AuroraBound Postmortem and Sales Figures - A Modest Part-Time Success

gamasutra.com/blogs/DanielOByrne/20170814/303637/AuroraBound_Postmortem_and_Sales_Figures__A_Modest_PartTime_Success.php

Blogs

by Daniel O'Byrne on 08/14/17 10:07:00 am Featured Post

Original posted on the Final Game Studio blog

It’s been 3 months since AuroraBound launched on mobile devices and so far it has done considerably better than I was expecting (though it's certainly not the kind of irrational break out financial success we all secretly hope for). While procrastinating on my next game, I decided to look over some of my notes and write up a postmortem of how everything went. I’ll try to stick to the standard ‘what went right and what went wrong’ format, but I’ll also throw in a section at the end with the game’s sales performance and analytics to date.

What Went Right:

1) Scope

If you read my previous blog posts, you’ll know that I developed AuroraBound in my spare time and, in what can only be described as a game development miracle, it kept pretty much to the original estimates of 6 months for development. To do this, the core game design and mechanics were purposely kept as simple as possible and any ‘extra’ features that weren’t absolutely needed were brutally cut.

When pre-production started, the rough goal was actually to make a space based RPG with a ship building & combat system loosely inspired by the Galaxy Trucker board game. The first thing prototyped was the ship building mechanic however - where you can only match up square component based on their connector types. I realized I was having fun just matching colored squares in a little grid and decided to cut everything else from the concept after a day or two and just focus on that. Not the easiest decision I have ever made, but since the main goal was to release a finished game, it was the right choice.

Whenever a section of the game was being designed, how long it was going to take was always a major design consideration. For example, once you enter a puzzle you need to complete it to progress, you can’t exit back out. You also can’t replay previously completed puzzles and there aren’t multiple save files. While these choices appealed to me for the specific game experience they provided, they also saved a lot of development time by allowing me to cut corners in level saving code and in ui design.

