Surveillance & Society

Opinion

Enclosures at Play: Surveillance in the Code and Culture of Videogames

Alex Dean Cybulski

University of Toronto, Canada.
alexander.cybulski@mail.utoronto.ca

Introduction

"Triple A" titles are the videogame industry's equivalent of cinema's summer blockbusters. These games are replete with disaster scenes, multi-million dollar budgets and celebrity voices. Their release is often heralded by expensive marketing campaigns rivaling their film counterparts. Given the game industry's focus on production values and spectacle, it might be surprising then that many players are often pulled from these immersive simulations by text notifications, a reminder the games they are playing are being systematically surveilled by software running on their videogame platform of choice. These notifications represent the accumulation of virtual trophies, what Microsoft calls "achievements," a system of in-game surveillance pioneered on the Xbox 360 videogame console in 2005. Achievements act as recognition of a player's videogaming prowess and these trophies are facilitated by complex surveillant algorithms and code built into the architecture of contemporary videogames. At first, the presence of overtly surveillant mechanisms in the playful game context might seem odd and unsettling. To explain the presence of surveillant mechanisms within the playful context of games, this article identifies surveillant aspects of videogame code and culture that facilitate data collection. This paper argues that modular programming architecture, data structures for dynamic simulation and the visibility of arcades have fostered the surveillant potential of videogames, and that these traits have been leveraged by the contemporary videogame industry to create surveillant games designed to monitor, measure and mine play for data from users.

If we want to understand how videogames facilitate surveillance it is vital to understand that on a technical level they are programs designed to operate like a simulation. More specifically, videogames are a simulation where an activity (real or imagined) is represented through the execution of code. The architecture of code is therefore important to the action of videogames, because as code is executed it is responsible for the flow of information to and from the user who is engaged with the simulation, and also to and from the computer which is executing the code. For these transactions to occur, code must render information passing from the user or the computer into a legible form, which it uses to execute a series of instructions. What is important about this transformative process is that at the level of code, both the user and the computer are, relatively speaking, talking in the same language. However, this language and its meaning is only known to the code of the game, as the user is instead provided with a representation of their input generated through the code. This means that code is both opaque and asymmetrical: it exchanges information for the representational action that occurs within a game, meaning that the actual application of a user's input and in turn, the effect that this input has on code and computer's execution of code is obfuscated. By analyzing the design of code in contemporary videogames, we can better understand how this makes code into a surveillant process of informational collection.

Cybulski, A.D. 2014. Enclosures at Play: Surveillance in the Code and Culture of Videogames. *Surveillance & Society* 12(3): 427-432. http://www.surveillance-and-society.org | ISSN: 1477-7487

To understand the role of videogame code as an intermediary and its potential surveillant applications it is crucial to identity two common aspects of its design. First, complex computer programs use what are known as data structures, shared variables and internal memory. These are parts of a program where information is organized and stored to be shared between different parts of a program's code. To use a relatively simple example: the storage of information in internal memory is often encountered in a game's score, when a player's activities within a game are systematically monitored, tallied and displayed to represent the user's skill and success within a game. Each time the user performs an action designated by the game the score is retrieved from the internal memory of a program, updated to reflect this action, and then stored again within this structure. Various other functions can manipulate and change information in a program's internal memory, as it is specifically designed to efficiently store and share information throughout a program. Examples of this might include data stores that can be used in a static way to record things like a player's name or to create dynamic and complex algorithms which are shaped by the player's activity, like an artificially intelligent soldier whose behavior is derived from data and used to anticipate the player's tactics.

Secondly, videogames and sophisticated simulations frequently use discrete, modular code to simplify their design, breaking a program up into segments that perform various functions. These functions might include the computation of complex algorithms or the collection of input data from a joystick or videogame controller. These modular "chunks" of code make it easier for programmers to fix and modify a complex program by editing specific operations as opposed to a large monolithic and incredibly complex program (Nutaro 2010: 2-3). The modular design of programs also allows a programmer to reflexively use the function of these modules, recalling them from code when needed to perform a specific task or to act in concert with other code segments. This storage and computation structure can produce complex results that range from a ghost inexorably hunting a player through a maze in the 1982 arcade game *Ms. Pac-Man*, to the shape and size of waves swaying the player's pirate ship back-and-forth as it sails the rough seas of the Caribbean in Ubisoft's 2013 game *Assassin's Creed IV: Black Flag*.

