Article

Games and Culture
8(3) 161-180
© The Author(s) 2013
Reprints and permission:
sagepub.com/journalsPermissions.nav
DOI: 10.1177/1555412013493132
gac.sagepub.com
SAGE

Wither *Mario Factory*? The Role of Tools in Constructing (Co)Creative Possibilities on Video Game Consoles

Casey O'Donnell[1]

Abstract
This article examines the role that game development tools have in the design and creation of video games. It does so through the lens of a series of patents by Nintendo that outlined the technological foundations for a truly (co)creative production platform for games. Game development tools shape and are shaped by the kinds of games they are used to produce as well as the underlying technologies of game consoles. The roles, sites, and means by which users are allowed to or encouraged to engage with (co)creative tools significantly impacts the kinds of interventions users may make. The article makes the argument that tools, like *Mario Factory*, form a technological foundation for (co)creation, participatory design, and convergence at a level that differs significantly from the current forms.

Keywords
Mario Factory, tools, game development, (co)creation, platform studies, actor network theory, science and technology studies, STS

[1]Michigan State University, East Lansing, MI, USA

Corresponding Author:
Casey O'Donnell, Michigan State University, 404 Wilson Road, Communication Arts & Sciences Building, East Lansing, MI 48824, USA.
Email: caseyod@msu.edu

Introduction

This article examines the role that "tools" play in constructing and structuring the (co)creative possibilities of game developers and their users/players. Game development tools, or the technologies that surround, inform, enable/constrain, and otherwise make game development possible, have been underconsidered in our analysis of (co)creative labor. The article takes "*Mario Factory*" as its subject to map critical areas of (co)creation in need of further discussion and inquiry. *Mario Factory* makes visible technologies and ideas that in many cases remain behind closed doors of video game development companies. These systems are often covered by numerous nondisclosure agreements (NDAs) or are highly guarded trade secrets, and for this reason *Mario Factory* provides valuable insight into game development practice. Even in the case of more widely available game development technologies, few have extended themselves in ways that make them truly accessible to "everyday" users.

Conceptually, the article turns to Montfort and Bogost's conceptualization of "platforms" (Montfort & Bogost, 2009), Pickering's notion of "mangles" (Pickering, 1995), and Rheinberger's concept of "epistemic things" (Rheinberger, 1997) in conversation with von Hippel's "User Toolkits for Innovation." Together these frameworks illuminate how user's (co)creative capacity is structured.[1] Contrary to much that has been written about (co)creation around video games, this article argues that video games and console video game systems, more acutely, are structured in ways that disable the majority of (co)creative opportunities for players. The design of the systems used to make games has specifically not engaged the user. This is not to say that (co)creation does not happen or occurs in only "unimportant" ways, merely that it is constrained in ways shaped by the tools available to users, much as the tools available to developers to shape the creative process of the game development. Put simply, user forums (Banks & Humphreys, 2008) and machinima (Lowood, 2006) are different in important ways from the collection of technologies presented in this article.

The article begins by exploring the patents associated with *Mario Factory* and what precisely it was imagined to be. The article then explores that possibility through its conceptual framework, ending with a discussion of what was potentially lost in the decision, by Nintendo, to not pursue further the invention.

Introducing *Mario Factory*

On Halloween of 1994, Nintendo filed for a series of four patents that were later granted between the years 1997 and 2000. This article refers to them homogeneously as "*Mario Factory*" (Hibino & Yamato, 1994; Suzuki, Yamato, Koganezawa, & Ozaki, 1994; Yamato, Nishiumi, Suzuki, Hotta, & Fujiwara, 1994a; Yamato, Nishiumi, Suzuki, Nakamura, & Kimizuka, 1994b). This sequence of documents describes a video game development, testing, and manufacturing system designed

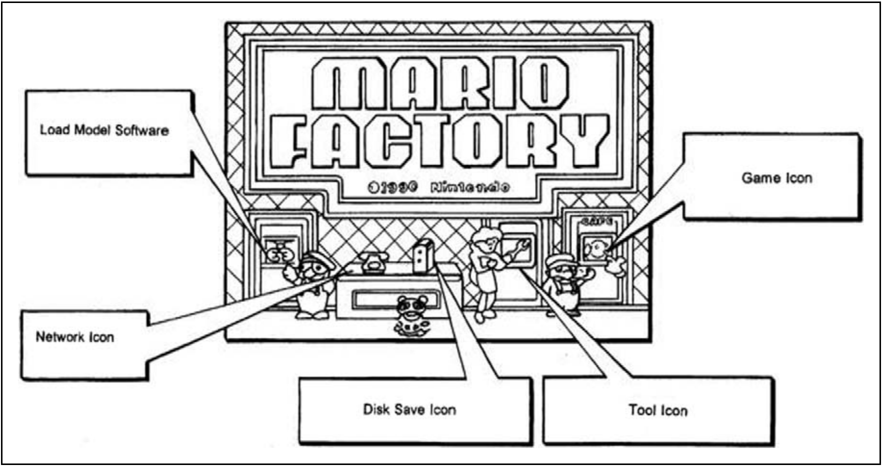

Figure 1. *Mario Factory's* title screen.

specifically for hobbyists and users to enjoy the creative possibilities of developing games for console video game systems. The title screen for this system can be seen in Figure 1. Many of the systems and ideas described in the documents have not yet come to market for licensed Nintendo game developers and certainly not for the general player or hobbyist game developer. *Mario Factory* is a critical lens for understanding and examining the (co)creative possibilities for users and hobbyists as they currently exist.

I make no claim that user-friendly tools do not exist or that there have not been significant developments in console manufacturers attempting to provide greater access to development kits. Rather, despite these efforts, none has come so close to delivering a fully integrated user-friendly experience that authorizes and encourages the kind of (re)mixing, mashing and sharing that is so often cited as crucial to convergence culture. I also make no explicit claim about the implicit public good of having tools such as *Mario Factory*. Indeed, there is a reasonable claim to be made that such things may not live up to their promised (or hyped) emancipatory potential (Kreiss, Finn, & Turner, 2011). Rather, I think *Mario Factory* captured, perhaps more elegantly than many modern tools, what many developers wish existed for console game development. Put another way, *Mario Factory* was a missed opportunity for the craft of game development.