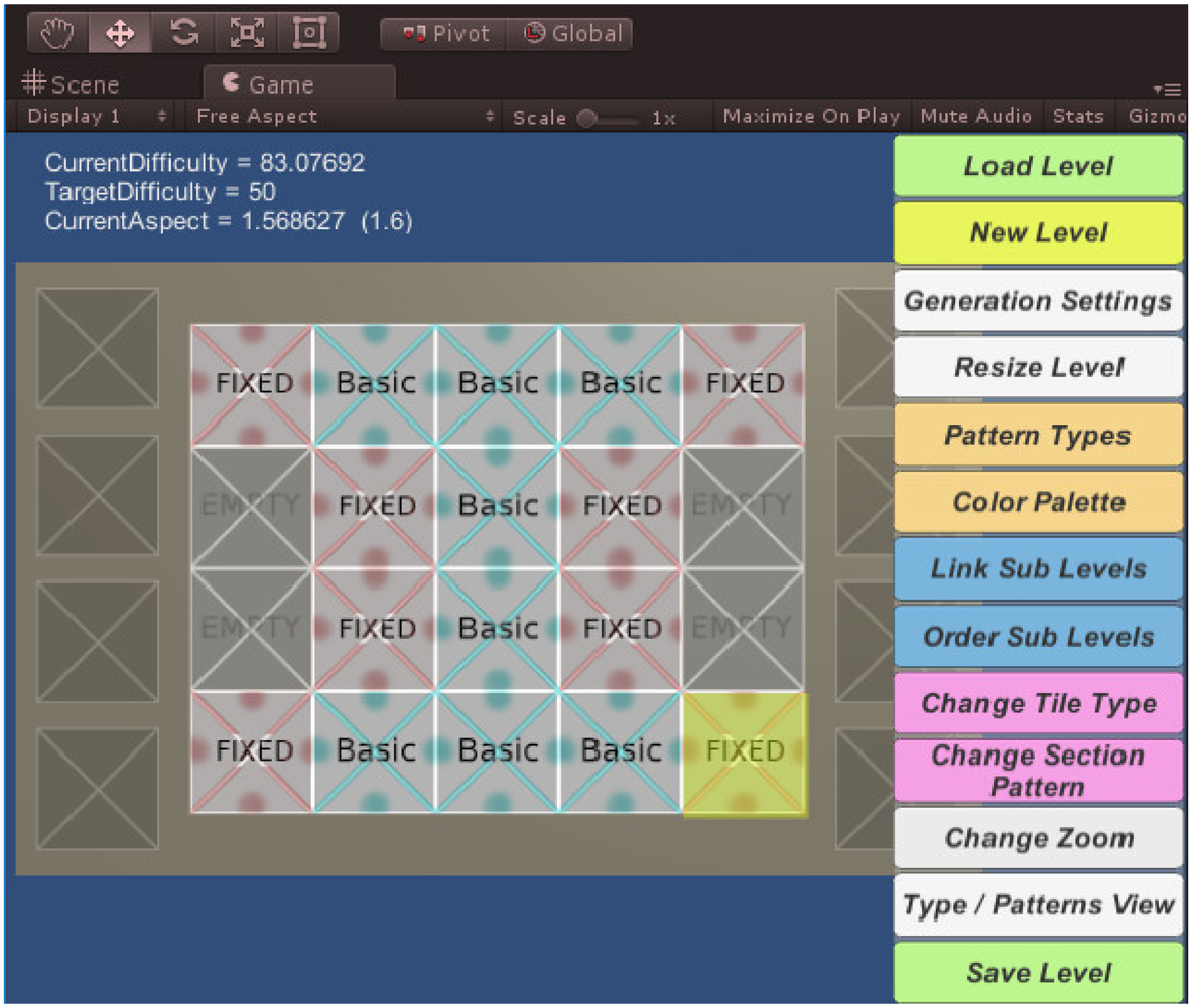
Another example of cutting corners time wise was the trailer. Once a solver was made, a very simple ‘auto-play’ mode was also added, which would automatically play through a list of pre-defined levels. This took maybe 15 minutes to set up, and after another hour of tweaking to get the timing right, and display the game's title when it finished, the result was the trailer for the game. I just ran that, recorded the screen and no further editing was required. It might not have been the best trailer in the world, but I was happy enough with it and the whole process took less than 2 hours. As an added bonus, it was simple enough to then alter the settings to auto-solve the levels in two distinct passes and take screenshots in the middle. I left it running, went off to get some food and when I came back I just picked my favourite 8 to be the promotional screenshots.

I also cut a lot of potential features that I knew the game could benefit from that I just didn’t want to work on, such as social media integration, leaderboards and achievements. Keeping my motivation up while working only on weekends and evenings was challenging enough - doing so while spending that time implementing 3rd party SDKs or more advanced save systems would have been a step too far.

2) Level Editor & Generator

As soon as the game concept solidified into a puzzle game, it was obvious that some sort of level editor would be required. Since the puzzles were relatively short, a lot of them would be needed. AuroraBound ended up shipping with around 200 handmade puzzles, and once a player completed them, they would seamlessly switch over to completing procedurally generated puzzles.

I don't actually have a lot of experience creating level editors - most of the previous commercial work I have done has involved creating products with several once-of, single level games in them. I decided to create the level editor inside of Unity, and that worked out pretty well - it's just a separate scene in the main game which allows me to load, save, display and edit level files (all of the levels are just simple XML). A bunch of buttons along the side were linked to editor scripts that perform the various operations needed.


```
CurrentDifficulty = 83.07692
TargetDifficulty = 50
CurrentAspect = 1.568627  (1.6)
```

One of my shameful little secrets as a programmer is that I love quickly writing terrible, throw away code and, since the editor was only ever for my use, I was able to throw the first version of it together in just a few hours. It looked terrible, and it ran even worse - for most of the project's development it had a rather impressive memory leak of ~0.5mb a second that would eventually cause it to crash if I didn't restart it… But for what I needed, it was perfect. Most of the editor actually ended up being rewritten several times (with equally appalling code) as the underlying structure of the puzzles substantially changed and it needed to be 'updated' to match.

