Digital Culture & Education (DCE)

Publication details, including instructions for authors
http://www.digitalcultureandeducation.com/

Platforms in the cloud: On the ephemerality of platforms

Casey O'Donnell

Michigan State University

Online Publication Date: 14th November 2016

PLATFORMS IN THE CLOUD: ON THE ephemerality of platforms

Casey O'Donnell

Abstract: *This essay explores the ephemeral character of platforms and the critical role that theoretical frameworks play in making sense of platforms as socio-technical assemblages. Through an exploration aimed at further complicating the Nintendo Wii as platform and exploring Twitter as platform, the essay considers the crucial role of theory in the unpacking of black boxes. In an attempt to render platforms accessible, it is quite possible they have been presented as more opaque, and their analysts have not adequately debugged the messes they have encountered.*

Key Words: Platform Studies; Alien Phenomenologies; Science and Technology Studies; Twitter; Debugging

Introduction

In the early summer of 2012 at a meeting of the Foundations of Digital Games (FDG), I assembled a variety of researchers interested in exploring the micro-blogging site Twitter from the perspective of platform studies. I populated the panel with researchers and developers who had explored or created games or software for Twitter. I asked each of them to think about Twitter as a platform. Yet, according to Montfort and Bogost's original charge of platform studies, such a thing would not qualify as a platform by their original definition. As others have noted, the conceptual category of platform has enjoyed a great deal of scholarly attention, which has not been explored by that of platform studies (Leorke, 2012). According to the more orthodox version, Twitter was simply "software that runs on platforms," (Montfort & Bogost, 2009, p. 3) rather than a real platform, "[w]hatever the programmer takes for granted when developing, and whatever from another side, the user is required to have working in order to use particular software," (Montfort & Bogost, 2009, pp. 2-3). Yet, what was interesting was that as I attempted to explore Twitter games from this perspective, I increasingly grappled with the ephemerality of precisely what a platform was. At first I assumed it was a consequence of my analytic transgression, but in reality, similar questions could have been leveled at any number of platforms. In my contribution to that FDG panel, I returned to *Codename Revolution: The Nintendo Wii Platform* (Jones and Thuruvathukal, 2012) and began to re-ask the question, "Where is the platform?"

The ephemerality of the Wii

In the spring of 2007, Chris Hecker, a game developer long associated with the Game Developers Conference (GDC), and in particular as part of the perennial 'rant' session, spoke. He began that year's rant with the relatively simple, but spirited statement, "The Wii is a piece of shit!"[1] This statement sent shockwaves through the enthusiast press surrounding the game industry. Game developers present in the room at that time, however, cheered and applauded. It was striking to see such heated enthusiast press coverage over something, which for developers was a humorous non-event. Hecker went further, mocking the console, presenting a slide that made it appear as if the "severely underpowered" machine was simply two GameCube consoles duct-taped together. Again, the audience roared. What struck me as funny, at the time, was that the original Development Kits, or DevKits, for the Wii actually were GameCube DevKits with a few additional wired inputs. At that moment, in 2007, I was nearing the end of a three and a half-year stint of ethnographic participant observation at a game studio working on a PS2 and Wii title. For this team, the Wii, as a platform, started as a GameCube DevKit—, not two of them taped together, just a single GameCube DevKit. I suspect that many game developers in the room that day were in on that aspect of the joke as well. Of course the Wii became the more self-contained and expanded device as time moved forward.

As I read *Codename Revolution*, I couldn't shake the feeling that something was missing. Even as an attempt to explore the platform of the Wii, the device remained too 'thingified', too black boxed, too polished. I always perceived one of Platform Studies' goals to be the dynamic decompression of highly black-boxed[2] machines and the games that ran on them. However, the Wii was presented very much as the device found on a store's shelf. It was slick. It was finished. Yet, as the numerous updates issued to the Wii's firmware over the years ought to indicate, it was far from finished. Ultimately, it was halfway through the text that I found the source of some of my anxiety.

