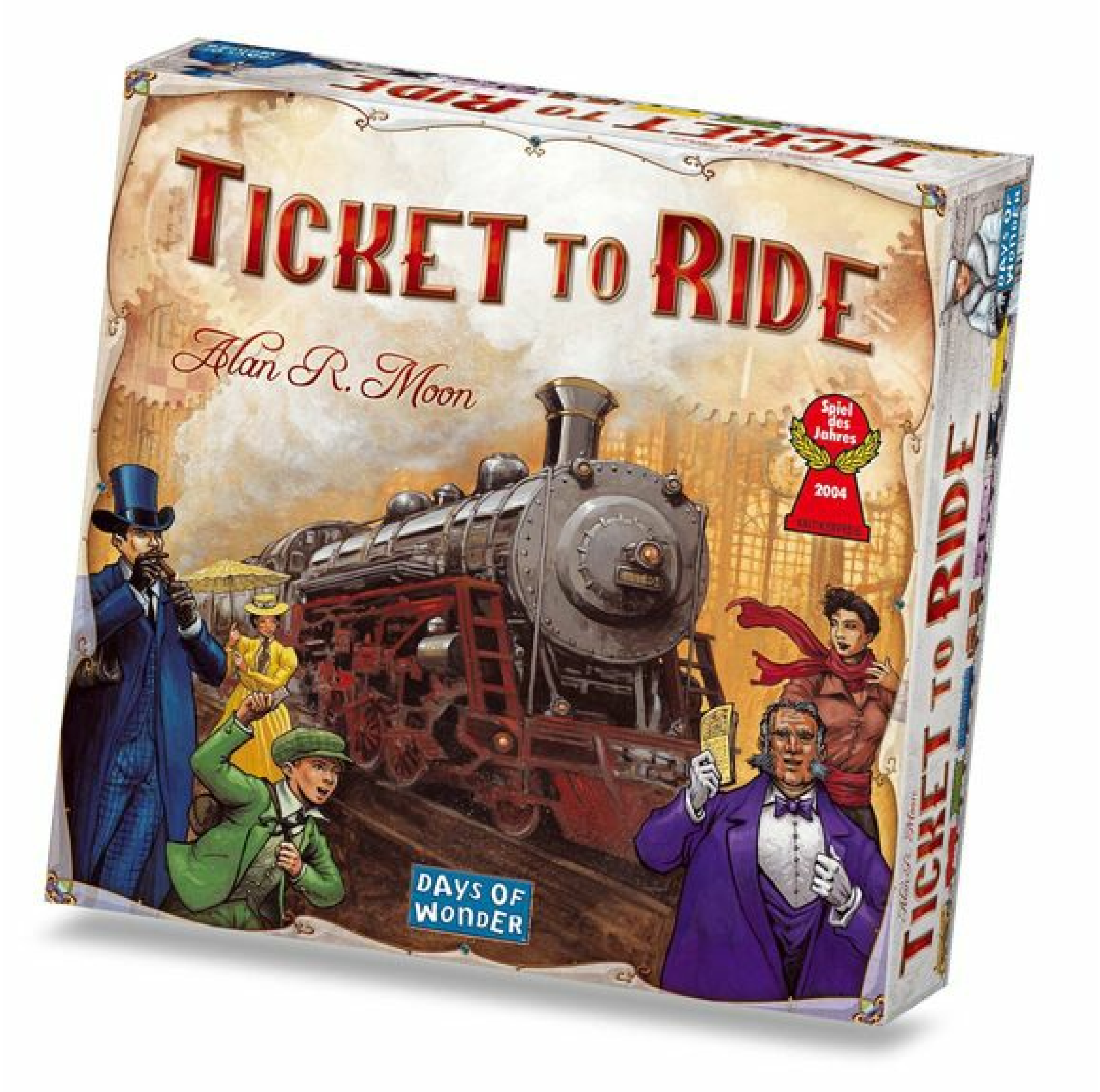
- What is the tiniest set of rules you can get away with, without breaking your game's appeal?

- How tightly can you write your rules, so that there are no ambiguities left in the players' minds?
- Last, but certainly not least, how can you successfully market your game in an industry that is inherently full of frictions, deemed as not newsworthy, and heavily tilted in favor of long-entrenched incumbents?

This last question -- *How can you successfully market a board game, in this day and age?* -- was foremost on our mind when we started the company in May 2002. From the onset, it was clear that we could not outdo or outspend our multinational competitors that had been at the top of their game for the past hundred years. We needed to leverage our experience from the high-tech space and simply build a company that ran faster, cheaper, and smarter than the big guys.

When it came to marketing, that meant leveraging the internet - not just to spread the word, but to help spread the game itself. How else could you teach an entirely new game to the largest possible group of people, in the shortest possible amount of time, at the lowest possible cost, than letting them try their hand online first? As a result, on day one we split our efforts down the middle, with half the company working nonstop to figure out how to publish and distribute amazing game experiences on cardboard, while the other half was busy crafting digital versions of these same games.

