## FRIDAY, JANUARY 22, 2010

### [Fragmentation? More like Fragmentawesome.](http://replicaisland.blogspot.de/2010/01/fragmentation-more-like-fragmentawesome.html)

I'm lucky enough to have occasional access to lots of different Android devices via my work. The whole point of the Android approach to apps is that you can write an app on one device (or even an emulator) and deploy it across everything. In my case, that's been pretty true. I've tried Replica Island on the following devices:

* Google Nexus One
* Verizon Droid by Motorola
* HTC Magic (The myTouch in the US, HT-03A here in Japan)
* HTC Dream (The G1)
* Samsung Behold II
* Samsung Galaxy
* HTC Hero
* HTC Tattoo
* LG Eve
* ODROID
* Covia SmartQ5
* Android for x86

The cool thing is, Replica Island ran on all of them without complaint. That's not to say it was playable on all of them--the Covia SmartQ5, for example, has no keyboard, trackball, or even orientation sensors, and you have to use a stylus on the screen (it also has no GPU so the game runs extremely slowly). And some devices (like the LG Eve) have directional pads that are really poor for games. I ran the game under Android for x86 via virtualization, and while it worked it was slow enough that I have new respect for the Android emulator. But the game runs, pretty much error free, on every single Android device I've tested it on.  
  
I don't have regular access to all this hardware to test on, so when a phone comes my way I jump on it, install the game, and try it out.  Excepting basic support for multiple screen sizes and input systems, I don't have any device-specific code. I developed Replica Island almost entirely on Dream and Magic devices; it's built for a trackball and touch screen. I've added support for directional pads as well, which covers almost all of the phones on the market. The game runs at a nice frame rate on the Dream/Magic hardware, and it's very smooth on faster devices like the Nexus One and Droid. But that's it--no special case code for any particular device anywhere. Cool!   
  
You might be interested in which of these devices plays Replica Island the best. The answer might surprise you. Wait for it... ok, it's the ODROID.   
  
Yeah, seriously.   
  
What the heck is the ODROID? Well, it's an Android device (not a phone) sold by a South Korean company called [Hardkernel](http://www.hardkernel.com/). Actually, the consumer version isn't even for sale yet, but you can buy test hardware for nebulous "development purposes" on their web site. The~~Wonderswan~~ ODROID runs Android 1.5 (sounds like they will have a 2.0 update soon), has an HVGA screen, a Samsung CPU at 833mhz, a touch screen, orientation sensors, and all the other standard stuff. What sets it apart is the game-focused form factor, a real directional pad, and four face buttons. The thing is clearly designed for games.

[](http://1.bp.blogspot.com/_EIf-QLT1Gfk/S1mzAkCL55I/AAAAAAAAAC4/3QV8gYte73k/s1600-h/Picture+12.png)

My ODROID arrived in the mail yesterday, direct from South Korea. It's clearly prototype hardware; the thing is made out of light plastic and looks fairly cheap. There's a very strange power button that doubles as a hold switch and screen orientation switch, and I keep accidentally hitting the capacitive volume controls where I expect shoulder buttons to be. The directional pad is actually pretty bad compared to what you'd find on gaming hardware like the Nintendo DS or Playstation Portable, but it's better than your average phone. The thing can talk to adb, it came with an SD card already installed, and supporting the A/B/X/Y buttons is trivially easy (they just map to regular keyboard events).  
  
But the reason that the ODROID is the best device for playing Replica Island isn't just because of the game-like form factor and controls. Unlike most other Android devices, the ODROID is a combination of fast CPU and medium resolution screen. The devices with larger screens tend to be fill-rate bound; though the Nexus One and Droid have extremely capable CPUs and GPUs, the high resolution screens on those devices work against them when it comes to games (it's almost impossible to break 30 fps on those devices via the GPU, though they can crunch really complex scenes at that speed without breaking a sweat). On the other hand, the Magic/Dream class of devices have the same HVGA resolution, but tend to be CPU bound--rendering is fast but I spend quite a lot of time running the game simulation. The ODROID has neither problem--its CPU is pretty fast and its GPU has no problem filling the HVGA display. As a result, Replica Island is silky smooth on the ODROD--a constant, reliable 60fps. Add that to the game-ready control scheme and you have a pretty great game experience.  
  