Even with the editor, creating new puzzles manually was very time consuming. To fix this I ended up creating a level generator that could create new levels based off a small set of parameters such as target difficulty, color set, pattern set and distribution of rotatable tiles to movable tiles. When designing the generator, about 30 puzzles were manually created in the editor and then used to work out desirable puzzle traits (like symmetry and repeated patterns in the final solution). These traits were then programmed into the generator.

Once the generator was working relatively well, it was hooked up to the level editor and used to create the starting point for each of the ‘hand made’ levels shipped with the game. I would quickly use the generator to cycle through dozens of possibilities for a given difficulty and then, once it gave me one I liked, the editor was used to tweak and refine it a bit to make it more visually appealing. This was necessary as it was important to carefully craft the initial puzzle difficulty curve and because while the editor did a good job at creating large levels, it struggled to create very good small levels at lower difficulties.

In the end ~1500 players have already completed the first 200 hand made puzzles (~10% of players who have downloaded the game). Some players are far beyond that however, with ~60 players having completed over 500 puzzles and with one player now having reached world 115, meaning they have completed around 1000 puzzles. Keep in mind, the difficulty of the puzzles regularly increases, so players who reach world 100 could be facing puzzles like this:

It’s also worth noting that the majority of the IAP revenue is generated in the more complex later levels, after the hand made levels have already run out.

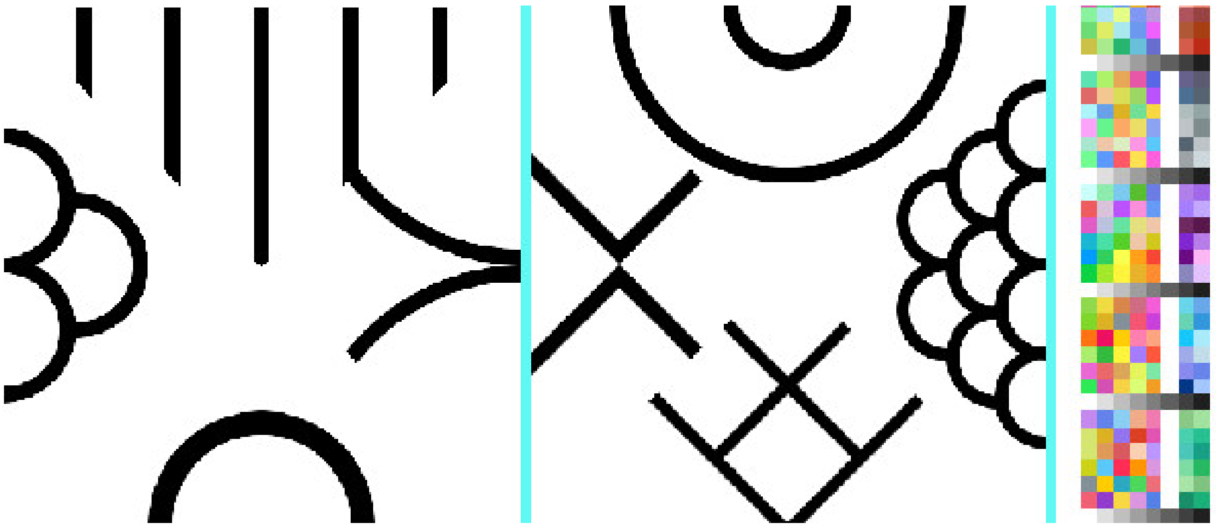
3) Art Style

I am not a talented artist - I have in fact seen my attempts at programmer art literally make some of the artists I work with during my day job cringe. Given this, and the fact that there was no budget for external art, I needed to find a style that I could not only produce, but do so relatively quickly. The style also need to be simple enough that the the app size could be kept as low as possible while having thousands of distinct levels - It needed to be under 100mb for the mobile network download limitation (and because splitting obb files in android is a pain), and ideally much smaller than that for countries with poorer internet connections. In the end, the app was <30mb in total.