Then we got lucky...

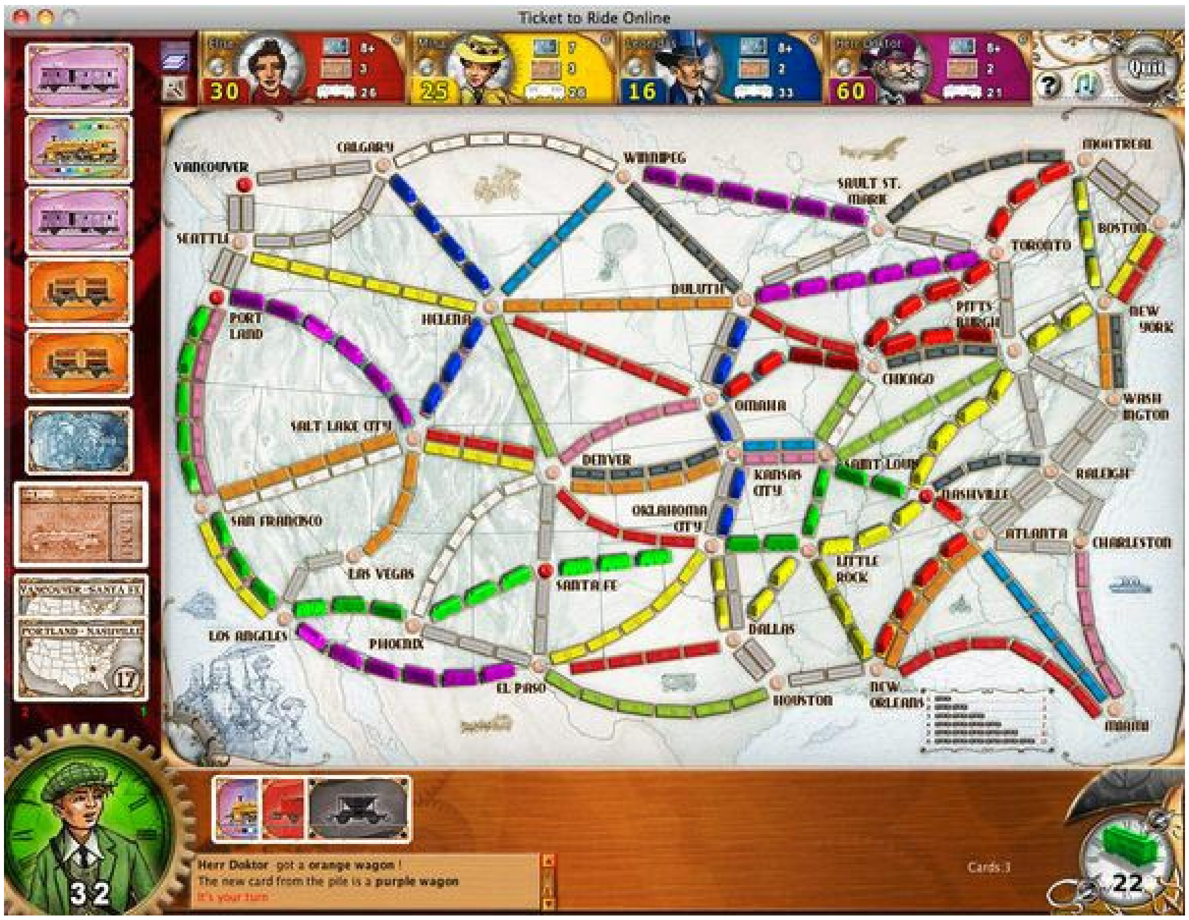


Two short years after starting the company, we published Ticket to Ride, a rail adventure game where up to 5 players compete to claim railway routes throughout North America by collecting and playing matching train cards. In the year it launched, Ticket to Ride won the German *Spiel des Jahres*, the world's most prominent board game award. It was the equivalent on an indie studio winning the Oscar for Best Motion Picture.

Within minutes of winning, we had orders for 50,000 games sent to our cell phones via SMS from eager mass merchants. That year, our sales exceeded $6 million. Our original sales prediction had been $700,000. To say we were shocked -- but happy! -- would be a tremendous understatement.

In 2005, to help further fuel the fire, we released *Ticket to Ride Online*, a browser-based, multi-player, digital adaptation of the game. Then, a funny thing happened...

People who had never before heard of this new Ticket to Ride game began playing... a lot. In some cases, a whole lot At last count, at least one dedicated Ticket to Ride "rail baron" has logged in over 60,000 games of *Ticket to Ride Online*; at an average 10 minutes per game, that's 10,000 hours in the conductor's seat. Many others have played well past the 20,000 games mark.