> Developing software for the Wii involves programming for its particular affordances and within its particular constraints. This includes first of all programming for the intuitive interface provided by the Wii Remote controller. In fact, Nintendo's SDK, which a third-party developer can purchase[3] after being successfully accepted and registered as an official Nintendo developer,[4] ships with a tool called LiveMove, made by AiLive, which offers a graphic user interface (GUI) for "training" a new game to recognize a repertoire of motions by the Wii Remote. The programmer turns it on, repeats a series of named sample motions (flipping a pancake, say) in various speeds and at different angles, and the program then captures those and automates the code for integration into a game under development (in this case, it might be a restaurant game). (Jones & Thiruvathukal, 2012, p. 73)

I have no idea when LiveMove became a product included with the Wii DevKits, but it certainly hadn't been part of the platform the team of developers working on *Spider Man 3* initially encountered. I recall the heady days of development; when the team working with on the Wii dumped WiiMote data into Excel in order to do the kind of statistical analysis that surely LiveMove was attempting to automate for the developer. So is LiveMove part of the platform? Or is it a swerve in the platform? The ambiguity of the data from the WiiMote is what led to the development of Motion Plus for the WiiMote. So, which one is the platform? As a developer, what is to be assumed? With questions like this, it was too late. I'd gone down the rabbit hole.

Nintendo's broader software SDKs, included with Wii DevKits, contain all sorts of utility libraries that encode ideas about how games ought to be developed. Memory use and allocation is actually quite different between games and "typical" software.[5] Nintendo's SDKs explicitly encourage the use of memory pools. The format that images and data take within the file system, assumed by the Wii's underlying operating system, are also encoded in calls specific to the Wii's SDK. The entire SDK remains a decedent of the GameCube's SDK. What about the wide use of the scripting language Lua by many Nintendo developers? All of these are part of the platform according to Montfort and Bogost, but it's most certainly not part of *Codename Revolution*. So, where precisely is the platform?

Once I started down this line of inquiry, it became much more difficult to isolate platforms. But, that is precisely what makes them such interesting units to explore. They swerve and shift throughout their lifespans. Examples abound, the Nintendo Entertainment System's (NES) use of various "mappers" on individual cartridges that adjust the behavior of memory mapping as it relates to the underlying Pixel Processing Unit. The introduction of battery aided game-save systems is also an excellent example. The Super Nintendo Entertainment System's (SNES) ability for cartridges to not only contain memory mappers, and semi-volatile memory, but actual co-processing units. For example, many developers' realized that texture context switches were particularly computationally expensive on Sony's PlayStation 2, which lead to the re-discovery and use of Atlas Texturing to work around those limitations.

Time and again, the very assumptions that developers can make about a system are liable to shift and move under their feet throughout the creative process and over the lifetime of a piece of hardware. Which, isn't to say that these objects aren't important and worthy of analysis, but that an approach that mimics those applied to the Atari VCS may prove insufficient to unpack and explore these important artifacts. Perhaps this critique matters doubly so as platform's tendrils reach ever further to distributed computer platforms.

On Twitter games

What makes Twitter an interesting platform is that it tends to exhibit many of the issues that are core to exploring what precisely is meant by platforms and our analysis of them. In February of 2011, the third Global Game Jam kicked off with an accompanying 'achievement,' titled, 'Aggregation,' which required that whatever game was created "uses or combines existing web services and online data (e.g. Google Maps, Gmail, Twitter, Facebook, airline services, news, stocks, etc.) as part of the gameplay." Out of this a handful of games were developed that made use of these elements. Two games, in my opinion, stood out. Each used Twitter's REST API[6] as a means for exploring gameplay. Twoetwy and twitapocalypse were two games, built on very different technologies, each using Twitter as a core element of their gameplay.

Twoetwy was built using Adobe's Flash authoring system. The game connected to Twitter's 'firehose,' or general public timeline. It used this data to populate a game space where a player assembles new tweets by flying a bird around on the screen, collecting falling words drawn from the deconstructed firehose-based tweets. Once assembled the player could then tweet the assembled phrase accompanied by the #Twoetwy hashtag.

Twitpocalypse was built on top of the relatively new HTML5/CSS/JavaScript standards and the Impact JS JavaScript game development library. Twitpocalypse went further than Twoetwy, requiring the user to authorize the application, which then allowed the game to crawl through the user's personalized feed. The game then allowed the player, pictured as the Grim Reaper, to harvest unsuspecting people glued to their phone screens. The player was is rewarded for reaping those killing people that they don't follow and are penalized for killing not of those accounts that they do follow. An incorrectly reaped soul then displays their profile image, indicating your mistake. The game played with what might be referred to as the "Facebook Effect;" do we know the people we follow or have friended?

