Postmortem: Appy Entertainment's Animal Legends

PRINT

By Rory McGuire

Animal Legends is the newest game from Appy Entertainment -- an independent developer out of Carlsbad, California. We're the developers behind *Trucks & Skulls NITRO*, *FaceFighter Ultimate*, and *SpellCraft: School of Magic*. Our motto is, "Deadly Serious About Stupid Fun."

Appy's games have been downloaded more than 22 million times and Apple has featured all our titles. With *Animal Legends*, we wanted to build on 2011's *SpellCraft* to create a game where players can command archetypal fantasy heroes on dangerous quests for loot and glory -- while still fitting Appy's well-known goal of making fast-playing, humorous games with broad appeal.

With *SpellCraft*, we added an asymmetric multiplayer mode that taught us a lot about developing multiplayer games on iOS. *Animal Legends*, with its asymmetric "borrowing of heroes" was a natural next step.

The development of *Animal Legends* lasted roughly seven months. The game was built by seven internal developers and a few external contractors (such as Quinn Dunki of One Girl, One Laptop Productions.) We launched in November 2012, worldwide. *Animal Legends* was our second free-to-play game.

What Went Right

1. Extensive iteration in mockup stage

The App Store has an unbelievable number of games and apps -- but buried within all that software are apps that *can help you make even more apps!*

AppCooker was our secret weapon in *Animal Legends* and it was used extensively in the early mockups of the game. For an iPad app, AppCooker is on the expensive side, but one very powerful tool is worth every penny: an iPad-native prototyping platform that allows developers to create flow between UI screens.

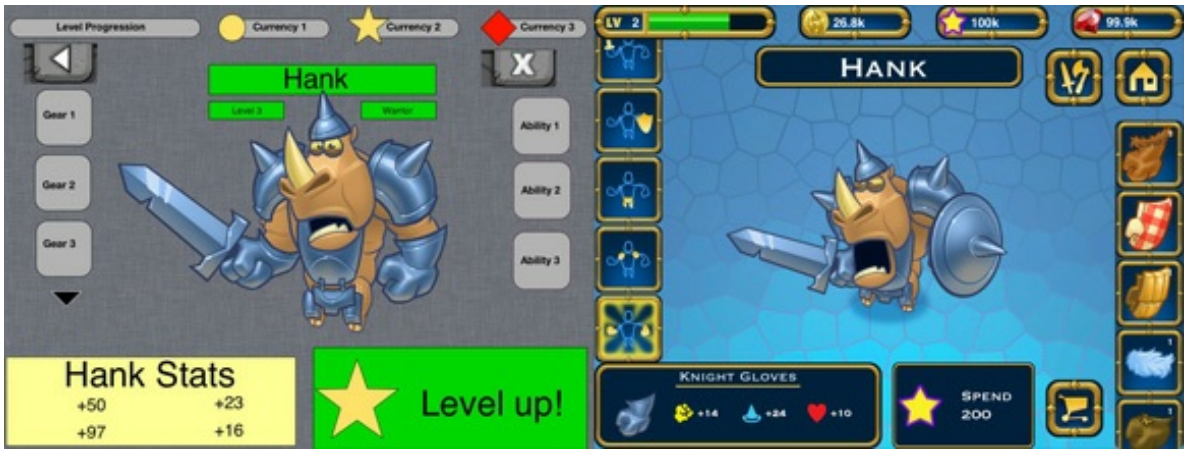
Developers can import images from an iPad and/or Dropbox, add them to a screen, adjust them, turn them into a button and then begin linking to other screens. The next step is to "play" the mockup to experience it as a full-screen app. Once happy with your prototype, AppCooker allows you to export these screens in multiple formats, the most useful of which is a free version of the product called AppTaster, which allows you to send AppCooker versions to whoever you like.

In roughly two days we completed a full first draft mockup of all the UI screens in the game and we were able to hand it over to someone and observe whether they were getting stuck or lost. This meant we could focus-test the UI without spending a single hour of programmer time. The ability to mockup an app as a webpage is one thing... Creating a mockup that runs on the target device at the proper resolution is a whole new game.