Herr Doktor got a orange wagon !
The new card from the pile is a purple wagon
It's your turn

Ticket to Ride Online

Fans of the board game had provided the initial critical mass of players crucial to the success of *Ticket to Ride Online* New online players then provided a steady stream of new buyers of the board game; and the cycle repeated, with sales of Ticket to Ride eventually sailing past the 2 million copies mark in 2011. But the best was yet to come...

Coming to iPad

In January 2010, Steve Jobs unveiled the iPad to a rapt audience -- our team included. The screen, at close to 10" diagonal, seemed large enough to display a real board, while the multi-touch interface and slim design hinted at the device's promise: completely transparent and invisible enough to the players to just leave the game itself on the table. What we saw was a true digital cardboard replacement that seemed designed perfectly for board games!

Our first game for the iPad was called *Small World*. Its nature, unlike Ticket to Ride, requires no hidden information, and its slim, elongated 2-player board made it a perfect fit for the newly-announced tablet. Immediately after the iPad announcement, we were coding and testing feverishly on Apple's simulator running off a Mac, tilting the computer's screen skyward between our legs to imagine what the experience would be like sitting facing off each other around an iPad.

We had a prototype ready just days before the official launch of the iPad and, on April 1, 2010, we became the first hobby game publisher to ship a board game for the iPad. Our competitors, Hasbro and Mattel, had offers of their own too, though theirs were released through partnerships with EA and Gameloft, companies that may have had early access to the device.

Though rough around the edges and in need of several updates, the game was well-received, quickly garnering some 50,000 units sold. More importantly, it increased the sales of the board game by over 30 percent within three weeks of the game's introduction. Sales have never dropped off since then, making Small World our second most successful board game behind Ticket to Ride.

So in May 2011, we finally brought our best-selling franchise, Ticket to Ride, to the iPad. We spent the previous 8 months lavishing effort on this version to make sure it'd be our most polished digital adaptation of any board game yet

The gamble quickly paid off. Within three days of launch, we'd already earned enough to pay for all our development fees (both in-house and contractors). Within five months, we'd sold over 180,000 copies of the game and its various maps -- not bad for a $6.99 game.

Then we challenged ourselves: what about designing a version of Ticket to Ride for smartphones, so that even more people could play and enjoy our game?

Completing the iOS Circle

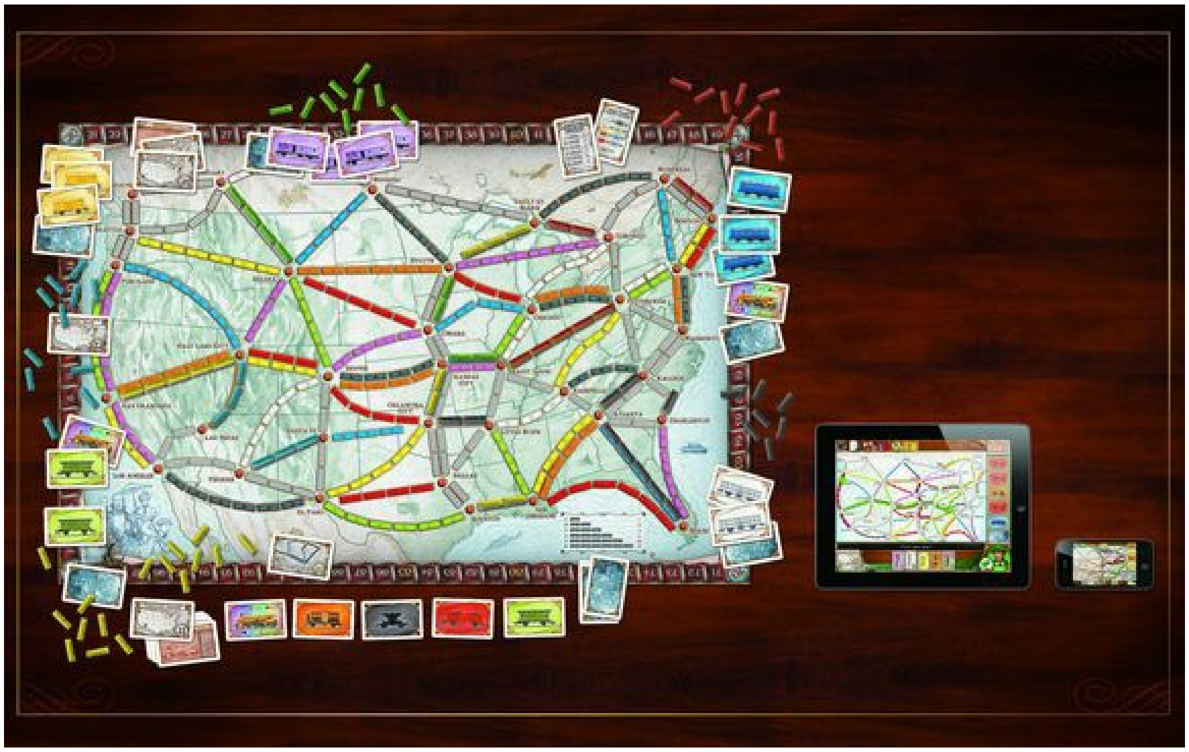
The first targets we had in mind of course, given our earlier successes, were the iPhone and iPod Touch. They were similarly intuitive devices and widely adopted; they also used the same APIs and similar hardware as the iPad, making them an ideal target.

