Loading… The Journal of the Canadian Game Studies Association
Vol 7(12): 59-80
http://loading.gamestudies.ca

The Programmer as Player: Uncovering Latent Forms of Digital Play Using Structuration and Actor-Network Theory

Matthew Wells
Faculty of Information, University of Toronto
mattwells.j@gmail.com

Abstract

Programmers create digital games, and players play them, or at least that is generally how we see things. But could programming also count as play? Are there scenarios in which programming is an integral aspect of digital play? This paper will make the argument that the boundaries we tend to draw between the digital game programmer and game player do not always apply, and that there are both historical and current areas of gaming where the roles are blended, and where the "player/programmer" engages in a diversity of play practices. The player/programmer emerged in the earliest days of mainframe computing, as programmers and engineers with the *Whirlwind* project at MIT leveraged the CRT screen – which at the time was an experimental add-on – to create display "hacks" that were deemed to be largely frivolous as compared to the more serious applications for which the computer was devised. This ethos of fun and experimentation persisted, leading to events such as the development of *Spacewar!* – a science fiction-themed shooter that was extremely influential in early gaming history – as well as the emergence of the hobbyist programmer/player in the microcomputer era (roughly from the mid-1970s through to the end of the 1980s). Today, the practices of the player/programmer are enacted within communities of game modders, and are also prevalent among Python programmers. This essay will employ both structuration theory and actor-network theory in order to define and delineate the various individuals, institutions, and technologies that incited and continue to sustain the role of the player/programmer.

Author Keywords

Digital game history; Spacewar; programming; coding; modding; hobbyist computing

Introduction

In the consumer electronics market, digital gaming occupies an increasingly prominent position within the larger category of "entertainment" products. Games and game consoles share prime store space with computers, DVDs and Blu Rays, CDs, cell phones, and e-readers. Media coverage of game-related events, including conferences and the release of highly-anticipated titles, has grown more substantial in recent years. In academia and in the industry, fundamental questions about digital games are being asked and analyzed, to various ends. All of this work

plugs into a larger discussion on gaming itself, and what criteria a specific set of practices must meet in order to qualify as a game, or as play. Sale and Zimmerman's (2004) treatment of this issue incorporates proposals made by no less than eight gaming scholars – including Huizinga's (1995) theory of play as an immersive, separate (from "ordinary life"), yet still rule-based practice; Suits' (2005) notion that gaming is a goal-based activity rendered inefficient (purposefully) by rules; and Crawford's (1984) conception of the game as a "closed formal system" that allows for player interactivity within an environment intended to produce conflict. These are foundational works that reflect a wide range of valuable research into digital games, but a common thread among them is a focus on the experiences of the game player/consumer. The gaming industry – the big corporations, the independents, the hardware engineers, the retail and online outlets, and so forth – also receives attention from scholars, of course, although development is not often associated with forms of play. Game development and play tend to be placed within different silos, however, this dichotomy is largely a conceptual invention. In the earliest years of gaming, and for many years thereafter, the vast majority of computer game players (as opposed to arcade or console gamers) also developed games; more accurately, they tended to modify existing games, and would then go on to distribute their new versions via various channels. This was not simply a matter of blending together two disparate hobbies. To play was to program for all but the most casual of computer users.

This paper will make the argument that our modern notion of what constitutes digital play should be expanded to include the practices of those who play via programming – a role I will refer to as the "player/programmer" – as well as those individuals and groups that engage in related practices, with "modders" being an important contemporary example (Sotamaa, 2007; Sotamaa, 2010). I will discuss how the first game-like programs emerged from within government and university-sponsored computer engineering projects, and how the ethos of game programming contrasted sharply with discursive practices from the same era in which the computer was presented as a practical problem solver (for example Licklider, 1960; Engelbart, 1962). Funding from government agencies in particular was contingent on developing machines that could solve the complex problems inherent in prosecuting a Cold War waged on a global scale (Edwards, 1997; Leslie, 1993). Games were developed by programmers interested in experimenting with the capabilities of the machines that they worked with by pushing them further than was generally allowable when they performed more utilitarian tasks. Display hacks, such as the *Bouncing Ball* program for the Whirlwind computer, were popular in the earliest years of computing, and these evolved gradually into more interactive projects such as TX-0's *Mouse in the Maze*, until finally there were programs such as *Spacewar!* that we would now recognize as games (Graetz, 1981; Hurst et al., 1989). I will argue that these early games inherited the practices and ethos of the display hack era, from coding to execution. For this reason, their work was largely viewed as frivolous, despite the sophistication of the code they were creating and modifying, and this served to dampen the influence they had within university campuses and other research institutions. Despite this, the player/programmer survived into the personal computer era, when BASIC became the language of choice for most users. As home computers became increasingly sophisticated, hobbyist game programming was sidelined, but with the emergence of modding, as well as new user-friendly programming languages such as Python, the player/programmer paradigm has persevered. By looking back to periods when the player/programmer was more clearly defined, however, we can come to better understand and promote the activities of today's modders and programmers.

In order to accomplish my goals for this essay, I will rely upon two similar, but somewhat disconnected sociological models. The first of these is structuration theory, as devised by Giddens (1984) and refined by later scholars such as Orlikowski (2000). In structuration theory, society is understood as being organized along a system of largely unspoken rules, which are continually reproduced and perpetuated as individuals and institutions engage with one another. Orlikowski extends this model by incorporating technologies as artefacts which assist in the building of structure, and focuses on various categories of enactments that users perform as they leverage such artefacts. As users engage with new technologies, according to Orlikowski (2000), they tend to either use them to support existing practices in which such technologies are implicated, or they choose to alter such practices. The second model is Actor-Network Theory (ANT), which was founded on the work of Latour, Callon, and Law, among others. ANT is a complex theoretical and sociological paradigm in which individuals and entities (including, somewhat controversially, objects) are viewed as actors linked in various sociological configurations. As Law puts it, ANT "is a way of suggesting that society, organizations, agents, and machines are all effects generated in patterned networks of diverse (not simply human) materials." (1992, p. 2) These networks, moreover, are sites of struggle in which configurations are never static, but rather are always in flux as the influence of specific actors and agents shifts over time (Law, 1992). There are several advantages to combining both of these models, the primary of which is the ability to understand the actions performed by ANT actors as enactments that build structure and normalize specific modes of behaviour. Such modes then go on to inform the growth and evolution of the networks within which they are embedded. With respect to early digital game history, those who programmed, played, and distributed games such as *Spacewar* were acting against the tendency to consider the computer as simply a tool to be used to solve complex mathematical problems. By making such a sharp distinction between problem solving and game programming, however, these same actors assisted in the enactment of digital gaming as a form of entertainment media. The structure they helped create would go on to cast gaming and game programming as frivolous pursuits, far removed from the more serious tasks that computers were expected to perform. The player/programmer does not disappear completely, however, as will be discussed later.