While this explanation of how code is structured omits much of the complexity behind programming videogames, it usefully identifies significant components of these programs which are oriented toward surveillance. In particular, the explanation identifies two vital traits of videogame code. First, it reveals these programs are structured around modules of code. Second, it shows there exist sites within this code where information is stored and shared for later use. There exist many programming techniques including "observer pattern" and modular "aspect oriented" programming which are intended to monitor interactions between the program and user, often altering the behavior of code to suit observations made by the program. The best model for describing this modular and discrete form of tracking found in the architecture of videogame code is what William Bogard has described as a surveillant "enclosure." The design of videogame code bears distinct similarities with enclosures because the panoply of functional modules and the data structures act in concert, efficiently creating, capturing and sharing information. Among the traits of an enclosure is to ensure the "mobility" of information while simultaneously capturing its flow (Bogard 1996: 5). This model is useful in understanding videogame code because data is not arrested by a program, but rather collected and exchanged between the user, the instructions found in code, and the instructions required to make a computer perform certain operations dynamically, mediating action and reaction seamlessly.

The mobility of these collected data is key to the interaction between the player, the game and the computer. For example, the data created by the code when it receives input from a player through a joystick is shared among a host of processes. It could be translated into the movement of an avatar on the screen or used by the game's adversaries to hunt the player through a maze. It is here that the legibility of this information in code is integral to the mediation between user and computer. As Alexander Galloway has argued, the effect of videogame code "transubstantiate[s]" physical input into digital information. This in turn causes physical activity within the computer: "at runtime code moves. Code effects physical

change in a very literal sense. Logic gates open and close. Electrons flow. Display devices illuminate" (Galloway 2006: 5). What is important about Galloway's argument as it pertains to surveillance is that videogame code has always relied on an intense process of data collection to create the "action" of play (Galloway 2006: 2). What is interesting about these processes of collection is that the surveillant aspects of code are ambivalent; they exist to ensure that videogames act as dynamic, seamless simulations where the role of computation is opaque and instantaneous.

However, code is not the only part of videogames that are surveillant. The early history of videogame culture is illustrative of this quality. The arcades, recreation centers and bowling alleys of the late 1970s and early 1980s served as the public gateway to videogames. Accordingly, one of the affordances of videogaming in a public space was a performative culture related to their play: a good player might attract a crowd of spectators, or a challenger would "quarter up," placing a coin on the corner of an arcade cabinet to signify their desire to play next on the machine. Players who were not immediately recognizable would at least be known by their initials placed beside high scores that were repeatedly displayed at the end of a game or while the game was in an inert state. At their apex, arcade games became a minor sensation in the United States with sponsored tournaments for the best players, proto-gamers vying for the top score on games like *Asteroids* and *Centipede*. Yet the visibility of arcades was short lived. They were commonly seen as undesirable spaces for youth and the popularization of home videogame consoles caused their rapid decline. With this shift into the home, videogames became a private endeavor for much of the 1980s and 1990s. During this time, games retained many of their arcade-like qualities: a player's score was often tallied and scoreboards were still used even if the systems themselves could not permanently keep this information in their memory in the same way arcade cabinets could. In this way, the logic of visibility common to the arcade still permeated the games designed for home videogame systems, even if only in a fragmentary way. Subsequently, the use of data structures for tabulating and reporting a user's performance persisted in configuring the experience of playing games even if there was no one outside a user's home watching them play.

When videogame systems began to connect to the internet, the public scoreboard of the arcade and the visibility that artifact afforded returned to videogame culture. In 1999 Sega's Dreamcast pioneered internet connectivity and allowed users to vie for the top score not merely among a small, localized community of players but with any user on the planet who could connect their system to the internet. Microsoft helped Sega network its home videogame system by providing the operating system for the console (Street 2013: 12). When Sega's fortunes in videogame console production declined, Microsoft entered the market with the Xbox, one of the first fully networked videogame systems that could use a broadband internet connection for gaming over the internet. For the second iteration of the Xbox, the Xbox 360, Microsoft heavily integrated the arcade logic of performativity in videogames and the visibility of the arcades by implementing digital trophies known as "achievements," which are awarded automatically as a player accomplishes objectives set by game designers.