It took a fair bit of experimenting and time, but the final style turned out relatively well. The game’s visuals are primarily achieved through the use of hand made triangular patterns and simple color palettes. ~100 black and white triangular patterns were created to make up the puzzle pieces and the best 34 were kept in the final game. Many of the patterns share the same outer edge connector, as this lead to more interesting and visually appealing incomplete puzzles.

36 unique color palettes were created, each of which contain 7 colors - 5 colors for the puzzle pieces in a given level and then a primary and secondary background color. It proved extremely challenging to come up with that many visually distinct (and appealing) color

palettes, especially since pieces are darkened when incorrectly placed, and the process took a lot of trial and error, but it was worth the time in the end. These patterns were combined into a single small image, with each color taking up a single pixel, and then sampled at run time to color a level.



When a ‘world’ (a collection of puzzles) is generated, it is given its own color palette and set of 5 patterns pseudo randomly - with the main restriction being that the patterns and background colors chosen not having been used in the previous two worlds. This gives every world a distinct feel, and with a pool of 45 patterns and 36 palettes, players almost never see the same combination a second time.

What went wrong

1) Procedural Audio

I have even less musical ability than artistic talent, and since there was no sound budget, I was left to search through the various online collections of royalty free music and sound effects. The goal was to find or create something soothing and relaxing, but since there were going to be an infinite amount of puzzles, there was a very real risk of whatever was chosen eventually sounding repetitive. There were a couple of free music tracks which seemed to provide the desired atmosphere, but filling the app with enough to prevent the music getting repetitive would have bloated it’s size.

I decided instead to try and write a simple musical generator that would procedurally create and play simple melodies, unique to each puzzle and world. After all, the target wasn’t an awe inspiring John William’s score, just some simple ambient background noise, and this approach should have avoided the perceived repetition problem. I found free samples of individual notes played on a variety of orchestra instruments and set about creating a very naive music generator.

The basic idea was to get a collection of ~6 notes for a wide variety of orchestra instruments and then set up some basic parameters for how that instrument to play. A new note every half second for example and there might be some logic for how notes should be selected - randomly or in some pre-defined ascending or descending sequence perhaps. When a player entered a new level, a pseudo-random sub-set of instruments would be chosen and one of them would start playing. As you progressed through the level, more instruments would be added (either simultaneously, interspersed, or replacing the last instrument) to help give the feeling of change or progress.

This plan ran into problems almost immediately, primarily centered around the fact that I know nothing about music theory - I can't even read sheet music. I tried looking up similar approaches and the background theory I would need but I either couldn't find what I was after or what I could find was too in-depth for me to consume in the time I had. One of the first issues I had was that certain notes apparently just clashed and didn't sound right when played together - a fact that I didn't know about and had problem hearing on my own when it was happening. The note samples I had all came from a variety of different sources and as such were different qualities - requiring a lot of time spent in Audacity trying to normalise, amplify or reduce noise. A lot of time was also lost culling instruments that just didn't give an appropriate sound (an organ for example, didn't really seem to provide the relaxed atmosphere being aimed for).

After about two solid weeks of making and iterating over the system, I reached a point where I just had to accept defeat, cut my losses and move on with other aspects of the project. The system was simplified a lot and set up to only play one instrument at a time while the overall volume of the instruments was reduced and the volume of the ambient pad loops I was playing them over was increased. The end result sounded okay, it even earned a few complements in reviews, but it never achieved the originally desired effect. I am convinced that an approach similar to the one I attempted could work for games like this, but I eventually had to accept that I just didn't have the required time or aptitude to implement it.