Structures and Actor-Networks

Before getting into the subject matter at hand, it is necessary to make the case that structuration theory and actor-network theory can be blended, and the benefits of such an approach. For structuration theory I will first discuss Giddens, and then I will shift to Orlikowski, as her technology-centric work better matches the approach I wish to take. For actor-network theory I will use works from Law, Latour, and Callon to provide the basic contours of the theory, before discussing how I believe it can complement structuration.

Giddens (1984) outlines structuration theory as a means by which to uncover the unspoken rules that govern the behaviours of individuals and institutions within any given society. This is not a deterministic model; Giddens stresses that he is not trying to suggest that "social life can be reduced to a set of mathematical principles" (p. 20). Rather, rule and practice exist in a symbiotic relationship, so that common practices within a society eventually become behavioural norms. As he explains:

> Let us regard the rules of social life…as techniques or generalizable procedures

> applied in the enactment/reproduction of social practices. Formulated rules – those that are given verbal expression as canons of law, bureaucratic rules, rules of games, and so on – are thus codified interpretations of rules rather than rules as such.
>
> (1984, p. 21)

The "social activities" of humans are therefore "recursive", to use Giddens' own phrasing: such activities are governed by rules, but they also help to normalize and perpetuate rules. Not every actor has the same capacity to reproduce and/or change rules, of course; Giddens stresses the importance of "institutions" in building structure, as opposed to individuals. But rules are only formed and reformed in practice. Giddens does not believe that structure exists outside of human activity, rather it is perpetuated and reproduced via the interactions among individuals informed by the structural information they received from earlier interactions.

Giddens does not stress the use of technology with respect to building and maintaining structure, which is where Orlikowski comes in. Orlikowski (2000) employs structuration in order to build an explanatory model with respect to how humans and technologies co-create one another, bridging the gap between technological determinism and social theories of technology. As she explains it, a finite set of uses are inherent in any technology, but humans acting as individuals and institutions enable one or more specific uses to emerge based on their needs. This process serves to influence the evolution of a given technology in order to better accommodate the selected uses:

> When humans interact regularly with a technology, they engage with (some or all of) the material and symbol properties of the technology. Through such repeated interaction, certain of the technology's properties become implicated in an ongoing process of structuration. The resulting recurrent social practice produces and reproduces a particular structure of technology use.
>
> (2000, p. 407)

Such social practices, however, are not developed in a vacuum. Rather, discourse helps inform potential practices:

> Use of technology is strongly influenced by users' understandings of the properties and functionality of a technology, and these are strongly influenced by the images, descriptions, rhetorics, ideologies, and demonstrations presented by intermediaries such as vendors, journalists, consultants, champions, trainers, managers, and "power'" users.
>
> (2000, p. 409)

Orlikowski argues that users are *enacting* the technologies they use, in that their interactions with these technologies enable specific elements of structure to emerge or re-emerge. As she puts it, "enacted structures of technology use…are the sets of rules and resources that are (re)constituted in people's recurrent engagement with the technologies at hand" (p. 406). Within such a

paradigm, applications of technologies emerge as a result of negotiations between individuals, institutions, discourses, and objects. While neither Orlikowski nor Giddens get into the issue of object agency, what does seem apparent is that objects express and build structure, at least when humans engage with them. In this way, then, there is something of a dialogue between both sides, with objects having the capacity to influence users towards certain ends.

This leads us to actor-network theory, which emerged out of concerns over the proper application of social factors to the study of science and technology. Latour (2005) directs much of his criticism towards the notion that such social factors are autonomous yet nebulous elements that are influential, but still disconnected from developments in science and technology. Callon indicates that the solution to this problem is to, when required, fully incorporate social elements into networks of agents and practices:

> If the hesitations, changes and evolutions that mark their [i.e. science and technology] development are to be understood, then interests, strategies, and power relationships which do not stop at the laboratory door must also be brought within the scope of analysis. In sum, though science and technology develop in some measure apart from the rest of the world, they are neither detached nor fundamentally different in nature from other activities.
>
> (1986, p. 19-20)

Law (1992) refers to the ANT solution to this problem as "heterogeneous engineering," a process he describes as follows: "[B]its and pieces from the social, the technical, the conceptual, and the textual are fitted together, and so converted (or 'translated') into a set of equally heterogeneous scientific practices" (p. 381). The translation work described by Law involves collecting these "bits of pieces" of practice and mapping them into networks of influence and control. The set of actors or "actants" that comprise a given network includes anything and everything that possesses any degree of influence, including individuals, institutions, ideas, and objects.

The notion of object agency is one of the most contentious elements of ANT. Latour (2005) defends it by noting that a given actor is never a source for action, but is rather a "moving target of a vast array of entities swarming toward it" (p. 46). Objects are conduits of agency, though they can also provoke actions that were not intended by their designers or developers. What ANT scholars do, then, is study how these agencies affect the configurations of networks. As Law puts it, the "core of the actor-network approach" can be described as:

> [A] concern with how actors and organizations mobilize, juxtapose, and hold together the bits and pieces out of which they are composed; how they are sometimes able to prevent those bits and pieces from following their own inclinations and making off.
>
> (1992, p. 386)

It is interesting to note that both structuration and ANT frame their respective models around metaphoric constructions linking individuals, institutions, and technologies. For Giddens and Orlikowski, structures emerge and are perpetuated when specific beliefs and practices become

ingrained in a given society. Orlikowski expands on Giddens work by allowing for technologies to influence, and be influenced by, these activities. For Latour and Law, networks emerge as various actors extend their influence out to other actants. These theories are not, of course, identical – ANT's claims to object agency have no true equivalent in structuration. But both theories complement each other by providing tools that the other, arguably, lacks. For structuration, ANT can provide more precise mechanisms by which to analyze specific structures. If we treat all elements in a given structure as actors, and then analyze how each actor behaves within its network, we could potentially identify how the emergent properties that come to define a given structure grow and evolve.

Conversely, structuration can potentially provide more contextual detail with respect to how actors communicate with one another in a network, particularly as we study how networks change over time. Orlikowski's (2000) categorization of enactments into three categories – *inertia*, *application* and *change* – will be particularly useful in this respect. According to this model, users will typically engage with a new technology either by altering their existing practices as little as possible (inertia), using the new technology to "augment or refine their existing ways of doing things" (application), or by using the new technology "to substantially alter their existing way of doing things" (change), with change obviously being the most dramatic approach (p. 421-423). These categories can be used, I believe, to inform our understanding of how certain actors behave in specific situations, and how their actions might have been different under certain circumstances. Application and change enactments will be particularly important in the discussion to follow.