The patents describe, in great detail, a multiprocessor computing system with one central processing unit acting as the game's processing unit and the other operating the editing and operating system. A "model" is loaded into the editing system via a pluggable random access memory cartridge, much like the cartridges common in game consoles of the time. The model game provides users with a starting point from which the games can be created. The *Mario Factory* software provides an interface by which the user can then modify or extend the game model in ways that can make it appear to be a new game. Put simply, that system "permits a user to modify andy of the game's moving objects, background screens, music or sound effects." (Yamato et al., 1994a, pp. 54–55)

Mario Factory was, at the simplest level, a multiprocessor computer and integrated game console. As can be seen in Figure 2, the system was imagined to have output to a television or monitor. It could have various connected peripheral devices, including a keyboard, mouse, controllers, and audio line-in. The device was also capable of writing directly to a game console cartridge that could then be played on other game consoles. Based on the diagrams of the system (the controllers, in particular) and the description, it was intentioned to be coupled with the Super Nintendo Entertainment System (SNES), though not limited to that system only, as indicated in the documents:

> [T]he game program executing hardware in the presently preferred embodiment may be implemented by hardware currently being sold by Nintendo of America as the Super Nintendo Entertainment System (SNES). The present invention, however, is not limited to the Super NES related game program executing systems but rather may be used with alternative game hardware implementations. (Yamato et al., 1994b, p. 56)

Mario Factory, viewed broadly, was a new tool designed to allow for (co)creative content for console video games. It was also a complete suite of tools by which users could begin (co)creating games. Perhaps most importantly, the device was designed to not only target game developers but specifically those unfamiliar with the game development process.

The device was intended for creating not just derivative games but as a system by which new games could also be developed. Combined with a read-only memory writing system, it allowed individuals to "fabricate" or actually make their own games. The device and the underlying software were also intended to be used by individuals without particular expertise in game development and programming in particular. It was "designed primarily for users who are unfamiliar with computer program[ing] or video game creating methodology" (Yamato et al., 1994a, p. 53). This system would allow such users, through the interface of the software, to interactively construct games. The tool's "icon driven, interactive computing system ... permits a video game to be executed, stopped, edited, and resumed from the point where the editing began with the editorial changes persisting through the remainder of game play" (Yamato et al., 1994a, p. 53).

Nested throughout this icon-driven system were a range of tools designed to meet the needs of various users. Although some were envisioned to be "relatively unsophisticated elementary school students" the same system was designed to be useful for "sophisticated game developers" as well. The software nested below each editor an increasingly more detailed array of editing screens that allowed the user to customize the underlying display, graphics, and logics of the game's systems (Yamato et al., 1994a, p. 53).

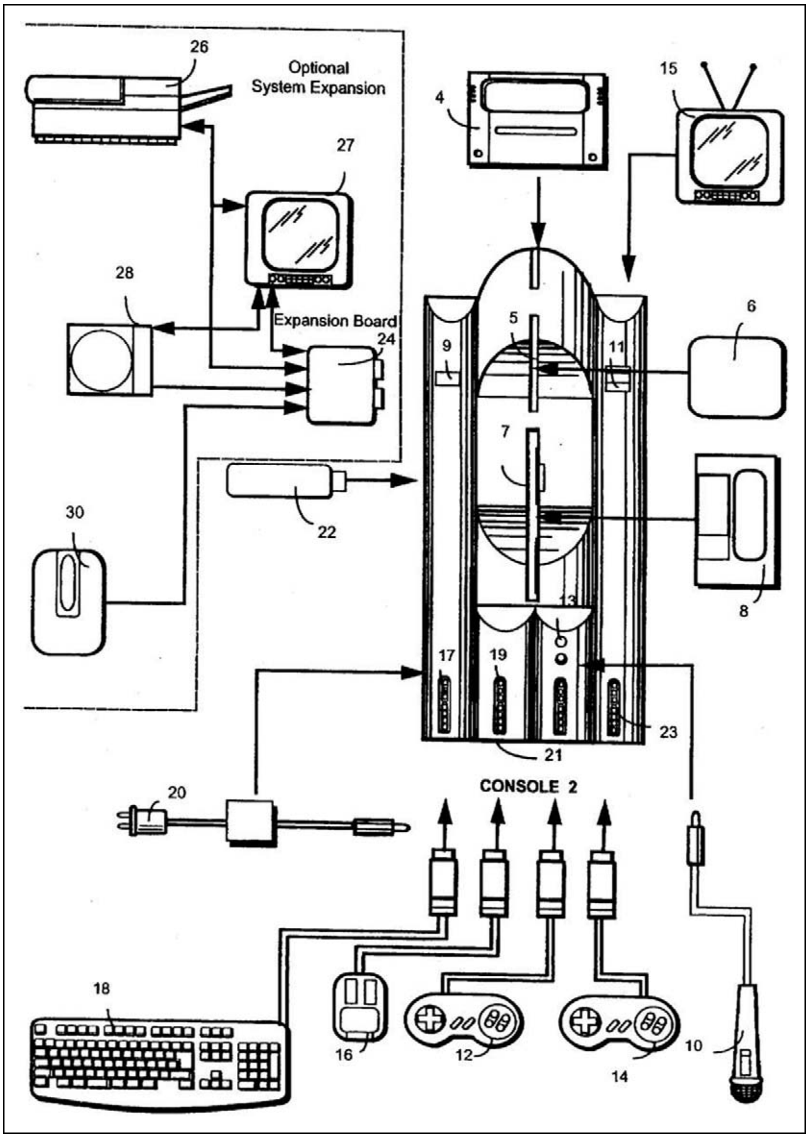

Figure 2. A diagram detailing the various components of *Mario Factory*.

As early as 1994, Nintendo was critically aware of the complexity associated with video game development practices and the kinds of interdisciplinary creative collaborative practice that is necessary for success. Although their patent hints at perhaps a declining collaboration between engineer, artist, and designer, it seems to be more about creating tools that foster effective collaborative practice between those groups.