Achievements are a fascinating subject of study with respect to the political economy of videogame production, as these digital trophies are indicative of the power disparity between the producers of videogame systems like Microsoft and the users who play games on their platform. These digital trophies and the tracking they represent are indicative of Microsoft's ability to forcibly surveil users, who have no way to "opt-out" of achievements, dictating that the surveillance of play is a normative operation on the console. Analyzing how this tracking is performed on the Xbox 360 and how it is represented to the user is indicative of the way in which Microsoft has leveraged pre-existing conditions in videogame culture and technology to facilitate surveillance. Achievements operate by leveraging the sophisticated software architecture of the Xbox 360 to communicate between different layers of software running on the platform at any given time. The first layer, the games themselves (and the code they run) communicates with a second layer known as a "runtime environment," a kind of operating system called the XNA Framework which works in the background to share information between other layers of software. When certain pre-

defined objectives are met by a user playing a game, the game's code transmits a piece of identifying information from the data structure known as a "string" to the XNA Framework. The XNA framework analyzes this string for pertinent information including the time and whether the device was connected to the internet. Afterwards, the Xbox 360 makes a pleasant sounding "plink" noise and a small dialogue box appears on the screen with the name of the achievement and the phrase "achievement unlocked." The effect is not unlike receiving a text message on a phone, but the connotations of its effect on play has significant political and technical implications for how the videogame and the platforms on which they are played operate.

After the achievement has been scored by the Xbox 360, a predefined number of points known as "gamerscore" are allocated to the user's account. Gamerscore acts as the new scoreboard of arcade cabinets, as these points are tallied on user's public profiles thus closely associating the process of surveillance performed by achievements with a player's sense of identity and skill. Jennifer Whitson has described features like gamerscore as "juicy feedback," sounds and other indicators like points that act as structural tools that create positive reinforcement and encourage user engagement (Whitson 2013: 166). Achievements are a robust system of tracking that can ascertain the state of certain singular objectives like "rescue the princess" but are also capable of recording more complex non-linear player activities like "shoot 100 zombies in the head" or "fall 5000 feet," statistics that are tallied over dozens, often hundreds of play sessions with a game. The surveillance of these complex non-linear player activities is the result of instructions found within modules of code which systematically record the frequency with which they are executed within the game's code and the outcome of their application. This information is then transmitted and stored to a value database, an enclosure of memory exigent to the actual playing of games where data about game play is systematically and routinely recorded. This is represented to the user as an achievement, but is indicative of a pervasive form of data tracking that operates from within the architecture of videogame code. What is interesting about Microsoft's branding of a brazenly surveillant process is that it distinctly hinges on the design of computer programs and their relationship to earlier public performances of games. This approach to tracking play within a videogame appropriates processes related to surveillance in games which where once ambivalent and associates them closely with a project of governance, assigning identities to gamers based on this watchful gaze.

Microsoft's competitors have been quick to follow the corporation's strategy on the Xbox 360. Sony, Nintendo and Apple have since implemented achievement-like systems on their videogame platforms as well. Popular videogames *Battlefield 4* and *Starcraft 2* even allow users to analyze the data derived from their play for further analysis and many fan-made websites give communities free access to this information as a tool to inform strategic players who apply this information in competitive play. On mobile phones, software platforms like Fuseboxx permit game developers the opportunity to analyze how users interact with their games in the aggregate. These widespread surveillant practices may seem unsettling to those who do not play videogames, but games scholar Mikael Jakobsson has observed through a series of ethnographies that videogame players find the concept of achievements and similar representational presentations of in-game surveillance highly rewarding and engaging (Jakobsson 2011). In particular, Jakobsson's analysis demonstrates that the presence of surveillance in videogames has been well-received in gaming culture as this watchful gaze is perceived to enhance and modulate the pleasurable experience of play.