Early Gaming

Digital gaming emerged in the 1950s and 1960s within the research institutions that produced the earliest mainframe computers. As will be discussed in more detail below, many of these mainframes were built for military purposes, as the Cold War raged and scientific research was funded heavily by concerned governments and militaries. Assemblages of researchers and equipment formed within universities such as MIT, the University of Pennsylvania, and Cambridge, as well as independent institutions such as Brookhaven National Laboratory (Edwards, 1997; Leslie, 1993). It was within these clusters that the first games emerged. The title of "first" computer game is often given to *Tennis for Two*, a *Pong*-like two-player ball bouncing game designed at Brookhaven by nuclear physicist William Higinbotham in 1958 for the Donner Model 30 analog computer (Brookhaven National Laboratory, 1981/2011). It seems likely, however, that A. S. Douglas' *noughts and crosses* (tic-tac-toe), created in 1952 for the EDSAC at the University of Cambridge, is more deserving of the honour (Winter, n.d.). From an ANT perspective, however, neither of these efforts truly succeeded in altering larger networks related to the development of computing technologies and the practices associated with them. They appear to both be one-off creations, without any serious influence. They may be contrasted with *Spacewar*, which was tremendously influential both as a game and as a locus for player/programmer practices.

Spacewar may seem like a rather rudimentary arcade-style shooter from our perspective, but at the time it was novel and innovative. The game involved two players that controlled "spaceships" displayed on a CRT screen, with the object being to shoot your opponent with torpedoes while also negotiating a gravity well centred on a star in the middle of the playfield.

Spacewar was conceived of and designed by Steve Russell, J. Martin Graetz (who wrote an account of the game's origins in a 1981 issue of *Creative Computing* magazine) and Wayne Witanen (though Witanen left school before the game was finished), a group of computer "hackers" working as programmers for various researchers at MIT and Harvard University. As discussed by Levy, the term "hacker" had emerged out of MIT's student-run *Tech Model Railroad Club* (TMRC), with certain practices being labelled as "hacks":

> "[A] project undertaken or a product built not solely to fulfill some constructive goal, but with some wild pleasure taken in mere involvement, was called a 'hack'…The most productive people…called themselves 'hackers' with great pride."
>
> (1984/2010, p.10)

The *Spacewar* developers were members of the TMRC, and helped steer the group towards the pursuit of computer hacks. This hacker subculture, then, had been enacted and re-enacted for some time before the arrival of the technologies used to make the game.

It is difficult to underestimate *Spacewar's* impact on computer gaming and on computer history in general (see Barton & Loguidice, 2009). Stewart Brand, the first editor of the *Whole Earth Catalog*, wrote a seminal piece on the game for *Rolling Stone* magazine in 1972. A full decade after its creation, the game had spread far beyond the MIT campus. Brand quotes renowned computer scientist Alan Kay as stating that "the game of *Spacewar* blossoms spontaneously wherever there is a graphics display connected to a computer." (1972, p. 53) *Spacewar* began the trend of science fiction games, which would later give rise to *Asteroids*, *Missile Command*, *Star Raiders*, and other arcade and console classics. But it was on the computer that *Spacewar* was truly at home, serving as a focus for both intense play and programming.

Why, then, was *Spacewar* so popular and so influential as compared to *Tennis for Two* and other early digital games? It was, of course, an extremely sophisticated program that seems to have captured the imaginations of those who played it. Beyond intrinsic qualities, however, we have to recognize *Spacewar's* place within a network of computing technologies that dates back to the immediate postwar period. As will be discussed momentarily, these computing technologies served both critical experimental and military purposes, incorporating design features that would be available in few rival machines for many years to come. We also have to recognize the ways in which the TMRC culture present at MIT helped to engender a playful attitude towards technology. In order to understand *Spacewar's* importance and influence, then, it is critical to situate the game within the larger context of both the network of actors within which it was embedded, as well as the enacted structures that informed its design.

Spacewar was programmed on a PDP-1, a transistor-based minicomputer donated to MIT by Digital Electronics Corporation (DEC), a recently-created business enterprise started by two former MIT computer scientists. The PDP-1 was the third in a line of large-scale mainframe/mini computers dating back to the final years of the Second World War. The first in the line – a machine that would come to be called the *Whirlwind* computer – was commissioned by the U. S. Office of Naval Research to MIT's Servomechanics Laboratory, and was originally intended to serve as a general purpose flight simulator. Eventually, the simulator aspect was dropped, and

research and development focused solely on the construction of a computer along the lines of the EDVAC machine that was being built at the University of Pennsylvania for the Ballistics Research Laboratory (Redmond & Smith, 1990). Due in part to this tumultuous early history, the Whirlwind project became a site for innovation and experimentation. It functioned in part, then, as a disordered network, "a network of electronic components and human interventions," to borrow Law's description of a physically and conceptually broken television (1992, p. 6). As a technological artefact, digital computers still had little in the way of structure that could mandate specifically how they could or should be designed. It was perhaps for these reasons, then, that Whirlwind was the first computer to employ a CRT unit – actually, several different units – for display purposes (Redmond & Smith, 1990).

It needs to be stressed that it was not obvious that a CRT display could be used in conjunction with a digital computer. And, in fact, with respect to Whirlwind, the display was clearly meant to be of secondary importance. Note the language used in this internal summary report from 1949 to describe the display:

> The display equipment now in use with WWI is intended primarily for demonstration purposes. It gives a qualitative picture of solutions to problems set up in test storage, and it illustrates a type of output device that can be used when data are desired in graphical rather than numerical form.
>
> (Servomechanisms Laboratory, 1949, p. 29)

The display here is clearly conceived of as an experimental device, added to show off the capabilities of Whirlwind outside of its primary work as a data processor. Norm Taylor, an engineer who worked on Whirlwind, presented a history of Whirlwind's displays at the SIGGRAPH conference in 1989 and essentially confirmed this perspective:

> Keep in mind we were not trying to build a display here; we were building a computer. All we used the display for was testing the various parts of the system so displays were ancillary completely to the main event.
>
> (Hurst et al., 1989, p. 22)

Such enactments, to use Orlikowski's categories, could be considered applications of the CRT display, in that they leveraged the technology as a means to simply monitor activity elsewhere in the system. Because Whirlwind was already a locus for improvisation, however, it did not take too long before more interesting uses – what Orlikowski would call change enactments – were devised. Taking advantage of Whirlwind's capacity to quickly solve complex differential equations, its programmers developed a program they called *Bouncing Ball*. Initially this was simply a program that showed a dot moving across the screen and bouncing along the floor. As Taylor explains, however, it was not long before an interactive element was added:

> A little later [*Bouncing Ball* developers] Adams and Gilmore decided to make the first computer game, and this was also in '49…You see that the bouncing ball finds a hole in the floor and the trick was to set the frequency such that you hit the hole in the

> floor. This kept a lot of people interested for quite a while and it was clear that man-machine interaction was here to stay. Anyone could turn the frequency-knobs.
>
> (Hurst et al., 1989, p. 21)[1]