Returning to the question of platforms, however, how would one characterize either of these game's platform? Neither is reducible to HTML5 or JavaScript or JSON or Flash or PHP or Twitter or Chrome or WebKit or HTTP. Even those objects have shifted and mutated over time. What does it mean to be a platform in contexts such as these?[7]

Analytic Invocation

In the formulation of *Racing the Beam*, much of the theoretical platform is never hinted at, ostensibly to widen the readership of the series. The lifeblood of a platform studies text is its own platform; the assumptions made by the researchers. These assumptions are the system through which material is made sense of and put into motion in the series. One can only suspect what precisely that underlying system was for Bogost and Montfort, though one rooted in Object Oriented Ontology (OOO) and Alien Phenomenology seems most likely.[8]

Yet, by not placing such material in the text, it is excised from the platform. This lack of conceptual nuance results in texts that explore platforms in ways that may have proven productive for the Atari VCS, but prove inadequate for others. To return to a concept deeply related to Latour's Actor/Actant-Networks, briefly indexed in the development of an Alien Phenomenology: that of Law's messes.[9] It is important to note, that Law's messes are deeply rooted in research practice (Law, 2003). Actor-Network Theory, for Law, goes hand in hand with research methods. Theory and practice are intertwined; they are part of his research platform. The mess, Law might argue, goes with interesting and unexpected research findings.[10] Rather than jettison the mess, I would press with a query specific to game development.

How do game developers make sense of the mess? Clearly, if one looks at the complexity of what massive teams of developers produce, games and the platforms they build these systems for, it's a mess. But, I don't believe that messes cannot be grasped. Even if it's a mess, there is a system at play that can be understood by humans. In the words of one informant, an engineer, from my early work with AAA game developers:

> There is an emotional component to it. You'll be like, "that can't be broken." You gotta get over that pretty quick. You say, "well, it's broken." Maybe you're afraid it's impossible. But you know, if you take the time, the gruesome horrible time, you can always, if you have to, map out the transistors and follow the flow of logic. It will come out in a wash. You have to be persistent.

I like to think of this approach as one modeled on the debugger. The debugger is a software programmer's tool that allows them to observe the execution of a program's code as it progresses though its execution. It is part technology, but also part mindset. It can be mind-numbingly tedious, as my informant noted with, "the gruesome horrible time." Yet, it is also part intuition. Some people are better at debugging than others. The process exemplifies what can make game development and the deconstruction of platforms difficult. Yet, it is precisely the ability of the analyst to 'step-into' those areas that intuition suggests are worthy of analysis.

There is no reason that our analysis of these large systems couldn't take on the character of debugging. What are the interesting sub-routines worth examining? Where are there the most bugs or glitches? Where are the fewest? Having a sense of what needs exploration is the kind of creative analytic work that I have cited as critical to the work of game developers. Why can't this approach be applied to our analysis of platforms?

None of this is to say that what has been produced as part of the platform studies series is not impressive. It certainly is and will likely continue to be. Rather, it is a provocation to would-be platform analysts to not only explore platforms already built, but to delve deeply into them in ways that surprise both reader and analyst. This may prove problematic, because many platform owners do not want the buggy, glitchy, or rough sides of their platforms made visible. Just as when Chris Hecker indexed a perceived failing of the platform, and was disciplined for his transgression, it is possible that platform studies may reveal aspects of platforms that are less than desirable or had unintended consequences, as noted about the NES (O'Donnell, 2011). It may also be that platforms were intended to be one thing, but over time possibilities are scrapped for reasons unknown, such as Mario Factory in the era of the SNES (O'Donnell, 2013).

Each platform explored by platform studies needs to debug, thoroughly and theoretically, each object. This will require a delving into the unit, its software, peripherals, code, and plethora of other elements. It becomes the analyst's job to develop an intuition for those elements of the system needing greater or lesser scrutiny. Being clear and upfront about what guided that particular deconstruction is critical.

Notes

[1] It is interesting to note that the rants of GDC often serve as important markers throughout recent game development history. They serve as an important release valve, or moment to index important shifts or issues facing the worlds of game developers. I have found them to be telling moments throughout my research on game developer practice and culture.

[2] While I respect the desire for platform studies to remain accessible, by not tipping their hat to the literature that supports the analytical framework of the texts, the render their platform opaque. It is critically important to what made *Racing the Beam* impressive and is lost to the lay reader. By disconnecting the text from those theoretical works, it renders the role it played in the analysis mute.