The final version of *Animal Legends* was extremely close to our early mockup, saving us a huge amount of time on UI

iteration. It's really hard to describe the vast difference between showing someone a UI flow on the device versus *describing* it to them.

We will be using AppCooker in all of our designs for iOS moving forward.



Left: Mockup, Right: Final

2. Online components

In the past couple of months, we heard of a number of small studios who were forced to pull their apps due to skyrocketing server costs. This had us a little worried since the core of *Animal Legends* -- being able to borrow and lend heroes to friends -- meant heavy interaction with social networks and an always connected setup. It was easy to do a "back of a napkin" estimate on what the costs would be -- our main concern was bottlenecks on the server, unprecedented hits which would cause the number of instances to spin up dramatically.

In *Animal Legends,* players can freely borrow heroes for adventuring from other players. Once the hero is used on a quest, that hero is sent back with a reward. This meant that players would need to register their social binding, see other players who are connected to their Facebook/Game Center accounts, pull down that player's data, pull down their hero's data, and then finally handle the various interactions of using that hero and sending a reward back to the player. This was a lot of data moving around for a mobile app. On top of that, combat was core to the main loop of the game and it needed to be accessible in three- to five-minute chunks -- that meant we had to *pre-cache* a lot of this data.

One of the best ways to be surprised in development is to hand a system over to hundreds of thousands of people to bang on it for man-years in a single day. When that system can cost you a substantial amount of cash, surprises can be *deadly*.

Using Rackspace we simulated the load of roughly 100,000 concurrent players for one hour to see what our costs looked like for server maintenance. Using mobile metrics we could safely extrapolate 100,000 concurrent players to roughly one million players per day. This took roughly 12 instances of Rackspace and it spun up an approximate 50 instances of our own servers to support it for datastore and bandwidth usage. We knew that the hit on datastore was highest when the player first started to play the game, so adding 100,000 fresh players would give us a worst case scenario.

After running the simulations we were pretty happy with the numbers. Our servers would set us back, but we could handle it.

To further protect ourselves, we made the caching frequency of a players' social data (one of our biggest server hits) remotely patchable. Players were updating their friends on how they were doing and what heroes they were using

roughly every hour. Having this variable remotely changeable meant that we could scale this out to six hours, or even 24. This slowed down social updates, but if this proved to be a big tax on the servers it gave us a throttle that we could adjust without shutting down the game.

Finally we put in a Big Red Button to shut all online services down and made sure that the core loops of the game would still work. If the lending of heroes proved to be too taxing on servers, we could flip the switch and shut down things gracefully on an app level. Though this was more work, it was far more elegant than pulling the server and creating tons of error messages -- or pulling the app entirely, which meant missing out on potentially millions of downloads.

3. Mitigation of additional risky components

At the beginning of the project, we scoped out our feature set within a roughly 80 percent accuracy and then began triaging what we believed the riskiest technical and gameplay components were. Emerging at the forefront was how our heroes and gear would work.

The design of *Animal Legends* was that players were collecting heroes, sending them on adventures and outfitting them with gear. Gear gets you stats and makes your hero better but also changes the visuals -- allowing players to customize them. We thought this was fun, compelling and the level of customization seemed endless*...* Until it became a little bit *too* endless.