Graetz, in his article on *Spacewar*, cites *Bouncing Ball* as an inspiration for their work, and reiterated its utility as a demonstration program:

> When computers were still marvels, people would flock to watch them still at work whenever the opportunity arose. They were usually disappointed…The mainframe, which did all the marvellous work, just sat there. There was nothing to see. On the other hand, something is always happening on a TV screen…On MIT's annual Open House day, for example, people came to stare for hours at Whirlwind's CRT screen. What did they stare at? *Bouncing Ball*.
>
> (Graetz, 1981)

Such demonstration programs continued to be produced on the TX-0, the transistorized successor to Whirlwind. The TX-0 was a product of MIT's Lincoln Laboratory, a facility that solidified the relationship between MIT's engineering units and the U.S. military that had been established in the Second World War and gained new strength at the outbreak of the Cold War. The Whirlwind project was initially supported by the U.S. Navy, as we have already seen, but this relationship soured somewhat in the immediate postwar period. The Air Force, however, having been designated as an autonomous service in 1947, was looking to extend its power and reach via projects such as a national early-warning air attack radar system. This system, which would come to be known as the Semi-Automatic Ground Environment (SAGE), was built out of the Whirlwind computer when naval funding dried up (Redmond & Smith, 1990; Redmond & Smith, 2000). Lincoln Laboratory, originally designated "Project Lincoln", was the institutional foundation for SAGE and later Air Force-related projects, a relationship that continues to this day.

The TX-0, then, inherited many of the technologies, practices, and personnel associated with Whirlwind. This included the CRT display hardware, as well as the ethos of experimentation that encouraged the development of *Bouncing Ball*. TX-0's most well-known display demonstration was an interactive program called *Mouse in the Maze*. Levy summarized its basic functionality as follows:

> The user first constructed a maze with the light pen, and a blip on the screen representing a mouse would tentatively poke its way through the maze in search of another set of blips in the shape of cheese wedges. There was also a 'VIP version' of the game, in which the mouse would seek martini glasses. After it got to the glass, it would seek another, until it ran out of energy, too drunk to continue.
>
> (1984/2010, p. 47)

There is clearly a bit more character to this game when compared to *Bouncing Ball*. It is playful, and perhaps even whimsical, as well as being mischievous with the drunken "VIP" version. This stands in contrast to the sober, serious uses for which the computer was intended. This is a point worth emphasizing. From an actor-network perspective, the power to develop and inscribe new technological forms was largely in the hands of the American military divisions that funded new and existing projects. The scientists and engineers that designed and built the military's machines were more closely connected to the computers themselves, however, thereby providing them with opportunities to make their own inscriptions. From a structuration standpoint, moreover, the computers themselves were flexible enough as technological artefacts to accommodate both sanctioned and unsanctioned enactments. The computer was gradually becoming a site of contestation over meaning and structure.

Spacewar, then, can be viewed as a culmination of several years of enactments in which mainframe CRTs were put to uses beyond the mere practical. This is a heritage that Graetz fully recognizes, as we have seen. The computers that served as platforms for *Spacewar* and its antecedents, however, had been designed and built to solve serious tasks. Without funding from the U.S. Navy and Air Force, and without an institution like MIT to support research via this funding, there would have been no *Bouncing Ball* or *Mouse in the Maze*. We cannot overlook, moreover, the fact that the PDP-1 for which *Spacewar* was developed as a commercial computing platform. As a result, *Spacewar* could be easily distributed to other settings, as Alan Kay noted in Brand's article. Critically, this put the game in the hands of many fellow programmers. As Brand describes it, many of *Spacewar's* players went on to reprogram it, digging into the code to change the rules of the game:

> Within weeks of its invention *Spacewar* was spreading across the country to other computer research centers, who began adding their own wrinkles. There was a variation called *Minnesota Hyperspace* in which you kept your position but became invisible; however if you applied thrust, your rocket flame could be seen.... Score-keeping. Space mines, Partial damage - if hit in a fin you could not turn in that direction. Then '2½-D' *Spacewar*, played on two consoles…Adding incentive, MIT introduced an electric shock to go with the explosion of your ship. A promising future is seen for sound effects.
>
> (Brand, 1972)

With the advent of networks like ARPANET, such variants could easily be transmitted across communications lines to whoever wanted to download the game, play it, and redesign it. *Spacewar*, then, was not a uniform product. While the original game was created by Russell and Graetz, other player/programmers were hacking at its code, enacting new versions of the game.

Despite all the time and effort that these player/programmers were putting into *Spacewar* – and the programming would have been particularly difficult on such old systems – *Spacewar* was still linked quite strongly with its display hack heritage. Like Graetz and the rest of the game's original programmers in the TMRC at MIT, moreover, these individuals enjoyed casting themselves as rebellious outsiders, sneaking into the computer labs once all the serious-minded folks had gone home.

Brand's article is suffused with this ethos. He describes the *Spacewar* fans he engaged with as "Computer Bums" (in capital letters), and here he once again quotes Kay, who describes the "standard" Computer Bum as follows:

> About as straight as you'd expect hotrodders to look…A true hacker is not a group person. He's a person who loves to stay up all night, he and the machine in a love-hate relationship…They're kids who tended to be brilliant but not very interested in conventional goals.
>
> (Brand, 1972)

The reference Kay makes to "conventional goals" is critical, as it implies that the work the Computer Bums perform is somehow not-conventional. But what exactly is conventional computer work? As it turns out, there were other programmers and computer scientists who were attempting to answer that very question, and their answers still influence how computers and the assemblages they attract are conceived of and enacted.

Engelbart and the Augmentation of Human Intellect

Douglas Engelbart was a renowned computer scientist, well-known now for his tech demo video that has since become known as the "Mother of All Demos." Filmed in 1968 (with the help of Stewart Brand), the system Engelbart was demoing was known as the oN-Line System (NLS), essentially a collection of software tools designed to enable collaborative development of documents and other data. The NLS was the culmination of many years of research conducted by Engelbart and his colleagues, and the demo introduced many of the elements of computing that have since become mainstream, such as the mouse, the windowed display, and hypertext (Bardini, 2000). Yet Engelbart was only indirectly focused on introducing such innovations. His true aim was the augmentation of human intellect, as he so often put it:

> By 'augmenting human intellect' we mean increasing the capability of man to approach complex problem situations to gain comprehension to suit his particular needs and to derive solutions to problems…Man's population and gross product are increasing at considerable rate, but the complexity of his problems grows still faster, and the urgency with which solutions must be found becomes steadily greater…Augmenting man's intellect in the sense defined above would warrant full pursuit by an enlightened society if there could be shown reasonable approach and some plausible benefits.
>
> (1962, p. 1)

While the military sought computing technologies to aid them in certain tasks, Engelbart conceived of a much more ambitious agenda: the augmentation of human intellect, with the ultimate goal of solving the world's most complex problems. These ideas were not entirely new, as Licklider (1960) had promoted the concept of "man-computer symbiosis" a few years earlier. But Engelbart expanded significantly upon this earlier work in terms of the complexity of his arguments.