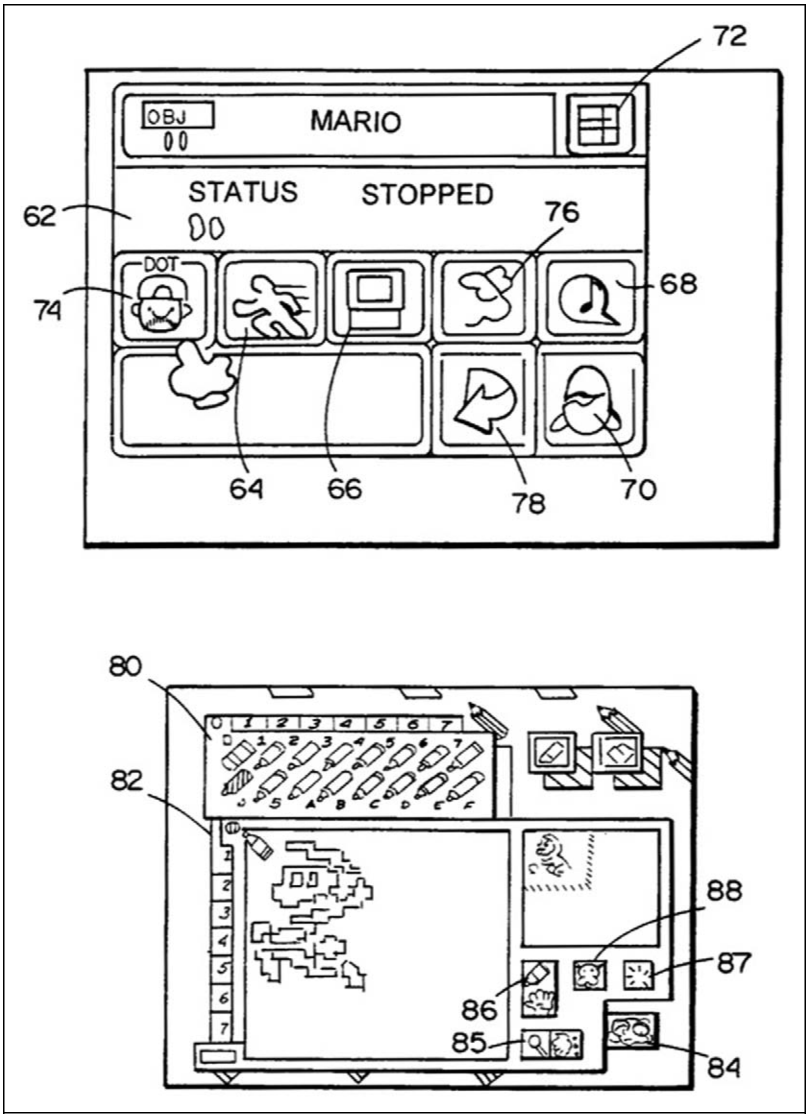

Figure 3. *Mario Factory's* pause/resume functions and image editing system.

The "strange" combination of artist, engineer, and designer has long been a particularly interesting aspect of video game development. It has long been cited by game developers and social scientists as being what makes the work of game development a particularly compelling area of inquiry. In early examinations of the creation of *Super Mario Bros.*, the close contact between game designer, artists, and

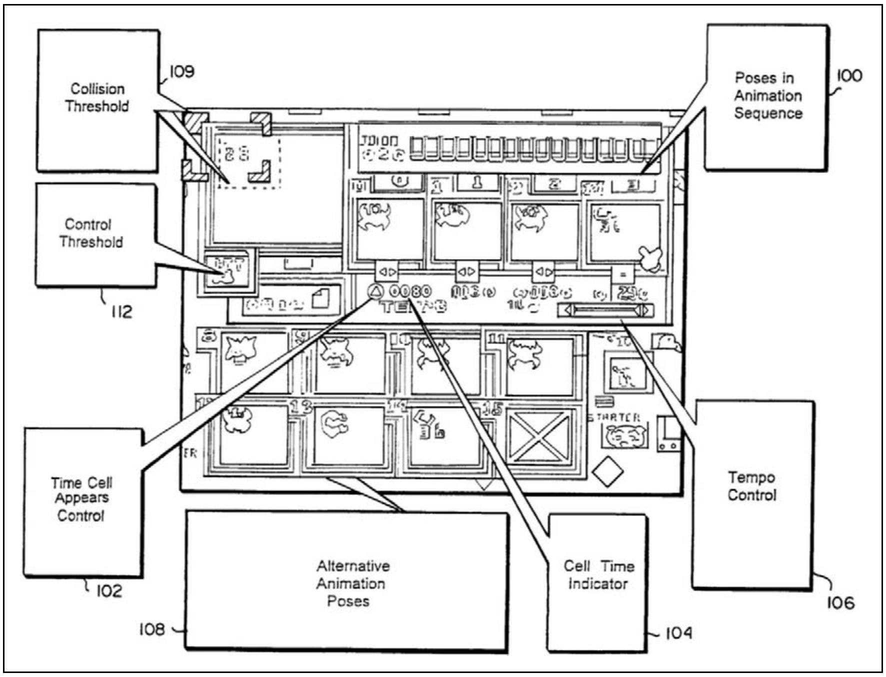

Figure 4. *Mario Factory's* animation editing system.

programmer is noted, with tools being created to act as intermediaries between these groups so that they can function with less "tedious" overhead.

As can be seen in the documents, there were numerous integrated editors for the creation and editing of different kinds of a game's artistic assets as well as design data. Pixel editors, which can be seen in Figure 3, with the requisite color palates to be used by an artist were available. Animation sequence editing, collision information, and the various player states could be created and edited within the interface pictured in Figure 4. This was, of course, a process that Nintendo had become intimately familiar since the development of the first Nintendo Entertainment System and the subsequent SNES.

It should be noted that in 1994 games developed for game consoles, like the SNES, were coded almost exclusively in assembly language. There was very little abstraction between the underlying computational system and how the code for a game was written. Especially in the case of game consoles, because they were less general purpose than other commonly available computer systems, there were far fewer general-use tools that addressed their very particular computation needs. Although a game console was a general use computational system, its highly specific construction and custom components made it a far more difficult platform to work with.