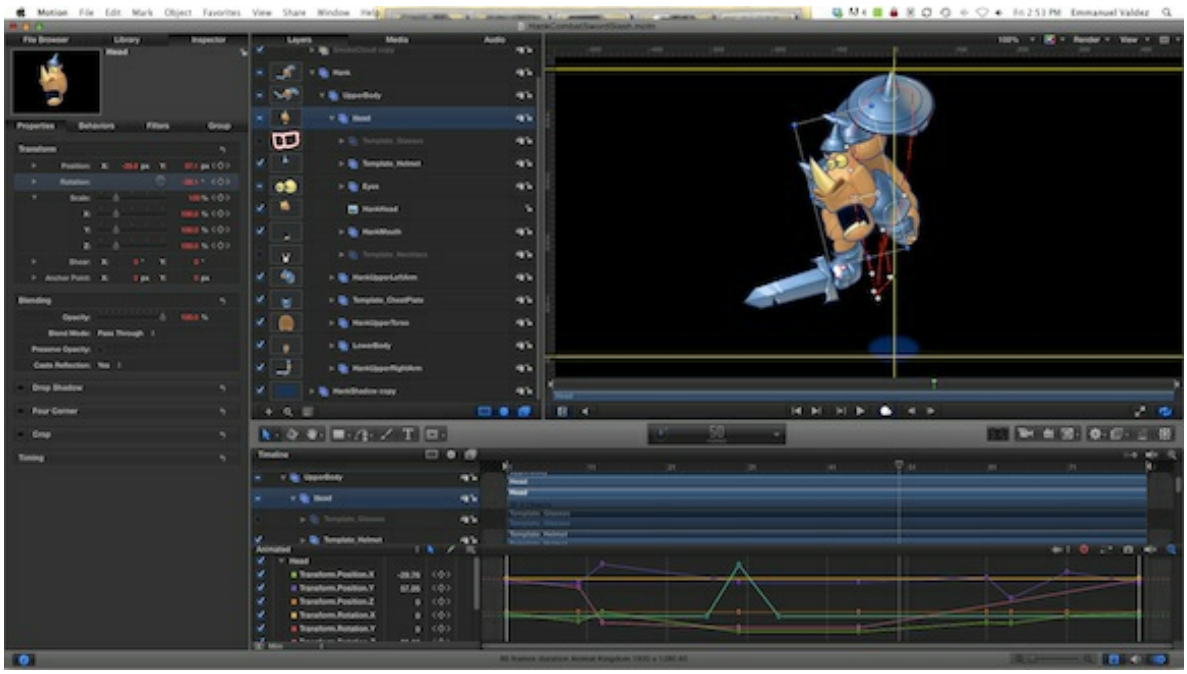
The gear system allowed for more than 200,000 permutations with just our launch content. Heroes could hit a certain critical mass where they were wearing different suits of gear, hitting different texture pages and finally overrunning memory in the app. We believed that if we shipped with three heroes, each carrying five sets of gear, we would be at the edge of expansion in the system. This meant we couldn't add more heroes or gear. This wasn't an acceptable solution for a game about collecting heroes and outfitting them with gear!

We spent a good 10 hours brainstorming solutions. There were lots of possibilities but most of them proved to require a very large effort in overhauling our internal engine.

Finally we conceived a simple alternative which was to create a dynamic texture page system. This would mean that each hero in play would get their own page and, regardless of what they were wearing or how many suits of gear we had in the game, the player would only have as many pages as heroes.

However, this also meant that each hero had to fit in a single page, so now we not only had to scope out these heroes but we also had to scope out *everything* that they could ever wear and the maximum PNG size for each item. If, months down the road, we wanted to add a "necklace slot" it just wouldn't be possible unless we put it on the character right there and then.

We began by sketching out all of the various body pieces which would compose the heroes and all the various pieces which they would wear. This would give us a complete footprint for our texture pages, to ensure a hero could fit in a single page. It would also give us the footprints for creating the gear and pieces themselves.



Click for larger image.

Once we understood the boundary conditions of the system, we could begin blocking out the size of the gear and how it would work with character animation.

---

4. High production values

By mitigating the risks above, we were able to block out all of our planned characters and all of their gear. While this meant artists were very tightly constrained to sizes that they could not deviate from, it also meant that they had a sandbox where they could play to their heart's content.

After having given the tech behind assembling characters such a fair shake we had a pretty good idea of what the system was capable of and what assets would work -- and which wouldn't. As a result, assets that we created in-house or through outsourcing were almost entirely plug-and-play, which saved us a good amount of debug time.