If Engelbart had not pursued his goals by developing an entirely new computing platform, his influence over the network in which he implicated himself would likely have been minimal. As we have already seen, machines were the primary loci around which actants drew their strength. *Spacewar*, for example, would not have had any influence across this network had it not been coded into the PDP-1. Tangible artifacts were required to create powerful actants. Fortunately for Engelbart – then a professor at Stanford – his project attracted the attention of the U.S. Air Force, which, as we have seen, had already been involved in MIT's computer engineering efforts. The project also received funding from the U.S. Department of Defence's *Advanced Research Project Agency* (ARPA), which was created in the wake of Sputnik as a means to boost technological research and development (Bardini, 2000).

The NLS system that Engelbart's team eventually developed was quite unlike any software system that had been built up to that time. While NLS is largely remembered for its visual and tactile qualities, as discussed above, underlying all of it was an architecture in which pieces of information were stored and linked in often complex configurations. The following passage, in which a portion of the 1968 demo is described, offers a glimpse of its capabilities:

> In the demonstration, he accessed files from the computer's memory and displayed them on a large screen in the auditorium. Then Engelbart collapsed the written textual descriptions into a series of one-line headings (a basic outline). A click on the mouse button would then expand the headings into the larger text. Another command displayed a graphic representation of how the various documents were linked in the system.
>
> (Barnes, 1997, p. 23)

In the end, NLS never quite caught on, and many of its designers went to work at Xerox PARC after becoming dissatisfied with the direction that Engelbart was taking his research. There they joined the *Xerox Alto* research team, where they would go on to create a system much like NLS, but with an interface that was much more user-friendly. Their work would go on to inform the design of Apple's early operating systems (Bidini, 2000). Engelbart, then, played a substantial role in determining the future of personal computing. As Bidini explains it:

> He [Engelbart] articulated a vision of the world in which these pervasive innovations [i.e. NLS] are supposed to find their proper place. He and the other innovators of this new technology defined its future on the basis of their own aspirations and ideologies. Those aspirations included nothing less than the development, via the interface between computers and their users, of a new kind of person, one better equipped to deal with the increasing complexities of the modern world. In pursuing this vision, they created the conditions… that prescribe the possibilities and limits of the technology for the users of personal computer technology today.
>
> (2000, p. 1-2)

The last sentence of the above passage is perhaps the most telling. By focusing so exclusively on

augmenting intelligence, Engelbart helped to impose a specific value system on computers and related digital technologies. The innovations that he introduced – the mouse, windows, etc. – were mapped to this value system, and perpetuated it as they became more popular. The computer as conceived of within this model was meant to play the role of assistant to humans tackling complex problems. Programs that did not solve what were considered to be real-world problems – display hacks and games, for example – were not worth considering.

Engelbart's value system was not the only one competing for prominence within the same network of actors. A major complementary field emerging at the same time was artificial intelligence (AI). Rather than using computers to augment human intelligence, AI was more concerned with developing *machine* intelligence. John McCarthy (2001), who is considered one of the founders of AI, defined it as follows: "it is the science and engineering of making intelligent machines, especially intelligent computer programs" (p. 2). He then defines intelligence as follows: "intelligence is the computational part of the ability to achieve goals in the world" (p. 2). There are a variety of methods in AI by which to pursue the goal of machine intelligence, such as artificial neural networks, and natural language processing routines. Marvin Minsky, another pioneer in the field, envisions intellect as consisting of countless disparate processes directed by a single conscious agent (Ford & Hayes, 1998). But ultimately, as with Engelbart's work, the focus was on using computers to solve problems, or "to achieve goals in the real world." AI has different priorities as compared to Engelbart's work, but the ultimate goal is still to make computers and computing "useful".

From a structuration perspective, the structures enacted by Engelbart's team and early AI researchers helped to shape an understanding of the functionality of the computer that is still with us today. We tend to think of computers now as instruments through which we accomplish specific tasks via software platforms such as Microsoft Office and Apple's iMovie. This is also how computers are generally advertised to us. On Apple's website, for example, for their iMac line of desktop computers they advertise the fact that "every new Mac comes with better-than-ever versions of iPhoto, iMovie, GarageBand, Pages, Numbers and Keynote. So you can be creative and productive right from the start."[2] Microsoft, moreover, claims that it is "the worldwide leader in software, services and solutions that help people and businesses realize their full potential."[3] It should be noted that both Macintosh and Windows computers are both quite capable of playing games, a quality which is advertised in some materials. But when marketed to a mass audience, we are presented with machines that ostensibly help us to achieve specific personal and professional goals.

This is not to say that either Engelbart or McCarthy had a direct influence on either Apple's or Microsoft's marketing strategies. But the research and development projects that they directed would go on to inform structural norms that continue to resonate. Their work was particularly influential as it happened at a time when digital computing was still in its formative years. From an ANT perspective, Engelbart, MacCarthy, and fellow researchers were able to arrange their network(s) in such a way as to largely shut out the role of the player/programmer. If a given program did not serve any tangible purpose, it was labelled as a "hack", or simply a frivolous distraction. *Spacewar* was not designed to augment human intellect, nor was it supposed to solve the world's problems, and therefore it could not be considered as a serious software project, despite its sophistication.

As discussed earlier, *Spacewar's* player/programmers embraced their roles as outsiders, and did not attempt to plug their work into the goal-oriented paradigms envisioned by Engelbart and his peers. If circumstances had been different, however, and had *Spacewar's* fans advanced an agenda that promoted and defended their practices and values, our present understanding of digital gaming might have been substantially altered. We might, in fact, have considered programming to be a fundamental aspect of the gaming experience. Instead, game programming generally tends to be treated more as a means to an end, a practice enacted by developers to create finished products.[4] This goal-oriented model of game development hews closely to the paradigm of the computer as problem solver, even if the problems faced by the typical game company are largely financial in nature. In keeping with the work of Engelbart, McCarthy, and other like-minded computer scientists, game programming is now largely solution-oriented. Once a proper solution has been generated – that is, once a game concept has been fully realized – the program(s) responsible for it are considered to be successful. The idea of playing around any further with the code seems rather illogical. If errors are uncovered post-release, of course, patches to the original code are generally provided. But "working" code is not meant to be tampered with.