Although details on the production processes of early Nintendo games are difficult to come by, a handful of journalistic accounts offer insight into the practices

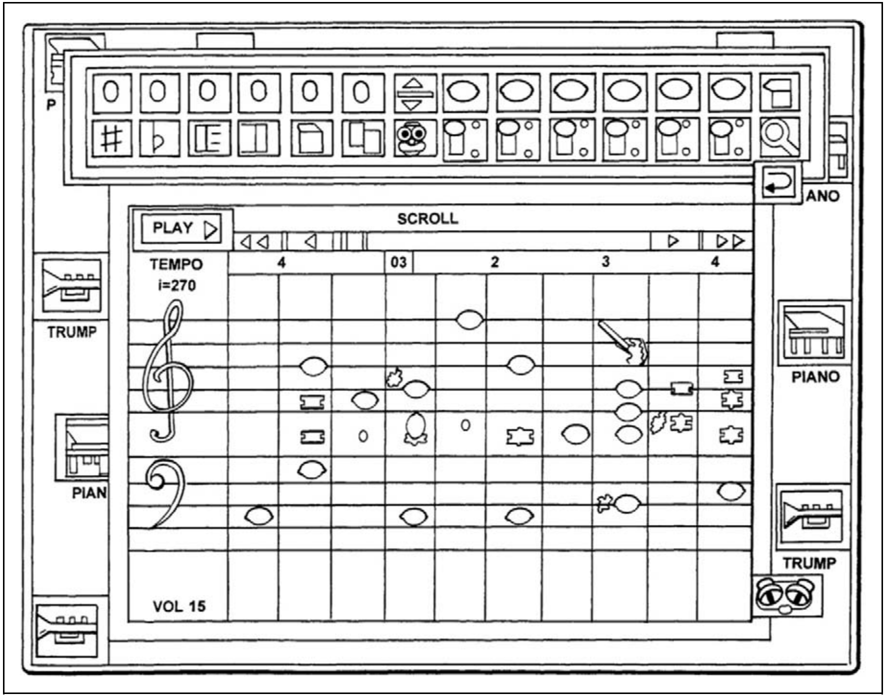

Figure 5. *Mario Factory's* MIDI audio editing system. MIDI = Musical instrument digital interface.

internal to the company. These practices, like drawing banks of images for use in a game, can be seen as influencing the design of *Mario Factory*.

> Technology eventually progressed to make some of the production stages easier. Originally Miyamoto had to paint each character. The colors in the painting were given numbers and the numbers were inputted into a computer, dot by dot. He showed programmers not only how the character looked but how it moved and what special traits it had (a bee, when hit, lost its wings but continued to stalk Mario; boats made out of skulls sank into a fire pit). The characters and their movements were written, line by line, as instructions in a computer program.
>
> Tools were developed to eliminated much of the tedious work. Diagrams and drawings were translated into computer graphics with technology called Character Generator Computer Aided Design (CGCAD). "Character banks" of images were stored along with the codes that described them. Movement, too, was now programmable from a bank of choices. (Sheff, 1993, pp. 53–54)

Many of these practices, deriving from the development of the original *Super Mario Bros.*, can be seen and felt throughout these documents. An "auto programmer" (Yamato et al., 1994a, p. 61) that allows the player to specify the pathway of an

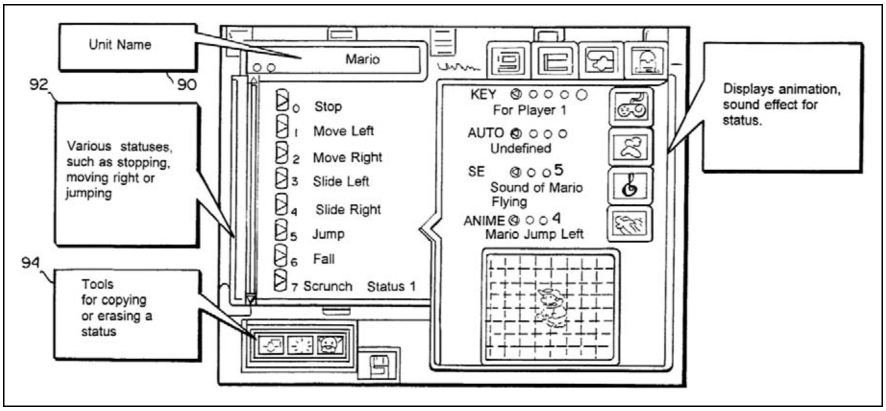

Figure 6. *Mario Factory's* model player state editor.

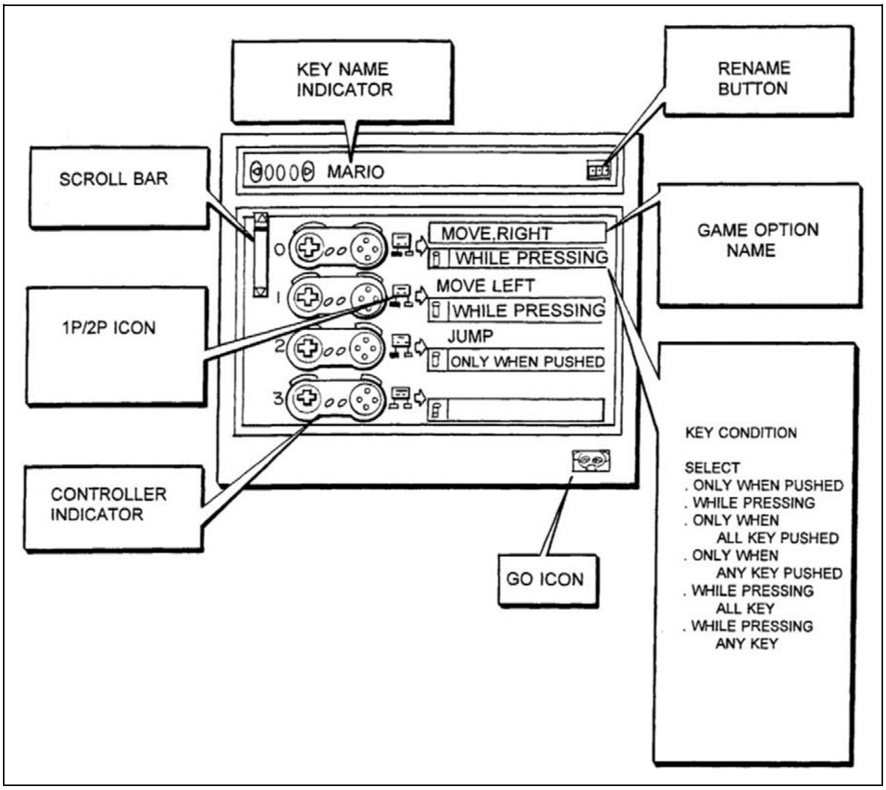

Figure 7. *Mario Factory's* player action editing system.

enemy throughout the game speaks directly to changing "how a character moved" in the above quotes. Of course, all of these editors come with limitations. For example, the auto programmer is limited to 64 nodes in the motion sequence. Pixel art is limited to 16 different colors (the hexadecimal numbers 0–F), and animations were limited to 16 frames (decimal numbers 0–15).