2) Early Solver

The main form of monetisation in AuroraBound is through players purchasing extra 'hints' if they get stuck on a particular puzzle and this mechanic obviously required that the game be able to solve its own puzzles. Very early on in development, maybe a month in, I spent a few days creating a solver that, at the time, I was extremely proud of - it was probably the cleanest code I had written in years. Since this was a feature players would effectively be charged to use, a lot of effort went into making it as 'efficient' for them as possible. A single hint would effect a single tile, so it was designed to look over the current state of the board and programmatically find the shortest path to a correct solution from where the player was now by removing as few incorrectly placed pieces as possible.

When it was done, it worked like a charm - always finding the best possible solution for the player… It just had one, tiny little (glaringly obvious in retrospect!) flaw… You see, when it was created all of the levels were hand-made and very small - but the puzzles would need to get a lot larger to provide an interesting challenge for most players. When tested later on with mid-sized levels the solver failed. Well, it technically still worked and found a perfect solution - it just took about half an hour to get there on a development desktop… Because of the multiple levels of recursion in the solver, each puzzle piece added resulted in a more than exponential increase in time taken to find a solution.

After a day of stubborn optimisation and swearing, I eventually accepted that my fundamental solver approach was just plain flawed and ripped it out in a mixture of anger and professional embarrassment. It was replaced with a simpler system that generates solver data when a level is first generated, embeds that data into each of the 'slots' on the puzzle board and then uses that to quickly 'solve' levels when needed. In practice, this works well (except for an embarrassing bug the game shipped with that has now been fixed) and provides almost the same experience for the user as the original solver, but it means that there is only one 'correct' solution to a puzzle when using the hint system.

3) Time Management

When work started on AuroraBound, I was determined to finish and ship the game and not just add it to my huge list of unfinished prototypes and side projects. To do this, I adopted a hard and fast rule of working on it every day for 1-2 hours (more on weekends). I generally didn't allow myself any exceptions to this unless something unavoidable came up. I also never allowed myself to work for more than 2.5 hours on a given day, no matter how well or poorly things were going - I don’t crunch to finish projects, ever.

This gave a clear amount of development time for the project, since the target timeline was six months, and it was possible to design, estimate and cut features to hit that target. At first, the system seemed to work very well, but about halfway through, I started noticing issues - my attention and motivation were lagging and the temptation to ‘just take today off’ was getting stronger. It wasn’t due to the work being done though - individual features were being finished every day or two and I was regularly swapping between different types of work (and if I really didn’t want to work on something, I just cut it). The simple fact was, I was burning out.

While two hours a day may not seem like much on its own, put in context it was effectively taking up the majority of what had previously been time I spent relaxing and playing games. On weekdays I get up early and leave for work at around 6.30am and then get home at around 5.pm. Exercise and a shower takes me to about 7 and then dinner might take me to 7.30 or 8.00 depending on whether it’s my girlfriend or me making it. If I add two hours of dev work in here, it’s basically time for bed and you will notice a distinct lack of fun and relaxation in that schedule.

Working like this was fine for a brief stint at the start of the project to get things going, but I should have cut back to 3-4 days a week as soon as I noticed the problem. Instead I powered through and ended up basically crashing as soon as the project shipped - post release plans for new features and to finish off and release the desktop version were put on indefinite hold. I misjudged the toll the lack of time off would take on me, and it’s not a mistake I’ll be making again. I’m starting to make some progress on my next game, but this time around I am making sure to keep some days free for the finer things in life (i.e. playing computer games instead of just making them)

4) Good, But Not Great

I feel confident enough at this point to say that AuroraBound is a good game, and a lot of players seem to really like it - but it’s not a great game. It’s not exceptional or different in any major way and it has no major, unique hook to attract press or streamers. This made it very difficult to get the attention of the press and major platform holders or for it to stand out noticeably amongst other puzzle games.