However, this pervasive surveillance in videogames has not gone without criticism. Critics like Emanuel Maiberg have argued these systems have become instrumentalized in the transactional style game design. In a blistering critique entitled *What Big Data Can't Teach us About Videogames*, Maiberg attacks games like those using the Fuseboxx system that collect data and use it to manipulate users into small in-game commercial transactions or continue playing games that increasingly provide loops of meaningless feedback and use positive feedback and its denial to psychologically manipulate or entice users into paying real money for small packets of enjoyment (Maiberg 2013). Additionally Maiberg and Whitson

have separately argued that aggregate collection of data in games is increasingly having deleterious influence on game design, with videogame studios opting towards data-driven game design processes and eschewing the expert knowledge of designers and the preferences of large audiences of players who represent the core audience of a specific game (Sinclair 2013). The result, as Maiberg argues, are vacuous games "reduced to a science," suggesting that games designed by data driven processes are transactions where enjoyment is the product of a financial exchange between the player and the game that metes out pleasure for money. In this respect it may be more appropriate to suggest that Maiberg's criticism of surveillance found in games reduces them to a crude business, a Pavlovian mechanism designed and honed by aggregate surveillance to extract a proportional value from consumers in exchange for pleasure by determining when they are most engaged with a game, limiting the artistic expression of game designers, the enjoyment of players and engaging in questionable business practices related to the manipulation of customers.

While Maiberg's criticisms address issues with the artistry and pleasure of playing games, there exists a deeper issue of political economy related to surveillance in games, namely the power disparity between players and the corporations who make the high-tech videogame platforms they use. More often than not these tracking systems, like achievements on the Xbox 360, cannot be turned off or evaded by opting-out. Additionally, there is little indication to players what kind of information they are sharing with the videogame or console's producers, meaning that players may not be aware of what kind of information they are giving up. A cursory examination of Microsoft's terms of service for Xbox Live, the online network for the Xbox 360 is troubling. Among other demands, Microsoft notes in this policy document that it reserves the right to "track, store, copy, distribute, broadcast, transmit, publicly display and perform, and reproduce: (i) your game scores; (ii) your game play sessions" (Microsoft 2013). What is incredible is that if the Xbox 360 possesses invasive surveillance processes capable of directly recording a user's game play they are not documented in much technical literature provided to game developers. Thus, it is likely there are extensive undocumented powers of surveillance on the Xbox 360 only hinted at in policy documents like the Xbox Live terms of service. Policy documents are therefore interesting with respect to videogame systems like the Xbox 360 because they are protective of the privileges which Microsoft has provided itself within the hardware of their videogame system. Simultaneously, these documents are also indicative of a void of protection for those using these high-tech devices, as compared to previously non-networkable videogame platforms which posed less of a privacy risk and whose policies were significantly less imperial in their reach into the consumer's home.

While this paper originally focused on the latent aspects of code and videogame culture which could be construed as conducive to surveillance, the key shift here was away from ambivalent processes of surveillance towards those branded and governed by corporate interests, in this case mainly Microsoft's. The consequences of these monitoring systems within the videogames industry and culture are widespread, influencing the design of games, the pleasure to from playing them and even user's personal privacy. Perhaps the most telling sign that these systems within videogames are surveillant is how inescapable they have become, indicating a significant power disparity between players and the peddlers of their entertainment.

References

Bogard, William. 1996. *The Simulation of Surveillance: Hypercontrol in Telematic Societies*. Cambridge: University of Cambridge Press.

Galloway, Alexander. 2006. *Gaming: Essays on Algorithmic Culture*. Minneapolis, MN: University of Minnesota Press.

Jakobsson, Mikael. 2011. The Achievement Machine: Understanding Xbox 360 Achievements in Gaming Practices. *Game Studies: The International Journal of Computer Game Research* 11(1). http://gamestudies.org/1101/articles/jakobsson

Maiberg, Emanuel. 2013. What big data can't teach us about videogames. *Killscreen Magazine.* http://killscreendaily.com/articles/big-dada/

Microsoft. 2013. Terms of Use. October 2013. Xbox.com. http://www.xbox.com/en-CA/Legal/LiveTOU

Nutaro, James. 2010. *The Building Software for Simulation: Theory and Algorithms, with Applications in C++*. Hoboken, NJ: Wiley.

Sinclair, Brendan. 2013. Dev's can't "create magic" with a spreadsheet in front of them. *Gamesindustry international.* http://www.gamesindustry.biz/articles/2013-11-11-devs-cant-create-magic-with-a-spreadsheet-in-front-of-them
Street, Zoya. 2013. *Dreamcast Worlds: A Design History*. Oakland, CA: Rupazero.
Whitson, Jennifer. 2013. Gaming the Quantified Self. *Surveillance & Society* 11(1/2): 163-176.