Mario Factory featured an integrated audio editing and composition system that was intimately tied to the underlying audio systems of the game console. Musical instrument digital interface (MIDI) audio could be assembled and modified throughout the development process using the editor in Figure 5. It was a direct interface not to some abstracted audio creation system but the hardware of the underlying game console's MIDI audio chipset.

Yet, it is not only the construction of basic game assets that make *Mario Factory* the interesting relic that it is, but also the underlying software that supports these various systems. Although the patents spend a significant amount of time discussing the various copy protection mechanisms integrated into the system, much is also made of the "unit"-based approach to game development. Further, it details that future editing systems could be added as new "models" were added to the system's repertoire.

In some ways, this is an early reference to the kinds of genres of games present in the market place. Although it specifically mentions role-playing games as a site for possible expansion, clearly the developers were aware of the linkage between certain game genres and associated gameplay styles that would enable user creativity without significant technological intervention.

Of critical importance to the system was the idea of "model game software." Encoded in this notion is one that there are various "genres" of video games. Examples such as role-playing games and "shoot-em-games" are given (Yamato et al., 1994b, p. 66). The use of these models would provide users the foundation for their games, while more "advanced" users could even construct new models for distribution. It was further presumed that these models may make use of their own custom editors, such as map editors for first-person shooting games or map and dialog editing systems for role-playing games. Thus, the model both articulated the underlying style of game and acknowledged that such models also encode and structure how the subsequent game must be constructed.

Programming has even been refigured in such a way to encourage accessibility, using "condition- and process-related operation tables" that are used to track the relational dynamics of the underlying game systems. Simultaneously, the system is designed for both "elementary school children" and "sophisticated game programmers" who are able to interface with *Mario Factory* through both "mouse" and "keyboard." Although the patents emphasize the importance of making the device accessible, they also repeatedly discuss that the same tool could be put to use by those more familiar with the game development process, even developing, internally, model software for use by their own development teams. Even during gameplay, "at any point in the game in which the user desires to change any displayed moving object character or portion of the screen background," the system can be stopped and put into an editing state (Yamato et al., 1994a, pp. 54-59), which can be seen in Figure 3.

The design screen mock-ups featured throughout the patents represent a hypothetical game design and development tool kit. In more than a decade, this kind of extensive set of tools has never been developed or released for general use. Even licensed video game developers do not have this kind of toolset by which to prototype or create video game systems. As video game development occurs now, much as it did at the time of the writing of the *Mario Factory* patents, "considerable program [/] designer activity is often required to modify a game under development in even very simple respects" (Yamato et al., 1994a, p. 53).

It is important to reiterate that *Mario Factory* was developed and never released or even publicly discussed. *Mario Factory* was a possible avenue for game development, with integrated tools aimed at users and developers that was never pursued. In 1994, nothing of this sort existed. Even now, the kinds of tools available for creating games on consoles is highly constrained. What makes *Mario Factory* so compelling was its complete breaking down of the barriers to entry for creating games for console systems.

Console development remains largely a dark art characterized by custom asset pipelines through which a game's art assets are processed for use in the game's engine. Custom-build scripts and compilers must be used and configured to work with software engineer's preferred integrated development environments (IDEs). Custom scripting languages must be integrated to each of those systems. Various middleware softwares that simplify or implement aspects of a game (i.e., physics or audio) must be combined with the various other systems that allow the process to function. This entire process is laborious in ways and for reasons that are far reaching (O'Donnell, 2011a). Console game development is not clear in the ways that *Mario Factory* attempted.

Which is not to say that personal computers have not benefited from approaches like those of *Mario Factory*. Systems like *Scratch* (MIT Media Lab, 2006), which do offer integrated editing systems, limited models, and integrated remixing, are linked strictly to their on-line platform. *Game Star Mechanic* (E-Line Media, 2010), another on-line system like Scratch, which does offer more game models to students, is still limited in the amount of customization that can be performed. *Mario Factory* possessed the ability, much like a video home system (VHS) of the era, to then bring your creations to others in a way that was simply playable on their SNES. Further, *Mario Factory* was always about game development, which Scratch is not specifically designed for (Hayes & Games, 2008). Even now tools like *Game Maker* (YoYo Games, 1999)*, Game Salad Creator* (Game Salad, 2007), *or Unity* (Unity Technologies, 2005) and game engines targeted at similar demographics do not offer the same kinds of integrated and highly platform-specific editors. Each relies heavily on external tools used to create the content seen in the game. However, with the exception of *Unity*, none of those systems can be used to author games on modern game consoles, let alone those of 1994.

The sole outlier in this story remains *Kodu Game Lab* (Microsoft Research, 2009) that was released for PC and the Xbox 360. *Kodu Game Lab* presented the user an interface through which they could construct game-like experiences. However, it lacked the ability to create experiences outside of its space and its primary goal was to teach young players procedural (programming) logic.

The Mangled Tool(kit)

Tools structure the roles in which users and their means of (co)creation are allowed to function. Users' tools are highly constrained by the sites and the methods with which they are allowed to engage with video game developers. This structuring significantly impacts three groups; users, game developers, and the game industry generally. For users, it primarily means that their role in the (co)creative possibilities are often not only constrained but also viewed as outside the interest or purview of game developers. User tools are also often viewed as a sort of second-order tool by professional game developers, designed for users but certainly not the tool of a "real" game developer. For developers it often means that their tools must be constructed in the process of working with new hardware. The tools provided to developers are far from this robust; thus, they create new tools that may or may not work as the device manufacturer assumed they would be used. In some cases, this is exacerbated by even hardware manufacturers not knowing how best to accomplish a given task with the hardware. For the industry more broadly, this means that significant time and energy is used in developing new tools with every generational change in hardware. It also means, that largely, the work of game development remains a mystery to users.

Mario Factory was, in effect, a DevKit[2] for the masses. This hints at a very different possibility than one that is currently experienced by game developers. DevKits were introduced so that game developers could create games for consoles, where the hardware differed significantly from that of personal computer's (PC). Nintendo developed technologies to bridge the gap between the PCs, where code was typically written, and the consoles, which ran the compiled code. The complexity of these devices has increased dramatically, coinciding with the complexity of consoles. These devices are cautiously guarded by game development companies and are only leased to licensed publishers and video game developers.

At the same time, however, DevKits and licensing are aspects of the game industry, which frequently disappear from the perception of developers once they have established themselves. Once a company has gained access to DevKits and software development kits (SDKs), they recede into the background despite the fact that they were once one of the major gatekeepers of access to industry networks.[3] In Latour's terms, these entities (or actants) of the workplace "sit in silence, as if they did not exist, invisible, transparent, mute, bringing to the present scene their force and their action" (Latour, 1999, p. 185). Perhaps even more important is that these actors connect with far-flung people and sociotechnical networks that further give it power.