[3] DevKits are actually leased devices. They are not bought or sold. Because the hardware distributed along with the SDKs, often have capabilities (such as ROM writers) that the manufacturers do not want to have "in the wild." The devices remain owned by the company that created it. Developers lease the devices and the software through licensing agreements.

[4] This process is actually quite fraught and subject to a variety of rules and regulations that are largely undocumented, but well known to established developers (O'Donnell, 2011). Interestingly, in the case of the Wii and all of Nintendo's consoles, the developer licensing portal is known as "Wario World." Quite humorously, the landing pad for third-party developers for Nintendo is named after the annoying antihero of the Nintendo world.

[5] While a game's engine may be thought of as software, games should not be thought of as "just" software (O'Donnell, 2012). Many practices that are deployed within game development would be frowned upon by "traditional" software development practices. Because games often have the luxury of taking over a device, it isn't necessary for them to behave like traditional desktop software.

[6] Representational State Transfer (REST) Application Programming Interfaces (API) are one of the most widely used by web-based service providers.

[7] And these contexts also shift over time. Facebook or Twitter may make changes to their API that require modifications to existing games, or they may simply stop functioning. In some cases, these changes are made with little or no warning provided to developers. Games may stop working and it becomes the job of developers to respond to users frustrated with a broken game.

[8] My assumption comes from comments found in: (Bogost, 2012, pp. 131-132). What I like about the particular section noted is the chapter's entitlement "wonder," which is what makes *Racing the Beam* such a compelling text. It is clear from the material that both authors are in wonder and wondering with the Atari VCS. *Codename Revolution: The Nintendo Wii Platform*, however, leaves me less with a sense of wonder and more a sense of closure, something that would puzzle any scholar exploring socio-technical systems.

[9] As far as Actor-Network Theory theorists go, Law has proven very productive for my work. From heterogeneous engineering (Law, 1989) to monsters (Law, 1991) to messes (Law, 2003).
[10] Rheinberger (1997) has noted that it is actually the potential for noise or new messes that make experimental systems useful. If they are completely incapable of providing unexpected results, they become simply tools, rather than part of the experimental system groping to understand new and interesting things.

References

Bogost, I. (2012). *Alien Phenomenology: or What It's Like to Be a Thing.* Minneapolis: University of Minnesota Press.

Jones, S. E., & Thiruvathukal, G. K. (2012). *Codename Revolution: The Nintendo Wii Platform.* Cambridge: MIT Press.

Law, J. (1989). Technology and Heterogeneous Engineering: The Case of Portuguese Expansion. In W. Bijker, T. P. Hughes, & T. Pinch (Eds.*), The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology* (pp. 111-134). Cambridge: MIT Press.

Law, J. (1991). *A Sociology of Monsters: Essays on Power, Technology and Domination.* New York: Routledge.

Law, J. (2003). Making a Mess with Method. Retrieved from http://www.comp.lancs.ac.uk/sociology/papers/Law-Making-a-Mess-with-Method.pdf

Leorke, D. (2012). Rebranding the Platform: The Limitations of 'Platform Studies'. *Digital Culture & Education*, 4(3), 257-268.

Montfort, N., & Bogost, I. (2009). *Racing the Beam: The Atari Video Computer System.* Cambridge: MIT Press.

O'Donnell, C. (2012). This Is Not a Software Industry. In P. Zackariasson & T. L. Wilson (Eds.), *The Video Game Industry: Formation, Present State and Future* (pp. 17-33). New York: Routledge.

O'Donnell, C. (2013). Wither Mario Factory? The Role of Tools in Constructing (Co)Creative Possibilities on Videogame Consoles. *Games and Culture*, 8(3), 161-180. doi:10.1177/1555412013493132

O'Donnell, C. (2011). The Nintendo Entertainment System and the 10NES Chip: Carving the Videogame Industry in Silicon. *Games and Culture*, 6(1), 83-100.

Rheinberger, H.-J. (1997). *Toward a History of Epistemic Things: Synthesizing Proteins in the Test Tube.* Stanford: Stanford University Press.

Biographical information

Casey O'Donnell is an Associate Professor in the Department of Media and Information at Michigan State University. His research examines the creative collaborative work of videogame design and development. This research examines the cultural and collaborative dynamics that occur in both professional "AAA" organizations and formal and informal "independent" game development communities. His research has spanned game development companies from the United States to India. His first book, *Developer's Dilemma* is published by MIT Press.

Contact: caseyod@msu.edu