But the greatest advantage was that the artists could *really* iterate on the heroes and the gear without worrying about the work getting lost due to technical changes. As a result, the heroes are incredibly unique and sport all kinds of crazy animations. By constraining our art team, we actually *empowered* them to iterate without fear of change.

5. Planning past the future

We knew we would want to launch a large series of content right after launch and we planned out our features for a 1.1 and 1.2 as part of the initial feature set. This meant that as our artists were creating assets for buildings, they were actually creating about 20 percent more than the 1.0 requirements for every type of asset (such as heroes, gear, monsters and buildings).

Doing art assets this way had a similar effect to shooting multiple movies at the same time. The tone on everything was fairly universal. Additionally, things were a little bit faster to make as artists were producing these assets while they were "in their groove" and not re-learning a pipeline to do one or two assets. When you are talking about assets which take one or two days to make, a half day re-learning process is a dramatic increase in time.

A large number of our content (buildings, heroes, monsters and backgrounds) was all ready to go for a Halloween update, a Holiday update and then a major update for 1.1 which added a few new features such as being able to train your heroes and make use of the extra gear you had lying about.

Having this content ready to go right after launch led to a very smooth update path, which helped the app gain lasting appeal and resulted in a healthy number of post-launch downloads.

What Went Wrong

1. Not enough time in soft launch/beta

The soft launch -- launching in a small limited market prior to going worldwide -- is starting to become a standard practice in mobile game development. For a period of a few weeks (or a few months) the game is released in a single market. American developers often soft launch in Canada. During this time, community managers can poll players for bugs and developers can fix any issues before the app is released into the wild globally. A soft launch is an extremely powerful tool -- especially for apps like *Animal Legends*, ones that are "always on" and built to be easily modified remotely.

Animal Legends was soft launched in Canada two weeks prior to the worldwide launch. While it allowed us to identify and fix several major problems, it was just not enough to get all the bugs within our advertisement services sorted out. These bugs mostly affected international inventories but we just didn't have enough time to get a fix in, resubmit, and

hit the date we had penciled in with Apple.

Ultimately, the quality of the gameplay experience *was* there but some of our secondary revenue services *were not*, so we opted to move ahead with our launch date. In retrospect, we did lose some revenue in international markets. However, had we delayed the launch, we could have lost our Featured slot with Apple.

In the future, we'll pencil in a much larger testing window before we give Apple a date. If we don't find any major issues, we can keep the app in the wings until the pre-set launch window.

2. Lack of extensive testing of remote data

Animal Legends (and *SpellCraft* before it) were built to be updated remotely. All aspects of the economy, combat, heroes, and gear are placed into "plists" which can be changed by updating our remote servers. This allows us to fix bugs or issues that players run into within just a few hours instead of having to resubmit the app.

Though we tested dozens of variables to make sure they propagated right, *Animal Legends* features *thousands* of variables. As you might expect, some of those (specifically, three of them) didn't update in the app properly when changed remotely. One of the variables simply refused to deploy; another would apply the change and, every time that specific game actor would update it, roll back to the original data.

Once they were identified we were able to resubmit and patch them within a short period of time. However, we lost precious time. In the future, we'll test *all* our remote variables before releasing the game.

3. Lack of preparation for emerging markets

We invested in a large amount of translations (English, French, Italian, German, Spanish, Simplified Chinese, and Japanese) for *Animal Legends* so we were ready for international markets. But we weren't fully ready for emerging markets -- especially China.

SpellCraft enjoyed a good amount of traction overseas with roughly 60-65 percent of the total number of downloads coming from places outside of North America -- mostly Europe. *Animal Legends*, however, has pushed that number up as high as 75 percent, with downloads mostly coming out of China, Brazil, and other emerging markets. To put the number in perspective, *FaceFighter, SpellCraft* and *Trucks and Skulls* had roughly 10-15 percent out of China. For *Animal Legends*, China represents roughly 30 percent!