The code name for this project, which eventually became the official name for the game, was *Ticket to Ride Pocket*. The name captured the spirit of our number one goal: to shrink the board game so much that you could carry it in your pocket while still maintaining the core gameplay experience that was critical to Ticket to Ride's success.

For us, this was an unusual project. We were not starting from scratch, as we already had an extremely well-received version of the game for the iPad. The iPad experience boasts a rich and polished user interface packed with features like multiple maps, on-line gaming, in-app purchases, multiple languages, videos, etc.

At first, this made it look like a fairly straight "port" project, where the main challenge would be to squeeze all this in a smaller screen. These kinds of projects are usually easy to deal with, as you already have debugged your algorithms, defined the major UI elements, seen real-life player interactions, etc.

Then we looked at the picture below and quickly realized all a straight port would give us would be a poor man's version of Ticket to Ride for iPad....



- Map of the tabletop board game version: 31 x 20.71 in
- Map of the iPad version: 7.76 x 5.82 in (1024 x 768 pixels @ 132 ppi)
- Map of the iPhone version: 2.94 x 1.96 in (480 x 320 pixels @ 163 ppi / 640 x 960 pixels at 326 ppi for Retina Display)
Click to Enlarge

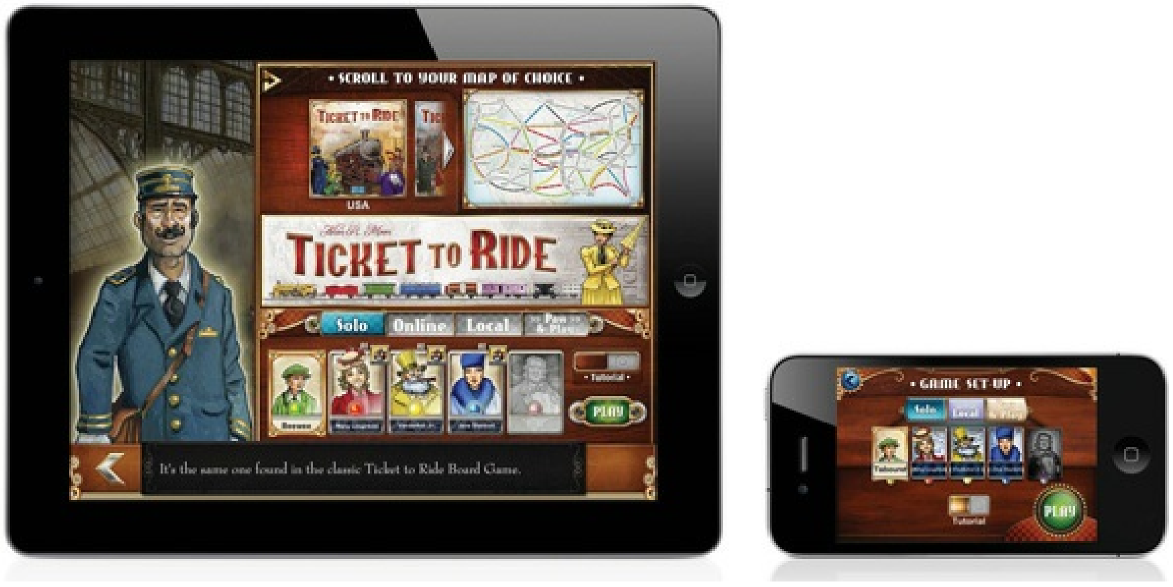
But a "poor man's iPad" is not how iPhone and iPod Touch owners want to feel about experiences on their device. Nor was it what we'd want for ourselves as players. The more we thought about how we'd play Ticket to Ride on our phones, the more we realized how different our behavior with these devices was than with our iPads.

We weren't discovering or even purchasing games the same way; we weren't playing these games in the same environments, nor for the same length of time. Our usage and behavior were completely different animals when we switched devices, and we knew we had to treat development differently if *Ticket to Ride Pocket* would be a success.

What Went Right

1. Rethinking the Design from the Ground Up

So we went back to the drawing board, with one idea in mind: the game would have to fit "spur-of-the-moment, I need a gaming fix" needs. The UI would have to be streamlined, with a game starting in no more than two taps from a single thumb. Keeping features out became as important as keeping some in, and the choices were tough:



The iPad UI vs. the quick-and-easy iPhone one to set up a game.
Click to Enlarge

A long-considered, and we believe ultimately correct decision, was dropping support for live online play. We have tens of thousands of players who log in on our servers to play together every day, across a variety of Macs, PCs and iPads

Unleashing a sudden surge of iPhone users with an inherently different behavior (prone to picking up phone calls, seeing their movie start or coffee served, etc.) among them would risk ruining the experience of everyone.