That's part of what's so awesome about the run-everywhere Android approach. Replica Island works on all Android devices, pretty much by default. It's fun to play (well, I think it's pretty cool) on most average phones. But when some crazy company wants to make some crazy device that's good at one thing, they can do so without requiring any changes to the applications themselves. In this case, the ODROID is a surprisingly solid game device, at least for games like mine (though other games, particularly 3D games with complex scenes, are probably faster on devices like the Nexus One). And supporting it costs me literally nothing; I just loaded it up and it worked. That's pretty neat.

POSTED BY [CHRIS](http://www.blogger.com/profile/16079744366139636224)AT [6:20 AM](http://replicaisland.blogspot.de/2010/01/fragmentation-more-like-fragmentawesome.html) [27 COMMENTS:](http://replicaisland.blogspot.de/2010/01/fragmentation-more-like-fragmentawesome.html#comment-form)

LABELS: [TARGETING ANDROID](http://replicaisland.blogspot.de/search/label/targeting%20Android)

## WEDNESDAY, JANUARY 13, 2010

### [The Elusive Perfect Platformer Camera](http://replicaisland.blogspot.de/2010/01/elusive-perfect-platformer-camera.html)

I've come to believe that platformers live and die by their camera system. The camera system is the code that decides how the player should be centered in the frame. The camera must somehow track the player as he moves such that the player can see what's coming. That might seem like a simple problem, but it's not. In fact, I'll go out on a limb and say that a bad (or even mediocre) camera system can ruin a 2D scrolling platformer game.  
  
I mentioned in a [previous post](http://replicaisland.blogspot.com/2010/01/tuning-with-metrics-redux.html) that the data coming back from my play testers showed them dying in droves in bottomless pits. I guessed that this had to do with the camera system scrolling up and down, and I was right; a review of the camera code revealed a lot of room for improvement, and after some tuning and bug fixing, I think the experience is much improved.  
  
But this experience again drove home the point I made in the intro paragraph: that the camera in a 2D scrolling platformer has the potential to affect the play experience dramatically--it has to be as perfect as possible. I've made a lot of side-scrollers before, and I should know this, but I was still surprised by how much play was improved by a few simple camera tweaks.  
  
If you are ever at a loss about what to do when it comes to 2D platformer design, refer back to Super Mario Bros. It's like the bible of platforming games--every problem that you might encounter has already been solved, and it's probably been solved in a way that works better than whatever you came up with. At least, that's been my experience. Take a look at this video from Super Mario Bros. 3. Pay attention to the amount of vertical scrolling that the game does when the player gets close to the top of the screen.  
  
  
  
You can see that the game almost never scrolls vertically. The really interesting case is around 0:56, where the level (which has previously refused to scroll vertically) scrolls up in one very specific point to get the secret 1up. It's like vertical scrolling is only allowed in very specific situations. You can also see this sort of logic at work when Mario grabs the tanuki suit and starts to fly--immediately the game begins to follow him vertically.  
  
Now compare the camera movement in Mario to the video below. This is Frogger Advance: The Great Quest, a GBA game that I worked on all the way back in 2001.   
  
  
  
Quite a difference, right? The camera is all over the place, but despite all of the motion it's pretty much impossible to see where you are going. Part of the problem is that Frogger himself is really big; he takes up so much space on the screen that the camera really has to move just to keep him in the frame. This is a leading camera--it's supposed to always show you the direction that that you are moving. But in practice the physics are so fast that even if the camera rushes to show what's coming up, the player doesn't have time to react. When we made this game we understood that players were dying because they couldn't see where they would fall after a jump, but we didn't understand what to do about it. If you watch this video, you'll see the player use Frogger's float move to slow his falling motion down; this move was added explicitly to combat fall-into-pit deaths. A better solution would have been to try to reduce the amount of movement of the camera by designing levels that don't need to scroll vertically and reducing the size of the main character.  
  