I followed all of the usual advice for contacting press and quite a few of them actually got back to me (though a fair few of those were with offers to feature the game for money which I politely turned down). While most outlets contacted never replied, one or two went so far as to let me know they had tried the game but wouldn’t be reviewing it and why (which I appreciated immensely by the way!) A few outlets did end up reviewing the game however, and were generally quite positive, including the Japanese site Isuta, which (through google translate) provided my favorite quote about the game:

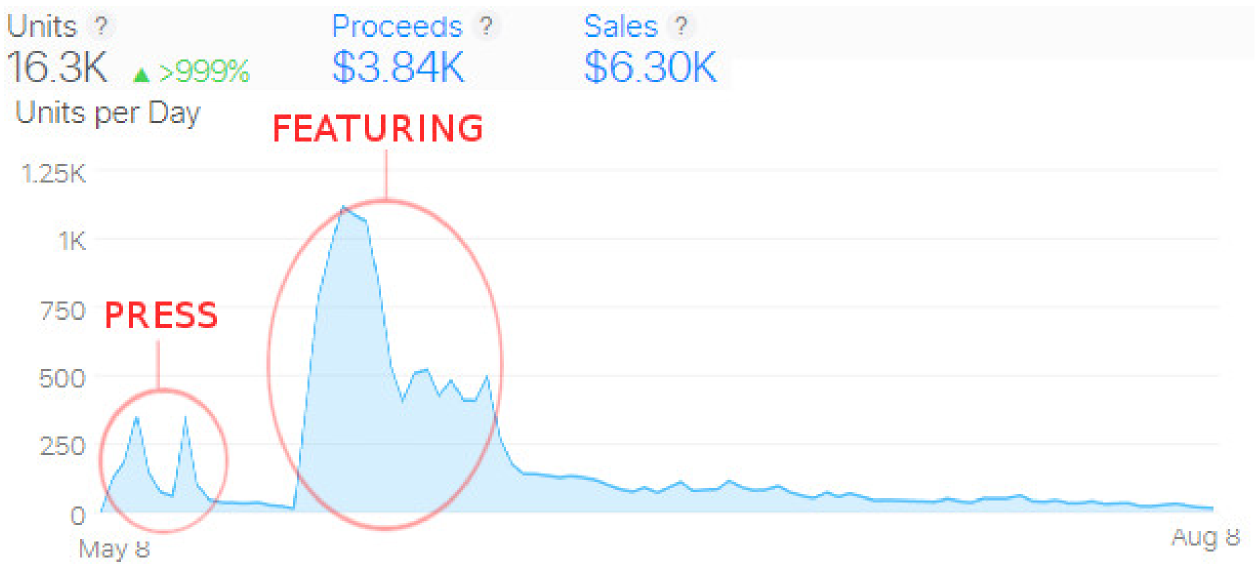
> “In a relaxed atmosphere, it is a rice cake that combines addiction and healing that you want to play forever ♪”

After the reception AuroraBound has received, I have decided to spend a bit longer on my next game and make it a bit more immediately impressive and different in the hopes of making it easier for it to get attention and stand out amongst the crowd.