But the iPhone and iPod Touch are so ubiquitous that we were also presented with a tremendous new opportunity: local, face-to-face play across multiple devices. There are a lot more instances when you sit down with friends, each with an iPhone in hand, than with iPads.

So we decided to code multiplayer gaming over a local Bluetooth or wifi network. And because most iPad users have an iPhone too, we decided to retrofit this new feature into the iPad version, so that an iPad owner could hand his iPhone to a friend and play with her.

And since the user experience would center more around solo gaming, Achievements would be key, too.

Once testing began, it all paid off: we found ourselves playing more impromptu games of Ticket to Ride in a few weeks of testing than we had in years prior. The redesign had been well worth the effort.

2. Adapting to a Smaller Device

Adapting the graphics to a smaller screen. This worked, both visually and in terms of gameplay, even on non-Retina Display devices. We redesigned the large U.S. map, simplifying things, increasing sharpness, etc. We included an automatic zoom-in, so that if the user leaves his finger on the same spot for one second, the map automatically zooms-in.

The fact that people are generally familiar with U.S. geography also helped. Color-blindness was the only issue we weren't able to solve to our satisfaction, unlike on the iPad, where the larger screen size lets us display specific marks on each route.

Thumbs are your friends. On a mobile, the in-game UI must be playable with thumbs only. The left thumb drops cards on a route, while the right one picks new cards. This was the key to keeping a fast pace of play, and critical in a game made entirely of micro-turns.

Retina Display is nice, but it does not give extra space for the UI . The Retina Display introduced with the iPhone 4 displays four times more pixels than the original iPhone. This makes for crisp graphics and text, but does not give more room to display UI elements. We designed all our UI screens at 960x640 pixels but shrunk them to 480x320 for usability testing. Unless you're born with the fingers and the eyes of an elf, doubling your display resolution won't help you navigate a denser UI.

3. Developing a Better AI

With an increased emphasis on solo play, a better AI was needed. The iPad had three different AIs, but a even half-decent player could beat them all easily. To develop this new AI, we brought on the engineer who had developed "Johnny", the challenging AI of our WWII wargame, *Memoir '44 Online*.

His methodology differed greatly from our usual "agile" approach to coding. Instead of releasing internal versions several times a week for all to see progress and give feedback, developing a new AI required an important upfront coding investment before there was anything to play against. This process included developing unit tests, a good debugging environment, benchmarks, etc. So it took a long time before we had code we could really play against -- a source of angst for all involved.

However, time and time again, the investment we made in terms of unit testing upfront paid off tremendously: we were constantly sure that nothing was broken during the final optimization process.

In-Depth: Developing a New AI

Because we opted to keep online play out of the iPhone version, the quality of our AI began to matter more. The then-current AIs weren't challenging enough for many of the advanced players trained on the board game and the online iPad version. A first good decision we made was to anthropomorphize the AI, assigning individual characters to variations of it. Not only did this make the UI more playful, it also challenged the players to figure out what strategy each AI might be using.

Beyond this, a new, stronger AI was also needed. We decided from the onset to limit this AI to the U.S. map, since this was the only map offered on the iPhone, and to focus it on being competitive in two-player games, because our historical data showed this was the most prevalent play configuration.

We then had to decide what strategy this new AI would use. At that stage we did not focus too much on implementation; instead we tried to take a player's perspective. To help us guide our heuristics, we analyzed the online games of our best ranked players, trying to find some common denominator to their games.

It was decided that the core of the strategy of this new AI would be to finish the game in as few turns as possible. Finishing a game quickly has two main consequences: first, your opponent is surprised and probably has not finished all his routes.

Second, to finish fast, you need to use at least 43 trains on a minimum number of routes. Since claiming long routes gives you many more points, this is a very effective way of scoring. We, humans, then played each other and the existing AI trying to implement as best as we could the above strategy. This validated our heuristics, and once this strategy was decided, we stuck to it until the end.

It was now time to implement. It took about two weeks to release a first version that proved to be both too slow and not good enough. Pressure started to mount, since with AI, you can never be sure that you will reach your established goals in a set amount of time.

It turned out that the mediocre quality of play was due to the extreme difficulty of reproducing some obscure but important bugs. As is often the case with AI, most of the time, effort and lines of code should be put in testing code that is not included in the release. In the rush to build a first prototype we had skipped the all important automated test suite and game analysis tools, and were now paying the price.

It took almost a whole month to catch up, during which no intermediate release were available and pressure continued to build.