For Replica Island, my camera algorithm is based on the concept of a "window." I actually thought of it as a sphere when I wrote it, but my good friend and ultra-veteran platformer author [gman](http://greggman.com/) pointed out that it's more accurate to think of a window. The center of the screen is defined by the center of the window, so when the window moves, the game scrolls. The rule that the camera's target (the player) must always remain within the window. When the player crosses out of the bounds of the window, the camera must move the window so that it contains the player at his new position. However, as long as the player stays within the window the camera does not move. So the player is able to cause scrolling in a particular direction by pushing up against a side of the window.  
  
To fix the levels in which huge numbers of users were dying, I adjusted the bounds of the window so that almost no scrolling occurs in the Y axis until the player approaches the top of the screen. The camera also does not allow the player to move below the middle of the screen. So now a small jump causes no vertical camera movement, but hopping off a ledge keeps the player right in the center of the display. This makes seeing what's below you a lot easier than before.  
  
But the heuristic wasn't good enough on its own, so I've also added a special object that, when visible, biases the camera in one direction or another. This lets me put camera hints in the map in areas that I know to be particularly problematic.   
  
Finally, on a few levels I squeezed the level size down so that there's almost no vertical scrolling at all. This makes these levels feel a bit more like Mario, as the game almost never scrolls up and down. This makes the jumping puzzles actually fun, rather than one leap of faith after another.  
  
So far I'm pretty happy with the results, but the real test will be to compare this new version of the code and levels with the data that I presented before; if my theory is right, the number of deaths from falls should be dramatically reduced. If I'm wrong, well, it'll be another round of iteration. It's worth it though; bad cameras are the death of 2D scrolling games.

POSTED BY [CHRIS](http://www.blogger.com/profile/16079744366139636224)AT [11:43 PM](http://replicaisland.blogspot.de/2010/01/elusive-perfect-platformer-camera.html) [6 COMMENTS:](http://replicaisland.blogspot.de/2010/01/elusive-perfect-platformer-camera.html#comment-form)

LABELS: [CODE DESIGN](http://replicaisland.blogspot.de/search/label/code%20design), [GAME DESIGN](http://replicaisland.blogspot.de/search/label/game%20design)

## SUNDAY, JANUARY 3, 2010

### [Game Play Video](http://replicaisland.blogspot.de/2010/01/game-play-video.html)

Here's some footage of an early level in Replica Island.  Sorry about the audio.  
  
  
  
I think I've figured out a pretty decent pipeline for recording game play (this was the test), so more video is on the way.

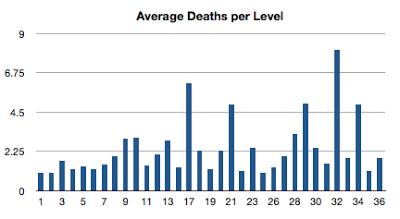
POSTED BY [CHRIS](http://www.blogger.com/profile/16079744366139636224)AT [4:52 PM](http://replicaisland.blogspot.de/2010/01/game-play-video.html) [6 COMMENTS:](http://replicaisland.blogspot.de/2010/01/game-play-video.html#comment-form)

LABELS: [MEDIA](http://replicaisland.blogspot.de/search/label/media)

## FRIDAY, JANUARY 1, 2010

### [Tuning With Metrics Redux](http://replicaisland.blogspot.de/2010/01/tuning-with-metrics-redux.html)

A while back I posted about the system I devised for [recording player deaths and plotting them on level images](http://replicaisland.blogspot.com/2009/11/tuning-play-with-automatic-and.html) to find trouble spots.  Since then I've improved the system a little and written some more tools to crunch the data I am receiving.  Several hundred players have now participated in testing Replica Island, so I have quite a lot of feedback to process.  The goal, of course, is to use anonymous statistics about play (specifically, where players died) to smooth out the difficulty curve and find frustration spikes.  
  
Rendering death locations on level maps is a good way to go about tuning individual levels, but what about the game as a whole?  I now have enough data to look at how players move through the game.  Maybe I can draw some conclusions about how the game is paced and how smoothly the difficulty increases.  
  