Far too quickly, the developers allow themselves to forget just how difficult it can be to work among the structures of the industry.

The device itself encourages a particular kind of use. It directs the user to follow particular design and implementation paths that actually encode a level of knowledge about game development practice. At the same time, it does so in a way that is compatible or in line with the underlying platform. It encourages proper use and directs the kinds of practices that have been deemed productive, practical, and reasonable as illustrated in Figures 6 and 7.

Thus, *Mario Factory* is important, because it provides an interface into the underlying "platform" of the video game console. The idea that *Mario Factory* offers a friendly interface to the platform of the underlying device is important, as it mediates between the developer and all of the various baggage that goes along with those platforms, that is, hardware specificities, operating systems, programming languages, software development kits, supported data formats, and the various ways these systems layer together. In choosing platforms, "media creators ... simplify development and delivery in many ways ... [yet] work that is built for a platform is supported and constrained by what the chosen platform can [or cannot] do." (Montfort & Bogost, 2009, p. 3).

These devices are particularly implicated during the process of developing games. For most developers, tools are a hodgepodge of technologies and practices that are developed alongside the game they help construct. Put another way, tools are very much like the "experimental systems" that both enable and constrain technoscientific practice:

> An experimental system can readily be compared to a labyrinth, whose walls, in the course of being erected, in one and the same movement, blind and guide the experimenter. In the step-by-step construction of a labyrinth, the existing walls limit and orient the direction of the walls to be added. It forces us to move around by means and by virtue of checking out, of groping, of tâtonnement. He who enters a labyrinth and does not forget to carry a thread along with him, can always get back. (Rheinberger, 1997, pp. 74–75)

Of course, the irony with regard to game development is that rarely is anyone actually "carrying thread" along with them. The daily activity and large teams associated with most game development largely prohibits the kind of note and record keeping that in science lends itself to analysis. Although it might be possible to examine the data associated with a game's source code repository or asset tracking system, such data are difficult to find. What can be said throughout the process of developing a game is that oftentimes things do not go quite as planned initially. It is most important to examine those, "elements in the network [that] prove difficult to tame or difficult to hold in place," and how "vigilance and surveillance have to be maintained, or else the elements will fall out of line and the network will start to crumble" (Law, 1989, p. 114). It is this vigilance, surveillance, and fear of untame users that tends to limit the availability of the numerous devices and technologies that form the foundation of console video game development. Especially, as we examine technologies that (co)create video games, it is important to remember that "machines have no inertia of their own; like kings or armies they cannot travel without their retinues or impedimenta" (Latour, 1987, p. 250).

The further a project progresses along particular paths, the more complex are the interconnections between the systems. Each game development tool connects with other "retinues or impedimenta." We must consider the tools of developers, which partially determine what is possible and what is not to each. Put more simply, tools shape the creative process. Devices in the laboratory of scientists have long been examined as important "epistemic things" that shape the possibilities within the lab.

> In configuring and reconfiguring epistemic things, scientists meet with resistance, resilience, recalcitrance. Not anything goes. If there is construction, it is constrained. (Rheinberger, 1997, p. 225)
>
> The dance of agency, seen asymmetrically from the human end, thus takes the form of a dialectic of resistance and accommodation, where resistance denotes the failure to achieve and intended capture of agency in practice, and accommodation an active human strategy of response to resistance, which can include revisions to goals and intentions as well as to the material form of the machine in question and to the human frame of gestures and social relations that surround it. (Pickering, 1995, p. 22)

It is at these junctions where tools most often appear. Tools mediate those points of connection, and those points most frequently occur where artist, engineer, and designer intersect. Other points indicate limitations of the hardware or the underlying platform of a system. Tools encourage users in particular directions or explicitly disable the ability to perform tasks that have often found to cause issue with other aspects of a game's underlying systems. The tools both enable and constrain developers in certain ways.

Mario Factory was about making game development accessible. Perhaps even more importantly, it was about enabling development in ways that were found to be largely successful through time and experience for Nintendo. Many authors are quick to point at the numerous means by which users can become (co)creators in the worlds of video game development, and they are not at fault for attempting to more closely examine this hidden labor. However, it is difficult to consider the kinds of activities that are actually possible for users among the diverse activities of game development.

The hardware that runs a given platform can (dis/en)courage particular approaches to its use. What an engineer might assume about how to approach a particular issue may actually be better solved by another. *Mario Factory* through its user-interface design would encourage "appropriate" use of the hardware. Although this could also be used to the detriment of developers, it is certainly a level of direction that is often not found in the documentation that accompanies game development kits.

It is this user-friendly interface to a complex piece of technology and the dynamic processes that surround it that distinguishes this device. *Mario Factory*'s interface and integrated designer tools provide more explicit guidance to users as to how particular tasks might be accomplished. Furthermore, it exposed different kinds of interfaces through its computer mouse, keyboard, controller, and microphone that enable user interaction in ways that they find recognizable.

> Toolkits for user innovation and design are integrated sets of product-design, prototyping, and design-testing tools intended for use by end users. The goal of a toolkit is to enable non-specialist users to design high-quality, producible custom products that exactly meet their needs. Toolkits often contain "user-friendly" features that guide users as they work. They are specific to a type of product or service and a specific production system. For example, a toolkit provided to customers interested in designing their own, custom digital semiconductor chips is tailored precisely for that purpose - it cannot be used to design other types of products. Users apply a toolkit in conjunction with their rich understanding of their own needs to create a preliminary design, simulate or prototype it, evaluate its functioning in their own use environment, and then iteratively improve it until they are satisfied. (von Hippel, 2005, pp. 147–148)

Yet even now, the majority of game development tool kits favor design languages far removed from those of typical users. It seems reasonable to assume that an entire game could not be developed strictly with the use of the controller but that is by no means the extent of the factory. It comes equipped with keyboard, microphone, and numerous other peripherals that may prove useful for the user creator. The patent documents even provide a better sense of the plethora of skills necessary for the creation of a full game, something uncommon among the general population. Art, animation, sound effects, music composition, level design, scripting of behaviors, and even a graphical programming language are components of the proposed tool kit. Perhaps most importantly, the device comes equipped with the ability to write a user's creation to a cartridge, which would be playable on the corresponding console. The ability to create games in a design language familiar to gamers and share them with friends seems more like what has been termed part of "convergence culture." This is "embodied, for example in the work of the game modders, who build on code and design tools created for commercial games as a foundation for amateur game production" (Jenkins, 2006, p. 141).[4]