The Player/Programmer in the Microcomputer Era

The emergence of the personal computer in the 1970s is one of the pivotal events in the history of digital computing technologies. Kafai (1995), argued that the "microcomputer" – a preferred term at the time – gave rise to the first hobbyist programmers:

> In the wake of the advent of inexpensive microcomputers in the late 1970s came the first wave of nonprofessional programmers. Children, older students, teachers, and computer hobbyists took to the keyboard to find an experience that nobody had been able to have in previous generations.
>
> (1995, p. ix)

This claim is not quite accurate, as there were in fact nonprofessional programmers prior to the personal computer era. Access came largely through schools; DEC in particular had managed to get minicomputers from their PDP line into numerous educational institutions, with the PDP-8 proving the most popular:

> The PDP-8 is sometimes described as the Model-T Ford of the computer industry because it was the first machine to be mass-produced and was easy to maintain. Initially priced at $18,000 – including Teletype terminal – it was cheap enough for almost any institution to afford. Small colleges and large public school systems were some of DEC's best customers for the system.
>
> (Johnstone, 2003, p. 61)

Here, then, we see new network configurations being built. DEC forged connections with schools and colleges, thereby creating a physical presence in these new environments via their PDP machines. These linkages would then prove useful to DEC employee David Ahl. Ahl, who

would go on to found *Creative Computing* magazine, one of the primary hobbyist computing journals of the 1970s and 1980s, was a major proponent of hobbyist programming, and encouraged schools that operated DEC computers to send him programs – and in particular, games – written by their students (Anderson, 1984). In 1975 he convinced DEC to publish a book entitled *101 BASIC Computer Games* which provided the source code for a huge variety of simple games that could be keyed into most standard versions of the BASIC programming environment (Ahl, 1975). In the preface to this work, he makes the following observation: "the educational value of games can be enormous – not only in their playing *but in their creation*" (p. 7; emphasis added). Ahl was attempting here to enact change as he made the argument that game programming and playing could lead to positive educational outcomes. By doing this, the emphasis was placed less on playing than it was on coding, though Ahl does not consequently diminish the value of play, noting the following:

> Most educators agree that games generally foster learning by discovery… Newton's second law is probably the furthest thing from the mind of a person sitting down to Play ROCKET. However, when the player finally lands his LEM successfully on the moon, the chances are very good that he has discovered something about gravity varying inversely with the mass of the LEM and the distance from the moon.
>
> (1975, p. 7)

Ahl's vision of the player/programmer would have significant influence into the microcomputer era. Due in part to the popularity of his BASIC book, new books and magazines full of type-in programs became a staple of 1970s and 1980s hobbyist computing.[5] The player/programmer had become legitimized via the discourse of education as expressed by hundreds of mainstream texts. Moreover, new computers were being developed in response to the emergence of the hobbyist player/programmer. Commodore's machines, such as the VIC-20 and Commodore 64, would boot up immediately into Commodore BASIC, allowing the user to build BASIC programs right away (Commodore Business Machines, 1980). The BBC introduced a line of personal computers as part of their "Computer Literacy Project," with a heavy emphasis on programming ("The BBC Microcomputer and me," 2011). These efforts were not all directed at game programming, but these machines all boasted graphics and sound capabilities that were absent from products such as the IBM PC, which were not meant to be learning machines. These hobbyist actants – the programmers, the books, the computers, etc. – were enacting new structures related to the role of the microcomputer, making it out to be something of a playground for amateur coders. The pedagogical approach taken by Ahl and others also emphasized the "fun" aspects of learning to code, meaning that games played a key ongoing role in this paradigm.

As computers evolved and grew more complex, however, the hobbyist books and magazines largely disappeared. When *Compute!*, perhaps the most well-known hobbyist magazine of the era, decided in 1988 to stop including type-in programs in each issue, editor Gregg Keizer explained the rationale behind the decision as follows:

> As computers and software have grown more powerful, we've realized it's not possible to offer top quality type-in programs for *all* machines. And we also realize that *you're* less inclined to type in those programs. You're more interested in hands-on

> features, dependable and forthright product reviews, and insightful columns.
>
> (1988, p. 4)

What Keizer was suggesting here, essentially, was that the average computer user (or the average *Compute!* reader) was now more likely to purchase the software and games that they wanted, rather than type in programs from a book or magazine. Technology writer John C. Dvorak cited cost and complexity to explain the decline of the hobbyist programmer:

> I hear from old time hobbyists, and they have a lot of reasons for quitting. The primary one was cited in the late 1980s, and I heard it often – the hobby was too expensive because the platform was changing too rapidly…Except for coding in HTML, which is not really programming, working with today's programming languages requires some schooling and a full-time commitment in too many cases."
>
> (Dvorak, 2003)

Dvorak was perhaps overly pessimistic, as will be discussed shortly. Undeniably, however, a specific assemblage of actors – an assemblage that included hobbyist programmers, hobbyist-friendly computers such as the Commodore 64, and the hobbyist programming books and magazines – was being supplanted by new technologies and new practices. And ultimately what was lost was the primacy of hobbyist programming. Machines such as the Commodore 64 and Atari 800 catered to users that both played and programmed in BASIC; contemporary computers serve a variety of purposes, and therefore cater to many different types of users.

The Player/Programmer Today

The player/programmer did not disappear at the end of the hobbyist microcomputer era, but today's personal computer is much different from the machines that were popular in the 1970s and 1980s. Beyond hardware improvements – which are, of course, significant – the rise of sprawling, GUI-based operating systems have rendered obsolete simpler, hobbyist-friendly systems such as Commodore BASIC. The games and other programs that are developed for today's systems must engage with Application Protocol Interfaces (APIs) that control how programs are displayed onscreen and stored in memory. Despite such challenges, however, there are areas where gamers are still enacting practices that engage with code. I will focus on two such areas here: game modding, and Python programming.

Modding encompasses a wide variety of practices, and therefore a general definition can be difficult. Sotamaa (2007; 2010) defines "mods" as "player-made alterations and additions to pre-existing games," and also notes that mods "range from simple rearrangements of game world elements to total conversions that can be relatively independent of the original game" (p. 240). Scacchi divides mods into five categories: "user interface customization; game conversions; machinima and art mods; game computer customization; and, game console hacking," noting that "each enables different kinds of affordances that govern mods, modding practices, and modders" (2010, p. 2). Perhaps the most well-known example of a total conversion is *Counterstrike*, a multi-player first-person shooter set within a contemporary militaristic context, which was modded out of the science fiction-themed *Half-Life* (Scacchi, 2010). Perhaps not

surprisingly, many modders become entangled in legal trouble related to intellectual property and copyright laws, even if they themselves are not earning anything in the way of royalties for their own work (see Postigo, 2008).