The first thing I did is graph the number of deaths for each level.  That graph looks like this:

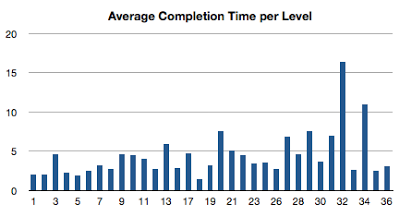
[](http://4.bp.blogspot.com/_EIf-QLT1Gfk/Sz4WsRFCSII/AAAAAAAAACQ/yE2OcD6dzgk/s1600-h/deaths_per_level.png)

As you can see, a few levels jump right out: levels 17, 21, 39, 32, and 34 all look like spikes in the death graph.  My theory here is that the game should get harder at almost exactly the same rate that the player gets better, so I would expect the number of deaths per level to stay uniform across the entire game.  And actually, this graph suggests that with the exception of the outliers mentioned above, I'm doing an ok job at keeping the difficulty increasing at a constant rate.  After fixing the very hard levels, I can see that the middle of the game, from around level 11 to level 24, there are some very easy levels as well.  I should probably go back and mix those up a bit to increase the difficulty.  So far, so good.  Looks like this is a pretty good metric for assessing difficulty.

But one thing this data doesn't tell me is how frustrating these levels are.  Just because a player died more than once doesn't mean that the level was frustrating; consider a poorly constructed level in which the player isn't in immediate danger but simply cannot progress.  In a case like that, the level could be extremely frustrating without the player dying a lot, and this graph wouldn't catch it.  
  
Let me tell you more about the data I have.  I collect a few basic fields for each event.  They are:

* event type - the type of event that occurred.  Usually "death."
* x, y - the position in the level in which the event occurred.
* time - the time since the level started, in seconds, until the event.
* level - the level in which the event occurred.
* session - a unique session key associated with the user (based on a random number generator).

For the first version of the game that I sent out to testers, only death events were recorded.  In the most recent version I added a new type: level completion events.  When a level is completed, it causes that event to be logged to the server.  With level completion events in place I can now track the total amount of time that a player spent on any given level by summing all of the time entries from death events associated with that user on that level and adding the result to the level complete time event.  Do this for every user and average the results and I can get a graph of how long each level takes to complete including restarts from death.  That graph looks like this:

[](http://1.bp.blogspot.com/_EIf-QLT1Gfk/Sz4ZbczDfBI/AAAAAAAAACY/Vcih5hdRjsE/s1600-h/time_per_level.png)

Now, Replica Island is a game on a phone.  It needs to be playable in short bursts.  So my levels are all designed with the idea that you can finish them in around 5 minutes.  And looking at the graph, it looks like my levels are pretty much in-line with that expectation.  Levels 32 and 34, which were outliers on the deaths per level chart, are again outliers here; those levels must really suck.  I mean, it's the end of the game, so the levels are supposed to be pretty hard, but the average player dies 8 times on level 32 and spends a total of almost 20 minutes trying to complete it.  That's probably super frustrating.  That level needs work.

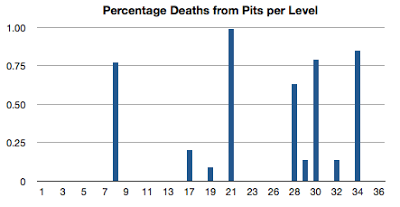
I can also see that there's something very wrong with level 18.  People are completing it in almost no time whatsoever, even though the other graph shows that most people die 2.25 times on that level.  And again, with the exception of the very long outliers, I can see that the levels in the end are sometimes too easy; ideally all the bars on this graph should rarely vary from the 5 minute mark.

Of course, these are all averages.  Some players might spend a very, very long time on a particular level (and actually, with my expanded view of the data, I can see some poor souls really are getting stuck for hours).  But on the other hand, I can't design levels for every level of difficulty; hitting the perfect median that is challenging for novices but not boring for pros is hard, but I think that averages are a good guideline.