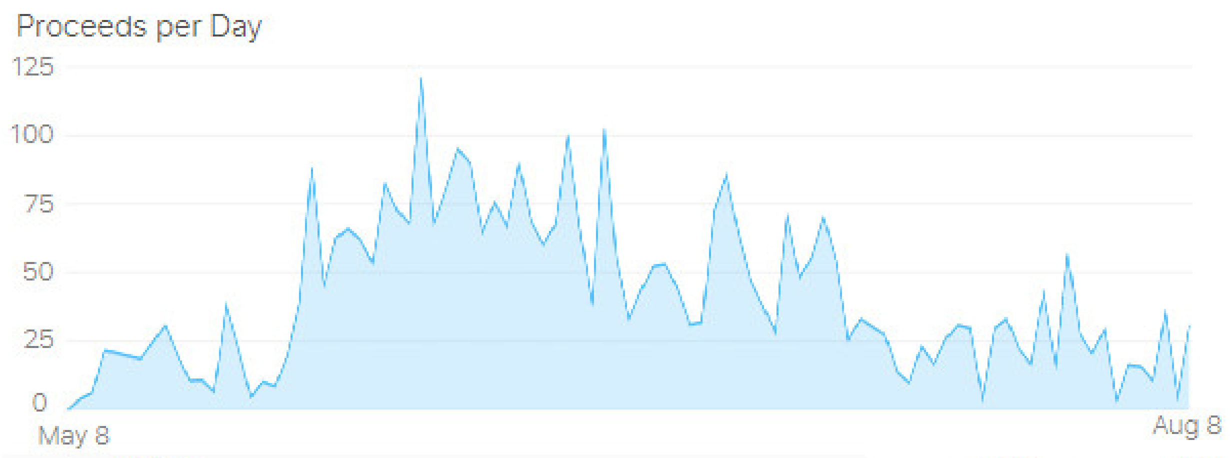
How Did It Do?

I’ll break from the usual right and wrong postmortem format here to share some sales figures and analytics data for AuroraBound's first 3 months.

The game has around 15,500 unique users, the vast majority of which (13,000) are on iOS, with ~1700 on Googleplay. Half of those GooglePlay users are as a result of an AdWords ad campaign I ran as an experiment though. It has only 22 downloads on the amazon app store and 330 on Windows phone (the result of a positive article on Windows Central). It received a small amount of featuring on the iOS app store (it was in the new section of the puzzle and board game categories in various countries) which helped immensely and probably accounts for the majority of those downloads, as you can see below:



In the first three months since its release, AuroraBound has recorded sales of $6,434 (which means a revenue of almost $4000 after the app stores take their cut), considerably more than I expected it to. As my last blog post explained, the game cost <$1500 to make and release (not including my precious time), so I am extremely happy with that. Almost all of that is coming from iOS at the moment, with GooglePlay sales amounting to only $134. You can see a breakdown of iOS sales over time below, and you will note it doesn’t directly follow downloads as only a small subset of users ever pay anything and they generally only do so after spending quite some time playing the game. It’s also worth mentioning that around two thirds of players come from non-english speaking countries (primarily China), so the money spent in localisation was well worth it.



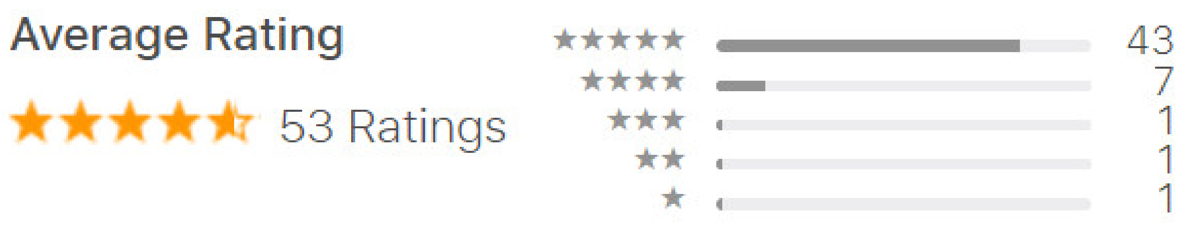
	Name	Units	USD
1	Medium Hint Pack (125) (AuroraBound - Pattern Pu...	482	$1.37K
2	small hint pack (25) (AuroraBound - Pattern Puzzles)	1.88K	$1.33K
3	Huge Hint Pack (2020) (AuroraBound - Pattern Puzz...	70	$642
4	Large Hint Pack (505) (AuroraBound - Pattern Puzzl...	45	$501
5	tip_creator_2 (AuroraBound - Pattern Puzzles)	4	$7.24
6	tip_creator_1 (AuroraBound - Pattern Puzzles)	2	$2.91

There are 372 unique paying players on iOS so far (a conversion rate of 2.7%), meaning the average revenue per paying user is $10.32, but the average revenue per user (including non-paying) is only $0.28. This is important, because it basically rules out using paid installs to increase revenue - the cost to buy a new user is generally between $1 and $4 at the moment.