Modding doesn't necessarily entail programming, but for mods that go beyond structural changes (such as level design) and alter game logic, coding is typically necessary. For the popular strategy game *Civilization V*, to take one example, the language Lua is used for game logic routines.[6] It seems reasonable, then, to claim that serious game modders are acting as player/programmers, and I believe that this is valid, though perhaps only up to a point. While modders engage with elements of digital games that are often quite sophisticated, their work can never be fully removed from the games themselves, apart from a select few efforts such as *Counterstrike*. This leads to situations where modders help to increase the revenue-earning potential of a given game without receiving any compensation themselves. As Sotamaa explains it:

> Game hobbyists work voluntarily to develop products for other hobbyists, fans and casual players to consume. Although the modifications are often downloaded for free from the Internet…the gamer needs to have a copy of the original game…Furthermore, high-standard mods can significantly increase the shelf-life of a game…Mods can also increase customer loyalty which correspondingly can be seen to boost the selling of expansion packs and sequels.
>
> (2005, p. 2)

It could be argued that no program truly exists independently of a platform, so that, for example, BASIC programmers using the Commodore 64 were helping to popularize the machine, thereby generating more revenue for Commodore without reciprocal compensation. The key difference is that these users were free to distribute and also make money off their work, perhaps by selling it to a hobbyist magazine. That is not to say that all – or perhaps even most – modders would want to profit from their work, if given the chance. We have already seen how modified versions of *Spacewar* and similar games from that era were passed around freely. Conversely, a modder will occasionally get hired on by the company that produced the original game (Kücklich, 2005). Still, concerns that modders are little more than "prosumers" – providing free labour to corporations while they ostensibly play – continue to be raised and debated (Hong, 2013).

Outside of modding, the communities that have clustered around the Python programming language engage in practices that very much resemble the activities of hobbyist programmers in the 1970s and 1980s. Python is an "interpreted, interactive, object-oriented programming language" that aims to be both simple enough for new programmers to learn and versatile enough to support more complex programming techniques ("General Python FAQ", n.d.). The language's creator, Guido von Rossum, strove for clarity and ease-of-use, but also for extensibility. Python's interpreter, which allows users to type in commands and code that are evaluated immediately, echoes the functionality of Commodore BASIC and similar BASIC interpreters from the period. As an online tutorial for the language explains it, "the interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw-away programs, or to test functions during bottom-up program development" ("The Python

tutorial: 1. whetting your appetite", n. d.) Critically, Python also operates under a very generous copyright framework:

> You can do anything you want with the source, as long as you leave the copyrights in and display those copyrights in any documentation about Python that you produce. If you honor the copyright rules, it's OK to use Python for commercial use, to sell copies of Python in source or binary form (modified or unmodified), or to sell products that incorporate Python in some form.
>
> (General Python FAQ, n.d.)

In keeping with Python's hobbyist character, technology writers such as Al Sweigart have taken to writing game programming books that echo the casual user-friendly works published in the 1980s. Sweigart's *Invent Your Own Computer Games With Python, 2nd Edition* deals largely with text-based game creation, shifting in later chapters to games that incorporate the Pygame graphical game programming module. His *Making Games with Python and Pygame* is more advanced, and deals solely with Pygame game programming (Sweigart, 2012a; Sweigart, 2012b). Both books are available for free, highlighting one of the ways in which the communities of actors clustered around Python foster its ethos of user-friendly, open-source, supportive, and positive programming. From a structuration standpoint, Python is enacted by users in ways which support this ethos.

Conclusions

Despite arguing the case for the player/programmer, I do not mean to suggest that more familiar game-playing practices should in any way be diminished. Particularly with the sophistication of modern digital games – both AAA and independent, though perhaps for differing reasons – the experiences of the gamer as player are often extremely rich. Adding programming into the mix, in fact, might actually reduce this richness in many cases. It might be argued, for example, that *Civilization V* is such a high-quality title that tinkering with its rules and assets can only make it worse. Or, perhaps, one might claim that programming a decent digital game using Python and Pygame can take so much time, and the end result could be so unsatisfying, that it does not provide sufficient "payoff" when compared to digital game playing.

Despite these issues, I believe that the player/programmer role must be identified and investigated because its presence can have a largely positive effect on gaming culture. It sheds light on ways in which we might expand on the affordances allowed players so that they might have more control over their gaming experiences, as well as an outlet to realize their creative potential with respect to the games they play. Such affordances are allowed in certain areas, as we have seen. But a wide-scale movement towards building a player/programmer ecosystem – assemblages of practices, programs, and communities that could span a number of different games – could make such tools available to much wider audiences, and could broaden our horizons in terms of thinking about how digital games might grow and evolve.

This paper was an attempt to use two theoretical models to help us understand how the player/programmer emerged, and why this role was more prominent in some eras as compared to

others. This is hardly meant, of course, to be the final word on this subject. To move beyond this largely conceptual study, two directions for future research suggest themselves. Firstly, it could be worthwhile to engage with individuals who currently engage in player/programmer practices. It would be invaluable to learn more about what they think of themselves and the work/play they do, and to get a better understanding of the games they play and the ways in which they customize their experiences of those games. Secondly, it would be interesting to build prototype games, or at least game engines and/or interfaces, in which player/programmer practices were taken into consideration. How might we modify the typical real-time strategy game, for example, so that programming in some form is required to play, rather than just being part of a modding package? Is it possible to create a set of tools that are simple to use, but are able to produce complex game mechanics? These are just some of the questions we might address if, in keeping with the player/programmers we study, we enact a bit of code ourselves.

References

The BBC Microcomputer and me, 30 years down the line. (2011, December 1). *BBC News*. Retrieved from http://www.bbc.co.uk/news/technology-15969065

General Python FAQ (n. d.). Retrieved from http://docs.python.org/2/faq/general.html

The Python tutorial: 1. whetting your appetite. Retrieved from http://docs.python.org/2/tutorial/appetite.html

Anderson, J. J. (1984, November). Dave tells Ahl – the history of Creative computing. *Creative Computing*, *10*(11), 66–74.

Ahl, D. (1975). *101 BASIC computer games*. Maynard, MA: Digital Equipment Corporation.

Bardini, T. (2000). *Bootstrapping: Douglas Engelbart, coevolution and the origins of personal computing*. Stanford, CA: Stanford University Press.

Barnes, S. B. (1997). Douglas Carl Engelbart: Developing the underlying concepts for contemporary computing. *Annals of the History of Computing, IEEE*, *19*(3), 16-26.

Barton, M., & Loguidice, B. (2009, 10 June). The history of Spacewar!: the best waste of time in the history of the universe. *Gamasutra*. Retrieved from http://www.gamasutra.com/view/feature/132438/the_history_of_spacewar_the_best_.php

Brand, S. (1972, 7 December). Spacewar: Fanatic life and symbolic death among the computer bums. *Rolling Stone*, 50-57. Reprinted at http://www.wheels.org/spacewar/stone/rolling_stone.html

Brookhaven National Laboratory. (1981/2011). Video games – did they begin at Brookhaven? *U. S. Department of Energy, Office of Scientific & Technical Information*. Retrieved from http://www.osti.gov/accomplishments/videogame.html

Callon, M. (1986). The sociology of an actor-network: The case of the electric vehicle. In M. Callon, J. Law, & A. Rip (Eds), *Mapping the Dynamics of Science and Technology* (pp. 19-34). London: Macmillan Press.