The Great Firewall presents problems with some of our online features (like Google's AppEngine, which isn't welcome in the country in any form). Other services -- such as Amazon S3 -- are very slow. While the Great Firewall didn't stop the game from functioning, it did occasionally stop us from deploying a patch or sending messages to our players.

We didn't integrate Sina Weibo or QQ either, which meant that *Animal Legends* in China lacks any sort of sharing to social networks.

We are currently making substantial changes to the game for China, including placing servers on Chinese soil and enlisting the aid of native Chinese companies. Fortunately, a number of companies have emerged to help western developers bring their titles to China, like Yodo1.

The Chinese market is by far the most alien for western game developers. What the Chinese consider appealing in terms of design, art style and business models are quite different from the world beyond the Great Firewall.

The Chinese market has always been important -- but it's now the top mobile market worldwide, surpassing even the U.S. Fortunately, companies are beginning to emerge to help Western developers not just get into China, but to have a fighting chance in their competitive app market. With the help of Yodo1 we've "deep localized" *Animal Legends*. Updating our art, marketing assets, localization and even our UI to meet Chinese players' expectations. The company has done a tremendous job and their CEO Henry Fong has been a great ally for our studio.

4. Mana economy

Every hero's abilities are driven by mana. Some of the weaker abilities can be cast for very little mana, while the more powerful abilities consume a good amount. Mana regenerates over time, but it's a key driver for combat, and the main throttle for *progression*. While the mechanic works well in combat, if the player leaves the app and heads back to their town they can't tell how much mana is left for each hero, whether they're currently in or out of combat.

The feedback the player receives on mana regeneration isn't really an issue if he or she is playing nonstop, but most play *Animal Legends* six times per day, leaving the app and getting back in multiple times.

The relevant issue seems to be to know for sure when heroes have full mana (energy) and if they are ready to adventure again. We are working on it for a future update, which will add a feature that communicates each hero's mana from the main town view.

---

5. Reward system created issues with player expectations

At the end of every battle, the player has a chance to win a number of rewards ranging from in-game currency (gold), XP, and gear. In the current spread, currencies are common and gear is a little more sporadic.

Defeating enemies and getting a random reward sounds like standard RPG fare, right? To make things a little more interesting for players, we decided to present them with *possible* rewards. Players can make their pick, get their reward and ogle the remaining randomly populated rewards as well. The idea was to compel players to do the mission again once they realized what the other rewards were. We felt this added extra pizzazz to an otherwise boring rewards screen while detailing the different kinds of loot available to players.

It *did* add pizzazz, but it also added frustration!

Though players can win more than three items, showing three choices inherently implies that the players' chances are equal for the items themselves, whether gold or gear. In the end, showing a player what they *could have won* made the game frustrating even though it followed the same rules probability-wise as the winning reward. As a result, almost every negative review we received mentioned the reward mechanic. Correlation isn't necessarily causation, but it was obviously a major issue.

Had we used a simpler interface to communicate what other items the player could have won, this wouldn't be an

issue. It would also have made the game easier to develop.

We've recently revised how the system displays rewards and dramatically changed the odds of winning "nicer" items. This has reduced a lot of the negative feedback on the reward mechanic. We'll continue to keep an eye on it and may completely revisit this gameplay mechanic at a later date.

What's Next for Appy

More and more iOS gamers download *Animal Legends* every day. We will continue improving the game based on player feedback, like we do with all Appy titles. Since launching Animal Legends we've added one new hero and five new monsters to fight. Our next patch will include more content and a player vs. player mode that allows players to fight with their heroes against others!

The lessons we've learned from *Animal Legends* and the tech that we've built will also apply directly to our next yet-unannounced-game, which is now in preproduction.

Data Box

Developer and publisher: Appy Entertainment

Release Date: 11/14/2012

Number of developers: 7

Length of Development: 7 months

Platforms: iOS universal app for iPad and iPhone/iPod touch

Development Tools: AppyEngine (Proprietary), Versions, Objective C, Ruby on Rails, Motion, Photoshop, AppCooker, Xcode