You may have noticed that while the game has only 372 unique paying users, it has sold 2,483 IAPs - most users who make a purchase end up making several (in a few cases over a hundred). 72 users have spent over $20 in the game, while another 111 have spent between $5 and $20. The most popular IAP by far is the 25 hint pack for 0.99 cent, with some users purchasing it hundreds of times despite it being worse value that the other hint packs (the 2020 hint pack for example is $19.99, and so would give 4 times the amount of hints per dollar as the 25 hint pack) I'm not sure if this is due to me not making the value proposition clear enough or some users only wanting to buy enough hints to get them through the current puzzle.

Unsurprisingly the least popular IAP is the 'Tip the Developer' gratuities, with 6 purchases in total. It was added it as an experiment in case there were users who wanted to support the app, but didn't want to buy any hints. Despite only generating $10 in revenue, I am glad I included it as a few of the reviews reacted positively to it being there as an option. Besides, with other In-App purchases already hooked up, $10 pretty much covers the dev and localisation cost of adding the gratuity option as well.

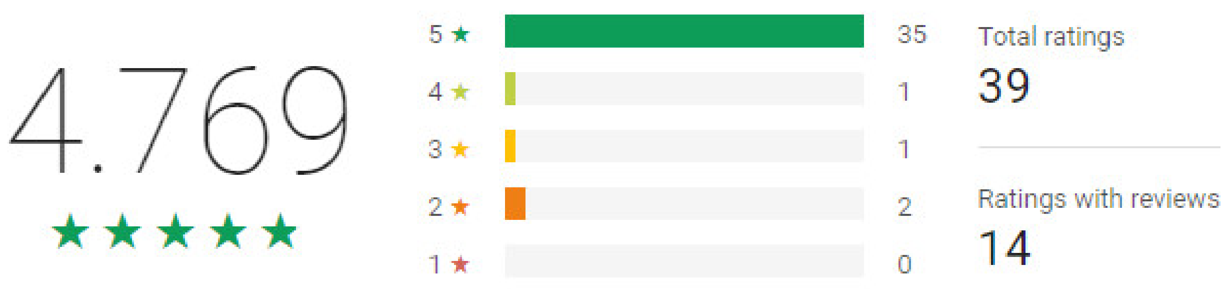
While the download numbers have been relatively small, they are better than many other similar apps and the game has generally been getting very good user reviews, averaging about 4.7 stars on both iOS and GooglePlay:



Average Rating

★★★★☆ 53 Ratings

★★★★★	43
★★★★	7
★★★	1
★★	1
★	1



4.769

★★★★★

5 ★	35
4 ★	1
3 ★	1
2 ★	2
1 ★	0

Total ratings
39

Ratings with reviews
14

Overall, I'm pretty happy with how AuroraBound turned out. I'll be releasing another update in the next few weeks fixing the last remaining known bug, and maybe adding a few more levels, and at some point in the next few months I hope to get back to the desktop version and release that on Steam and the Mac App Store. Work has already started on my next game and it's going relatively well (if a bit more slowly than I would have liked). It should be ready for release early next year and I have tried to integrate the lessons I learned from AuroraBound into its design.

Related Jobs

Galvanic Games, Inc — Seattle, Washington, United States
[06.22.18]
Multiplayer Game Engineer
Armature Studio — AUSTIN, Texas, United States
[06.22.18]
SENIOR GAMEPLAY PROGRAMMER
Square Enix Co., Ltd. — Tokyo, Japan
[06.21.18]
Experienced Game Developer
innogames — Hamburg, Germany
[06.21.18]
QA Engineer/ Software Engineer in Test
[View All Jobs]