Commodore Business Machines. (1980). *Commodore BASIC, version 4.0*. Commodore Business Machines. Retrieved from http://www.commodore.ca/commodore-manuals/

Crawford, C. (1984). *The art of computer game design*. Berkeley, CA: McGraw-Hill/Osborne Media.

Dvorak, J. C. (2003, 20 October). Will the last computer hobbyist please turn out the lights? *PC Magazine*. Retrieved from http://www.pcmag.com/article2/0,2817,1358125,00.asp

Edwards, P. N. (1997). *The closed world: Computers and the politics of discourse in Cold War America*. Cambridge, MA: The MIT Press.

Engelbart, D. C. (1962). *Augmenting human intellect: A conceptual framework*. Menlo Park, CA: Stanford Research Institute.

Ford, K. M, and Hayes, P. J. (1998). On computational wings: Rethinking the goals of artificial intelligence. *Scientific American*, Special Issue on "Exploring Intelligence", *9*(4), 78-83.

Giddens, A. (1984). *The Constitution of society: Outline of the theory of structuration*. Berkeley and Los Angeles: University of California Press.

Graetz, J. M. (1981, August). The origin of Spacewar. *Creative Computing*. Reprinted at http://www.wheels.org/spacewar/creative/SpacewarOrigin.html

Hong, R. (2013). Game modding, prosumerism and neoliberal labor practices. *International Journal of Communication*, *7*, 984-1002.

Huizinga, J. (1955). *Homo ludens; A study of the play-element in culture*. Boston: Beacon Press.

Hurst, J., Mahoney, M. S., Taylor, N. H., Ross, D. T., and Fano, R. M. (1989). Retrospectives I: The early years in computer graphics at MIT, Lincoln Lab, and Harvard. Proceedings from *SIGGRAPH (Association for Computing Machinery, Special Interest Group on GRAPHics and Interactive Techniques) '89*. Boston. MA.

Johnstone, B. (2003). *Never mind the laptops: Kids, computers, and the transformation of learning*. New York: iUniverse.

Kafai, Y. (1995). *Minds in play: Computer game design as a context for children's learning*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Keizer, G. (1988). Editorial license. *Compute!* 96, 4.

Kücklich, J. (2005). Precarious playbour: Modders and the digital games industry. *fibreculture*, *5*. Retrieved from http://five.fibreculturejournal.org/fcj-025-precarious-playbour-modders-and-the-digital-games-industry/

Latour, B. (1983). Give me a laboratory and I will move the world. In K. Knorr-Cetina, and M. Mulkay (Eds.) *Science Observed: Perspectives on the Social Study of Science* (pp. 141-169). London: Sage.

Latour, B. (2005). *Reassembling the social: An introduction to actor-network theory*. Oxford: Clarendon.

Leslie, S. W. (1993). *The Cold War and American science: The military-industrial-academic complex at MIT and Stanford*. New York: Columbia University Press.

Levy, S. (1984/2010). *Hackers: Heroes of the computer revolution, 25th anniversary edition*. Sebastopol, CA: O'Reilly Media.

Law, J. (1992). Notes on the theory of the actor-network: Ordering, strategy, and heterogeneity. *Systems practice*, *5*(4), 379-393.

Licklider, J. C. R. (1960). Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, *HFE-1*, 4-11.

McCarthy, J. (2001). What is artificial intelligence? *Laboratory for Research and Development in Scientific Computing*. Retrieved from http://lidecc.cs.uns.edu.ar/~grs/InteligenciaArtificial/whatisai.pdf

Montfort, N. (2005). *Twisty little passages: An approach to interactive fiction*. Cambridge, MA: The MIT Press.

Orlikowski, W. J. (2000). Using technology and constituting structures: A practice lens for studying technology in organizations. *Organization science*, *11*(4), 404-428.

Postigo, H. (2008). Video game appropriation through modifications: Attitudes concerning intellectual property among modders and fans. *Convergence: The International Journal of Research into New Media Technologies*, *14*(1), 59-74.

Redmond, K. C., & Smith, T. M. (1990). *Project Whirlwind: The history of a pioneer computer*. Bedford, MA: Digital Press.

Redmond, K. C., & Smith, T. M. (2000). *From Whirlwind to MITRE: The R&D story of the SAGE air defense computer*. Cambridge, MA: The MIT Press.

Scacchi, W. (2010). Computer game mods, modders, modding, and the mod scene. *First Monday*, *15*(5). Retrieved from http://firstmonday.org/article/view/2965/2526

Servomechanisms Laboratory, Massachusetts Institute of Technology. (1949). *Summary Report No. 20: Third Quarter, 1949*. Cambridge, MA. Retrieved from http://dome.mit.edu/handle/1721.3/40719

Sotamaa, O. (2005). "Have fun working with our product!": Critical perspectives on computer game mod competitions. *Proceedings of DiGRA 2005 Conference: Changing Views – Worlds in Play*. Vancouver: University of Vancouver. Retrieved from http://www.digra.org/digital-library/publications/have-fun-working-with-our-product-critical-perspectives-on-computer-game-mod-competitions/

Sotamaa, O. (2007). On modder labour, commodification of play, and mod competitions. *First Monday, 12*(9). Retrieved from http://firstmonday.org/article/view/2006/1881

Sotamaa, O. (2010). When the game is not enough: Motivations and practices among computer game modding culture. *Games and Culture*, *5* (3), 239-255.

Suits, B. (2005). *The grasshopper: Games, life and utopia*. Peterborough, ON: Broadview Press.

Sweigart, A. (2012a). *Invent your own computer games with Python*. 2nd edition. Retrieved from http://inventwithpython.com/

Sweigart, A. (2012b). *Making games with Python and Pygame*. Retrieved from http://inventwithpython.com/pygame/

Winter, D. (n.d.). Noughts and crosses – the oldest graphical computer game. *Pong-Story*. Retrieved from http://www.pong-story.com/1952.htm

---

[1] As Taylor indicates, Bouncing Ball might be yet another contender for the honour of being the first computer game, though I have not seen his claim repeated or reaffirmed outside of this presentation.
[2] http://www.apple.com/ca/imac/
[3] http://www.microsoft.com/canada/media/default.aspx
[4] There are notable exceptions to this perspective, which I will address.
[5] To cite one example, a website devoted solely to the *ZX Spectrum* (released in 1982) lists dozens of books that were available at the time: http://www.worldofspectrum.org/books.html. Interestingly, no formal, academic study has been conducted on this issue.
[6] http://civilization.wikia.com/wiki/Modding_(Civ5)