It is important for *Game Studies* to attend to the technological tools, expertise, and cultural spaces of game production, as they contribute significantly to the kinds of games that are produced.[5] In particular, the tools used to create a game have significant impacts on its underlying design and mechanics. For example, in examining the mechanics of *Gears of War* and its mechanics, "[i]nspried by the cover system of *kill.switch*, *Gears of War* combined a linear level structure with action sequences where the dominant strategy is to take cover and patiently create an effective combat tactic" (Sicart, 2008). At the same time, those cover locations actually had to be tagged specifically using the tools created by *Epic Studios*. The mechanics are often intertwined heavily with the systems that come together under the vague term "tools." Indeed, *Epic*'s tools have progressed to the point where they are a product in and of themselves, the *Unreal Engine*.

Even our models for analysis of games need to understand the relationship that game elements, such as, "object inventories," "interfaces," and "interactions" (Consalvo & Dutton, 2006), have with the tools used to generate a game. Many of the properties of those items, on screen information, and dynamics are specified by game designers not directly in "code" but using the intermediate tools that provide them with interfaces to the code that then interprets that information. Game developers often refer to this generically as "data." Art assets, data, and code ultimately combine to form the resulting technological artifact that is understood as a game.

Although many users may participate by playing a game and offering insight into how it might be made different via web forums (Banks & Humphreys, 2008), through user-interface MODs (Taylor, 2006), or through established SDKs for MODers (Nieborg & van der Graaf, 2008), or by constructing additional content for games like *The Sims* or *Little Big Planet*, this is not the same kind of convergent activity as remixing, editing, and uploading a video to YouTube, which, in a sense, *Mario Factory* was about. Writing in a web forum is a very different activity from remixing the tiles of one game title into another game with completely different mechanics or editing the graphical characteristics of a game element. Each of these activities and the possible technological interventions available to the (co)creator are quite different. Tools shape and constrain the activities of general users doubly, for they know no other interface into these systems.

Wither Mario Factory?

Thus, the question "Wither *Mario Factory*?" is a complex one. It is a question that ultimately asks, where are the user tool kits for video game innovation? More importantly, the focus needs to be on the capacity of these user tool kits; what do they allow or disallow? To what depth can the user (co)create? To what extent can they distribute their (co)creations? I ask these questions, because I believe they are important for users and for the video game industry more broadly. User tools are also frequently viewed as a sort of second-order tool by professional game developers, designed for users, but "certainly" not the tool of "real" game developers.

In this context, *Mario Factory* becomes a foil, a contingent direction, which (co)creative and user-centric technologies could have developed into a compelling space for user-generated designs and innovation. Instead, the space remains heavily patrolled and surveilled. Even web forums have admins and managers to control users. The web forum as a technology connects with other systems that socially and technologically shape the (co)creative process. This does not argue that (co)creative "development" can only exist with this kind of expansive game editing, remixing, and manufacturing device but that the options that avail themselves to users are different in each technological context. It is also imperative to examine the options available to the (co)creating user in each context. In what ways can users engage with the materials and arguments taking place in the development process?

Mario Factory represents a pivotal moment in the broader history of the video game industry, where the potential for (co)creative labor was examined, even patented, but ultimately never pursued. The implications are more important than they may appear on the surface. Robust user-facing tools like *Mario Factory* form the foundation for a new set of accessible tools in which users may engage with video games in various ways and at various levels. In many cases, users who begin working with remixing games may do so only at a cursory level but may find themselves working to learn the requisite skills to become more and more engaged in the practice of making games.

The emphasis on control, exhibited in how console video games are developed and distributed, speaks toward the anxieties that many console manufacturers felt, given the social and economic context in which they were introducing the devices. *Mario Factory* was developed during a time of high anxiety surrounding video games as a new media technology (Williams, 2003). Nintendo's determination to control production speaks to apprehension on the part of companies to allow the unpredictable user into the productive space, for they suddenly become, at least socially, culpable for the creations of their users.

The future of (co)creative labor in the console video game industry is ultimately linked to the possibilities of the surrounding tools and their ability to speak to the users' existing skills. Currently, this is not the case. Tools available to users and professional developers alike require specialized knowledge and significant time investment. A reassessment is necessary in light of how (co)creative production practice has changed the space of media (co)production. These models ultimately spring from a historical trajectory that placed secrecy as a foundational concern for developers, publishers, and manufacturers. Users have long been absent from the conversation, largely being considered a periphery aspect of video game development. However, ideas about how users fit into the production process have changed, and a return to the concepts central to *Mario Factory* that of the user accessible tools for (re)mixing, altering, and creating games should be returned to.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

Notes

1. Some might argue that my use of the word "structure" or "structuring" is contrary to the use of concepts drawing on actor-network theory (ANT), which Pickering and Rheinberger draw

on in their respective conceptual frameworks. It is often presumed that ANT examines, "the stability and form of artifacts ... as a function of the interaction of heterogeneous elements ... shaped and assimilated into a network," that networks and structure are somehow divorced. However, heterogeneous engineering as a vein within ANT specifically examines, "the way in which solutions are forged in situations of conflict," (Law, 1989, pp. 111–113), which I tend to call "structure" as short hand for those elements that shape, shift, and forge the creative process. ANT returns continually to the notion of stability. Thus, structuring occurs even within the actor network. Perhaps this use of the term comes closer to what some might call "protocols" or the "protocological field" (Galloway, 2004, p. 17). Regardless, the issue is one of control, conflict, and force in the realm of creative collaborative work.