Eventually, most bugs and algorithm flaws were ironed out, but the game was still too slow for a mobile app. Our speed was averaging six seconds for a single AI query that would sometime peak at 20 seconds on an iPhone 4! And this was per opponent! So it'd be quadrupled in a five-player (4 AI) game. Even worse, on the simulator, where we were averaging 2s / AI / turn, the AI still felt way too slow. It was decided we could not compromise on responsiveness, and thus would have to match the speed of our other AIs, at less than 0.3s / AI / turn.

The main culprit was well known, thanks to the Xcode performance tools; most of the time was spent inside the Objective-C set container. NSSet was not designed to have another NSSet as one of its element -- unless you really are in no hurry, that is! We tried to fix this as best we could using the fact that it is easy to modify built-in objects in Objective-C. It helped, but remained far from sufficient.

In fact, the culprits were many, and included not only the Foundation Objective-C standard library, but also Objective-C itself. The solution was, of course, to use C++ and its über-efficient STL standard library. How long until shipping? Two weeks? Too dangerous for comfort!

At this stage, Objective-C++ came to the rescue. Instead of doing a massive port from Objective-C to C++ -- which would probably take the entire two weeks we had left and would risk not working at all because of all the bugs introduced while porting -- we were able to do incremental steps, literally class by class, each time modifying and running all the automated and manual tests and fixing the manageable number of bugs introduced during the port of each particular class.

It all went well, and at the end of two weeks, we had a working version of the AI that passed all our tests and met all our criteria. We now had a pure C++ AI that performed as fast and well as we had hoped.

---

4. Not Reinventing the Wheel

Even though the feature set differed, we did have the benefit of a prior iPad version. The iPad and iPhone share the same developments Tools (Xcode), programming languages (Objective-C / C++), and APIs (Cocoa Touch, Core Animation), so we could re-use the game engine from the iPad and exclusively focus on the user experience, which was the most challenging part of our iPhone port.

Even for the UI, we re-used base classes of View Controllers and Animations, which were carefully derived to address the specifics of the small screen.

iOS Developers often use middleware libraries for graphics rendering and animations: we did not, and instead entirely based our developments on Apple's built-in frameworks: Cocoa Touch, Core Animation, and AV Foundation Framework.

Why?

- We don't care about being 60 FPS: Ticket to Ride is not an arcade game, but a turn-based one. We could achieve nice transitions and animations thanks to Core Animation framework that fit well with the Views (UIView) from Cocoa Touch.
- Objective-C is very powerful for RAD and UI Development.
- Apple APIs are mostly very concise and easy to use.

5. Iterating Prototypes Quickly

At Days of Wonder, we strongly believe in iterative and agile development. Especially when developing for a multi-touch 3.5-inch screen: it's all about experimenting with new UI ideas and refining them every step of the way. Like any new recipe, you never know how it will taste before mixing your ingredients, even if they are top quality.

Our graphics designer produced static screens in Photoshop and emailed them directly to her iPod Touch in order to have a first idea of how they looked on a real device. When we agreed on the still image, we immediately prototyped the screen with behaviors (triggers on buttons, animations, and more).

For this purpose, the Interface Builder/UIKit duo is the perfect match to test and produce ideas that come to mind. Since every UIView (the UI Element in Cocoa Touch) is based on a Core Animation Layer, it's very easy to animate them with fancy effects. All the animations in- and out-game were done this way.

No direct OpenGL. Pure Core Animation*.*

What Went Wrong

1. Feature Creep

We love working in an "agile" way: delivering quality internal versions on a very regular basis to collect feedback and validate ideas. However, this also means you need to accept important changes when you realize you are in a dead end or need to develop a new feature.

For example, local play was not an idea we had at the project's start. Instead, it came up two months into the project, but immediately seemed so valuable we decided to do move forward with it. We tried to guesstimate the additional work required, and pushed the release date back by three weeks to be safe. However, we had only very limited experience with local networking in iOS.

Local play appeared as a must-have for two reasons. First, we thought it might help the viral/word-of-mouth spread of the game: "I found this great game; pick it up and we'll be able to play together right away!" The second reason was that our sales of Ticket to Ride for iPad were going through the roof. Most iPad users also have an iPhone, so bringing local play on both the iPhone and the iPad games would tie the two games together, and likely help boost the initial sales of *Ticket to Ride Pocket*.