So now I know which levels are probably worth investigating.  A cursory look at the levels identified by the death event graph revealed an interesting pattern: a lot of people seem to be dying by falling down pits.  Replica Island, like many other side-scroller games before it, features pits that you must jump across.  If you fall down a bit and off the bottom of the level, it's game over.  It turns out that many of the levels in which people are dying in droves are levels that feature a lot of pits, like this one:

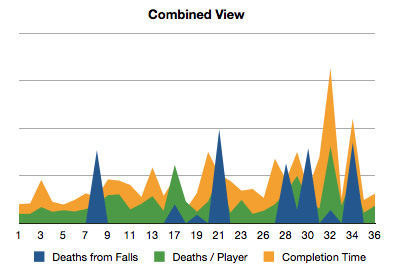
[http://2.bp.blogspot.com/_EIf-QLT1Gfk/Sz4bqfu0RtI/AAAAAAAAACw/W37UBvRVlwY/s640/pits_of_death_level.jpg](http://2.bp.blogspot.com/_EIf-QLT1Gfk/Sz4bqfu0RtI/AAAAAAAAACw/W37UBvRVlwY/s1600-h/pits_of_death_level.jpg)

Every little dot on this image is a player who died, and 99% of them are in pits.  So it seems like I might have a problem with pits that affects a lot of different players.  Rather than make that assumption, however, I graphed it:

[](http://1.bp.blogspot.com/_EIf-QLT1Gfk/Sz4ZebH_hJI/AAAAAAAAACg/hV3asc7qNdQ/s1600-h/death_from_pits.png)

I can detect a pit death from my data because the y coordinate in my x,y field goes negative only in this case.  And yep, it looks like levels with a lot of pits are deadly.

So how pervasive is this problem?  I know by looking at the death event graph and the completion time graph that I've got some outlier levels to deal with, and this latest graph tells me that some of the levels are affected by some specific problem with pits.  So what's the relationship between the pits and the overall level flow that I'm trying to control?  Let's graph it and find out!

[](http://1.bp.blogspot.com/_EIf-QLT1Gfk/Sz4ZhEkEz6I/AAAAAAAAACo/Q_ti7QmRvwY/s1600-h/combined_view.png)

This is pretty neat, right?  I can tell that in several cases, in particular level 34, the pit problem is leading to a lot of deaths which is causing the level to take forever to complete.  I can also see that the pit issue isn't my only problem: problematic level 32 has almost no pits, and yet the number of deaths and the time to complete that level is still way out of range compared to the rest of the game.

In case you are interested, the problem with pits is actually pretty simple: my camera moves too much in the vertical direction.  Classic platformers that involve a lot of jumping, like*Super Mario Bros.*, are very careful to limit the amount of vertical movement required by the camera when making jumps.  You can always see the landing point before you leave the ground in the *Mario* games, which makes jumping over pits a fun challenge rather than a frustrating penalty.  But in Replica Island, the protagonist [can fly](http://replicaisland.blogspot.com/2009/09/why-android-flies.html), which requires the camera to track in the vertical axis a lot more aggressively.  That means that if you are falling you might not actually realize that there isn't going to be any ground below you until it's too late.  The solution to this problem is probably to make the camera smarter about pits, and maybe to throw some visual indicator into the mix as well.  We'll see--I have a couple of solutions in mind but it'll take another round of testing to verify them.

Anyway, based on an extremely simple packet of data from a sample of several hundred players, I can pretty accurately pinpoint levels that need polish and game mechanics that are sources for error.  As I push forward to a public release, my goal should be to first fix the mechanical issue (the pit problem), and then polish levels like #32 that are broken for some other reason.  That's a much better task list than just some gut feeling about the quality of my levels.

POSTED BY [CHRIS](http://www.blogger.com/profile/16079744366139636224)AT [8:20 AM](http://replicaisland.blogspot.de/2010/01/tuning-with-metrics-redux.html) [4 COMMENTS:](http://replicaisland.blogspot.de/2010/01/tuning-with-metrics-redux.html#comment-form)

LABELS: [GAME DESIGN](http://replicaisland.blogspot.de/search/label/game%20design)