2. The structure and role of licensing and DevKits as well as the preeminence of the console video game industry is a perennial area of analysis (Aoyama & Izushi, 2003; Dyer-Witheford & Sharman, 2005; O'Donnell, 2011b; Williams, 2002). It is not the attempt of this article to retread this particular issue, but instead to examine the relationship that these devices have with the game development process and ultimately how the actual release of a product like *Mario Factory* might have shifted users (co)creative possibilities on video game consoles.
3. In some ways, this is a fairly typical science and technology studies (STS) kind of argument that studying the context of video game development is as important as studying the artifacts produced. This kind of approach is difficult, however, because often game companies are unwilling or unable to provide the kind of access that would be necessary for these kinds of analysis. It is for this reason that I demonstrate, through this article, that we can actually talk about devices found in video game development companies even though they may be otherwise kept under nondisclosure agreement (NDA) or be obscured from the analysts' view.
4. Although many examine "convergence" as a kind of social process linking producers and users of media, this article reframes it as a sociotechnological issue linked to professional and production issues. In part, this is an attempt to think through what game development might look like if tools were designed to support a more "participatory" and user co-constructable model.
5. The "DevKit" is distinct from "development kits" as defined by some authors (Postigo, 2003). There is slippery and important language to keep in mind. The SDKs or software development kits are distinctly different though intertwined with DevKits. DevKits typically have accompanying SDKs. However, it is possible for companies to release SDKs without having DevKits. The hardware of the DevKit and access to documentation and other resources like on-line discussion forums are part of what distinguishes them from an SDK.

Game Engine References

E-Line Media. (2010). *Gamestar mechanic* [Browser-based Flash Application]. New York, NY: Author. Retrieved from http://gamestarmechanic.com

Game Salad. (2007). *Game salad creator* [PC, Mac Application]. Austin, TX: Author. Retrieved from http://gamesalad.com

Microsoft Research. (2009). *Kodu game lab* [PC Application, Xbox Live Indy]. Redmond, WA: Author. Retrieved from http://fuse.microsoft.com/projects/kodu

MIT Media Lab. (2006). *Scratch* [PC, Mac Application]. Cambridge, MA: Author. Retrieved from http://scratch.mit.edu

Unity Technologies. (2005). *Unity* [PC, Mac Application]. San Francisco, CA: Author. Retrieved from http://www.unity3d.com

YoYo Games. (1999). *Game maker* [PC, Mac Application]. Dundee, Scotland: Author. Retrieved from http://www.yoyogames.com

References

Aoyama, Y., & Izushi, H. (2003). Hardware gimmick or cultural innovation? Technological, cultural, and social foundations of the Japanese video game industry. *Research Policy*, *32*, 423–444.

Banks, J., & Humphreys, S. (2008). The labour of user co-creators: Emergent social network markets? *Convergence*, *14*, 401–418.

Consalvo, M., & Dutton, N. (2006). Game analysis: Developing a methodological toolkit for the qualitative study of games. *Game Studies*, *6*. Retrieved from http://gamestudies.org/0601/articles/consalvo_dutton

Dyer-Witheford, N., & Sharman, Z. (2005). The political economy of Canada's video and computer game industry. *Canadian Journal of Communication*, *30*, 187–210.

Galloway, A. R. (2004). *Protocol: How control exists after decentralization*. Cambridge, MA: MIT Press.

Hayes, E. R., & Games, I. A. (2008). Making computer games and design thinking: A review of current software and strategies. *Games and Culture*, *3*, 309–332.

Hibino, T., & Yamato, S. (1994). *Security systems and methods for a videographics and authentication game/program fabricating device*. Kyoto, Japan: Nintendo Co., Ltd.

Jenkins, H. (2006). *Convergence culture: Where old and new media collide*. New York: New York University Press.

Kreiss, D., Finn, M., & Turner, F. (2011). The limits of peer production: Some reminders from max weber for the network society. *New Media & Society*, *13*, 243–259.

Latour, B. (1987). *Science in action: How to follow scientists and engineers through society*. Cambridge, MA: Harvard University Press.

Latour, B. (1999). *Pandora's hope: Essays on the reality of science studies*. Cambridge, MA: Harvard University 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, MA: MIT Press.

Lowood, H. (2006). Storyline, dance/music, or PvP? Game movies and community players in world of warcraft. *Games and Culture*, *1*, 362–382.

Montfort, N., & Bogost, I. (2009). *Racing the beam: The atari video computer system*. Cambridge, MA: MIT Press.

Nieborg, D. B., & van der Graaf, S. (2008). The mod industries? The industrial logic of non-market game production. *European Journal of Cultural Studies*, *11*, 177–195.

O'Donnell, C. (2011a). Games are not convergence: The lost promise of digital production and convergence. *Convergence*, *17*, 271–286.

O'Donnell, C. (2011b). The nintendo entertainment system and the 10NES chip: Carving the videogame industry in silicon. *Games and Culture*, *6*, 83–100.

Pickering, A. (1995). *The mangle of practice: Time, agency, and science*. Chicago, IL: University of Chicago Press.

Postigo, H. (2003). From pong to planet quake: Post-industrial transitions from leisure to work. *Information, Communication & Society*, *6*, 593–607.

Rheinberger, H.-J. (1997). *Toward a history of epistemic things: Synthesizing proteins in the test tube*. Stanford, CA: Stanford University Press.

Sheff, D. (1993). *Game over: How nintendo zapped an American industry, captured your dollars, and enslaved your children*. New York, NY: Random House Inc.

Sicart, M. (2008). Defining game mechanics. *Game Studies*, *8*. Retrieved from http://gamestudies.org/0802/articles/sicart

Suzuki, T., Yamato, S., Koganezawa, N., & Ozaki, Y. (1994). *Video game/videographics program fabricating system and method with unit based program processing*. Kyoto, Japan: Nintendo Co., Ltd.

Taylor, T. L. (2006). Does wow change everything?: How a PvP server, multinational player base, and surveillance mod scene caused me pause. *Games and Culture*, *1*, 318–337.

von Hippel, E. (2005). *Democratizing innovation*. Cambridge, MA: MIT Press.

Williams, D. (2002). Structure and competition in the U.S. home video game industry. *The International Journal on Media Management*, *4*, 41–54.

Williams, D. (2003). The video game lightning rod. *Information, Communication & Society*, *6*, 523–550.

Yamato, S., Nishiumi, S., Suzuki, T., Hotta, T., & Fujiwara, K. (1994a). *Video game/videographics program fabricating system and method with superimpose control*. Kyoto, Japan: Nintendo Co., Ltd.

Yamato, S., Nishiumi, S., Suzuki, T., Nakamura, T., & Kimizuka, M. (1994b). *Video game/videographics program editing apparatus with program halt and data transfer features*. Kyoto, Japan: Nintendo Co., Ltd.

Author Biography

Casey O'Donnell is an Assistant Professor in the Department of Telecommunication, Information Studies and Media at Michigan State University. His research examines the creative collaborative work of video game 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 research examines issues of work, production, copyright as well as third-world and postcolonial aspects of the video game development workplace.