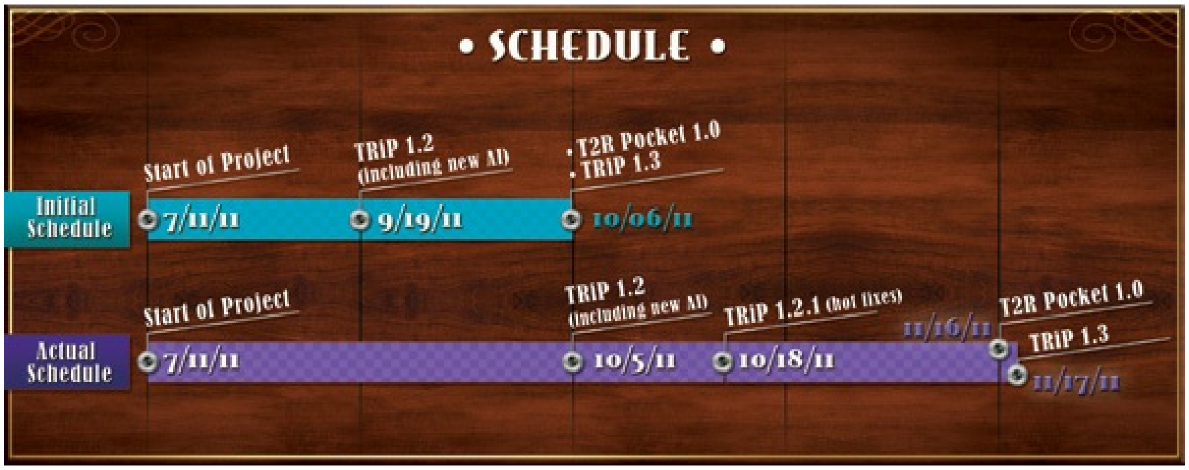
However, when combined with the Context Switching issues described below, this feature creep ended up making us walk on a very tight rope... with no safety net.

2. Context Switching

Adding local play to the feature set also meant releasing a new iPad version of the game concurrently. Surely, owners of the iPad version of Ticket to Ride would be the very first ones to want to try local play. Besides, this would also allow us to test our AI in the real world.

So the decision was made to add not just one, but two iPad releases: one in September, that would include a first version of our new AI and support for iOS 5, and one in November at the same time as the iPhone version, with the Local Play feature -- and support for Chinese and Japanese languages on top of it!

This made our lives much more complicated. Instead of focusing exclusively on the iPhone, we had to switch back and forth between projects -- even if they were very similar and shared a lot of code in common.



We underestimated the overhead involved and time needed to test each release, paying the price by working 16-hour days, 7 days a week, during the last month of the project. Going into "crunch mode" at the end of a project is not uncommon in this industry -- but this should NOT mean it's normal or acceptable, nor should we consider it a casualty of the job. Agile development is helping us greatly reduce this phenomenon, but we obviously still have some progress to make.

We conducted postmortem discussions after the project, and decided to add a few "rules" when doing the initial planning of a new project, like adding some incompressible "exploration phases" early on when new technologies are involved (very similar to Scrum's "spikes"), etc.

---

3. The 20 MB App Size Limit

Another big hurdle was the size of the app. We desperately wanted the app to remain under 20 MB, so that people could buy and download it over 3G. This caused all kind of headaches.

At mid-development, the whole package was 50MB and half of the features and the UI were not implemented yet!

We started by doing the obvious tricks, reducing our graphic assets to 8-bit PNGs while preserving good quality and alpha channels as much as we could. We used ImageAlpha, an app that helped us drastically reduce our PNG images.

However, several important assets arrived late in the development: the video tutorial and the localized assets. These are not things you can finish early on, since they both need a final user interface.

The tutorial video had to be embedded in the app, because you do not necessarily have a high-speed connection when playing the game for the first time. So that ruled out the YouTube streaming solution we took for the iPad. Worse, we needed audio in 3 different languages: English, French and German.

It was out of question to have three videos, so we had to include the three audio channels in the same video, and recode the video player instead of using the iOS off-the-shelf one, in order to select the right audio channel depending on the language. We spent a lot of time fine tuning compression quality, etc. At the end, the video took 5.2 Mb, or more than 25 percent of our overall allowed size!

The UI is quite polished, with lots of bitmaps. Each item that needs to be localized triples the size of the UI assets. Lesson learned: next time, we will use stand-in assets, duplicating the English ones right away.

So in the final days, we were just below the 20 MB: around 19.5 MB. We thought we'd made it. Big mistake!

Reading some articles, we discovered with horror that Apple's DRM system adds an unpredictable amount of data to your app! After some more digging, we found a heuristic to evaluate the overhead. It looks like the code is encrypted through the Apple DRM, and that this encryption does not compress as well as regular code.

So we stripped some code. Fortunately, there was some left-over from the iPad code base that we could get rid of, but this was risky business. We had to be incredibly careful because, unlike C++, Objective C does not tell you easily if you are removing code that will actually be used (you have to be really careful about compiler warnings).

As you can imagine, our testing people were not happy to hear they had to revisit the whole app and make sure everything was still fine after such surgery. So we treaded carefully, and were able to save the few megs we needed. Here again, lesson learned: when unsure of your app size, strip unneeded code at the start of your development, not at the end!

We submitted a 16.7 Mb to Apple that turned to be… a 16.7 Mb on the App Store. Life can be so cruel sometimes...

4. Switching Development Tools

We used Xcode 3.2 to release all prior versions of Ticket to Ride for iPad. We jumped on to Xcode 4 for an intermediate version (TRiP 1.2.1) as soon as Apple officially released it. It turned out to be slow and unstable -- all of which contributed to make it a painful experience. Xcode 4 is promising and, no doubt, things will improve. We just switched too early and paid the price.

Lesson learned: do not switch to a brand new tool in mid-development, especially when your schedule is tight.

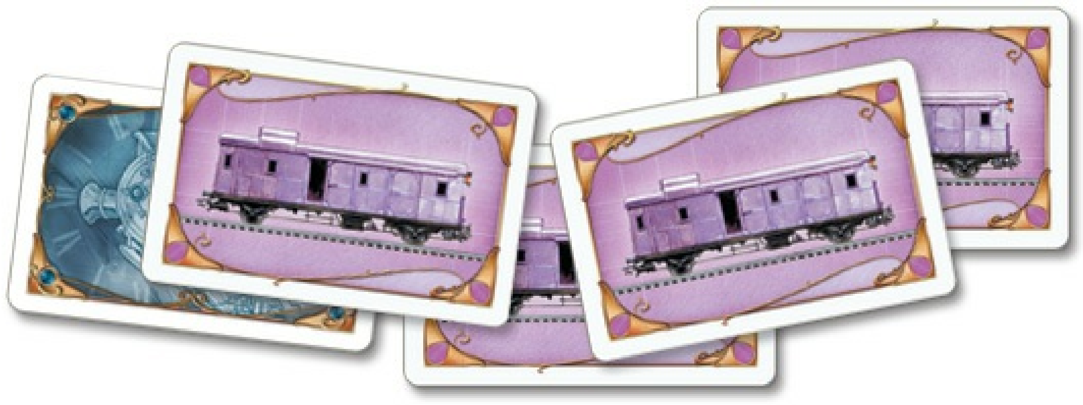
Unfortunately, you don't always have much choice when dealing with iOS development. Everybody agrees that you need to keep you development tools up-to-date, but the question is: when?

- When Apple announces that it is the official version? You run the risk of stumbling on new bugs because the tools were never really used in real conditions.

- Between two projects, when you have some peace? Surely, but how often does this happen? Short development cycles reduce the opportunities for "peaceful" breaks.
- Only after collecting some feed-back from other (early-adopter) developers?
- When Apple stops supporting the old version? But then you are always late learning and using the new features of the OS.

More generally speaking, these issues help illustrate the old debate between two competing visions of software development: A "Microsoft-like" vision, which favors long backward compatibility (lots of Windows 95 apps still run on Windows 7), and offers development tools stability but less innovation versus a more Apple-like "move forward or die" philosophy, which keeps pushing developers (and users) toward innovation, even if this requires them to jettison older products (68k CPUs, to PPC, to Intel, Mac OS 9, to Carbon, to Cocoa, MPW, to Code Warrior, to Project Builder, to Xcode 3, to Xcode 4, etc.).

On top of this, the behavior of some iOS APIs sometimes change significantly and in an undocumented manner from one version to another. Until now, Apple's "look forward" approach has worked, but the risk of poor backward compatibility and device fragmentation may eventually crop up as new generations of iOS devices keep coming to market.



5. Going for Same-Day Releases

Our final challenge was to simultaneously release the new iPhone app and the iPad upgrade on the same day. There, we were not so lucky.

After the usual week of waiting anxiously for the apps to go in review, our iPhone app was rejected because Apple thought we might be using some private APIs. It seems they run an automatic code analyzer that spots the names of private API functions. Our alert was a false positive: by pure bad chance, we'd named a couple of functions the same way as Apple. So we had to recompile and re-submit **both** apps in a rush. *Ticket to Ride Pocket* was approved right away, because its review had already been underway. But the iPad update hadn't gone into review yet, and took a bit longer.

Ultimately, Apple came through though, in a big way: the iPad update was approved 24 hours after the release of the iPhone game, and our customers were delighted. But we went through a few sleepless nights of waiting.

Conclusion

Ticket to Ride Pocket was our first iPhone project. It's off to a great start -- the app was picked as Game of the Week throughout Europe within a day of launch, and the sales got off to a strong start with 50,000 copies of the game sold in the first five days.

We learned a lot developing it and are confident we'll make new mistakes, not the same ones, next time around.

Data Box

Developer: Days of Wonder

Publisher: Days of Wonder

Release Date: 11/16/11

Platforms: iPhone / iPod Touch

Number of Developers: 2 coders, 1 computer graphics artist + contractors (music, video, original artwork)

Length of Development: 5 months

Budget: $100K+

Lines of Code: 65000 (Objective C / C++)

Development Tools: Xcode 4.2, Interface Builder, Git, Photoshop.

Return to the full version of this article