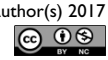
*Article*

new media & society
2018, Vol. 20(7) 2315–2332



DOI: 10.1177/1461444817715020
journals.sagepub.com/home/nms


# Voodoo software and boundary objects in game development: How developers collaborate and conflict with game engines and art tools

**Jennifer R Whitson**
University of Waterloo, Canada

**Abstract**
This article describes how game developers successfully 'pull off' game development, collaborating in the absence of consensus and working with recalcitrant and wilful technologies, shedding light on the games we play and those that make them, but also how we can be forced to work together by the platforms we choose to use. The concept of 'boundary objects' is exported from Science and Technology Studies (STS) to highlight the vital coordinating role of game development software. Rather than a mutely obedient tool, game software such as Unity 3D is depicted by developers as exhibiting magical, even agential, properties. It becomes 'voodoo software'. This software acts as a boundary object, aligning game developers at points of technical breakdown. Voodoo software is tidied away in later accounts of game development, emphasizing how ethnographies of software development provide an anchor from which to investigate cultural production and co-creative practice.

**Keywords**
Boundary objects, collaboration, ethnography, game developers, game development, platform studies, software studies, studio studies, voodoo software

**Corresponding author:**
Jennifer R Whitson, Department of Sociology and Legal Studies, University of Waterloo, PAS Building, 200 University Avenue West, Waterloo, ON N2L 3G1, Canada.
Email: jwhitson@uwaterloo.ca

> We are standing in a large and well-known game studio. White cubicles stretch in three directions off to a horizon terminating in skyscraper views of the city. None of the developers have remarked on the view. They are feverishly working to develop and launch a game in ten weeks and have hit the midway point.
>
> Clark is attempting to bring Alex's shark to life, importing it from 3D Studio Max to Unity. Something has gone terribly wrong. The shark floating in the middle of the black gridded screen is lifeless, not moving in response to its embedded animations. Moreover, it is entirely deflated and oversized, as if a steamroller has pressed it entirely flat except for its fins.
>
> Clark moves to re-inflate the shark and ensure it is the right size. It starts to weakly thrash its tail. But it flickers in and out of existence. More importantly, it is still deformed, now looking like someone had attempted to fold it up like an accordion from teeth to tail. Clark guesses that this mutation is due to Alex improperly exporting the file to Unity. But he's not sure. It may be a 'Unity Problem' or something he himself has done incorrectly. Half the ten-person team starts to gather around his screen, muttering suggestions.

Unsurprisingly, software tools exert considerable influence on the work of developers and the final shape of their products (Bogost and Montfort, 2009). Drawing from Star and Griesemer (1989) and the field of Science and Technology Studies, this article shows how the concept of 'boundary objects' can be exported from the study of science to better understand the vital coordinating role of game development software tools in the creation of culture, both in terms of professional workplace culture, but also games culture at large. Extending Latour's (1986) concept of an immutable mobile, boundary objects are plastic and are interpreted differently across communities but contain enough immutable content to maintain integrity. Accordingly, boundary objects such as software tools allow for an actor's local understanding (i.e. one intern on a development team) to be reframed in the context of some wider collective activity (i.e. collaborative game-making) despite the lack of a shared language, skill set or even consensus with other teammates on what the final project should look like.

As this article will demonstrate, in the field, software tools such as Unity 3D and 3D Studio Max clearly operate as boundary objects. However, at various times, they are also anthropomorphized by developers and seen as exerting human-like agency and even magic. This is especially apparent during times of social conflict. Software operates as a team member that must be coaxed into alignment rather than a static tool, becoming 'voodoo software', an actor's category that refers to software seemingly exhibiting a mind of its own, operating in a manner counter to users' input and goals, and thus shaping the interactions of the team, the shape of the game and the collaborative processes of developers. Voodoo must be dealt with before developers can continue with their tasks; however, it plays a largely unrecognized role in shaping game development practices.

## Context and background

This article details an ethnographic study of 10 intern developers at a large international game studio in Canada, providing a picture of how new game developers from widely

divergent backgrounds come together, learning to collaborate not only with each other but also with sophisticated game development software tools. More broadly, this research focuses on the sociomateriality of digital games, showing how they are the result of constitutive entanglements of the social and the material, a creative intertwining of humans and technological practice in 'designing play'. A number of other game studies scholars highlight the sociomateriality of play, arguing that games do not exist in and of themselves – they are not just rules embedded in hardware – but are also constituted by and operate in conjunction with players and the spaces they operate within (Chen, 2012; Steinkuehler, 2006; Taylor, 2006). In parallel, I largely focus on the production of games, arguing that the shape and form of games is not only dependent on human designers but also the interventions of material artefacts such as game software. Not inconsequentially, the resulting game is a mangling of human and material agencies (Pickering, 1995). This necessarily changes how we conceive of authorship, and how we ascribe authorial intent when analysing games and player's interactions with them.

Knowing how to make games, and acting within a game development team is always and irremediably a contingent process, as the team grapples with the trial and error of learning how to manipulate technical systems to produce 'fun', but also the ambiguity of interpretive processes, of experience, of history and conflict (see also, O'Donnell, 2014). Team members improvise new practices as they invent, slip into or learn new ways of interpreting and experiencing the world of game development. Their situated practice often involves reflection and experimentation (whether this occurs via recounting one's 'tasks' during the daily scrum meeting or creating a 'grey box' digital or paper prototype to test out the feasibility of implementing a new mechanic). It is these in-the-moment reconstructions of thought and action where knowing how to be a game developer is learned and games are created.

Ethnographic methods are an ideal way to learn about game development processes, and to specifically trace the impact boundary objects have on both team collaboration and the final shape of a game. The vignettes that comprise this article were collected during the summer of 2012 while working on the GAMEDEV fieldwork project, hosted by the game studio, PlayHouse, and located in a large Canadian city.[1] Introduced to a game prototype created by two PlayHouse employees, 10 interns were instructed to remake the multiplayer cooperative game. At the end of the 10 weeks, the final game was presented to PlayHouse employees and posted online as a free download. Other than initial 'how-to' sessions hosted by PlayHouse employees, the team was functionally separated from the daily processes and flows of the studio. All design decisions, from who would occupy what position to how far they would deviate from the original game were left completely up to the team – giving them complete creative control.

Despite this autonomy (or, perhaps, as a result of it), the team was quick to adopt processes and technologies which they felt mirrored the 'real' development happening elsewhere in the office. They hosted their design documents on the company's internal wiki pages, and set schedules, tracked individual performance and assigned tasks using Jira tracking software. They adapted a method of agile development with daily scrums (stand-up progress meetings) and broke up their production into week-long sprints, with a new playable version of the game shown to PlayHouse's Creative Director every Friday. Fieldwork consisted of ethnographic observation, formal and informal

interviews, video and photographic documentation, and analysis of internal communication channels, including access to the team's Facebook, Skype and project wiki pages.

Software development, particularly game development, is inherently cross-disciplinary. Kerr (2006) and O'Donnell (2014) separate engineers (programmers) from designers, from artists and from management. These four components are interdependent, working together to provide a cohesive game experience.[2] Game developers as such do not reflect one homogeneous community, nor do they share the same body of background knowledge, but rather reflect smaller communities of practice that are separated according to discipline and must work together to develop a cohesive game.

Each of these groups speaks a highly specialized disciplinary language, and thus communication between these different groups is difficult without a common touchstone: the games they all play. References to games, game mechanics, play styles and genres become primary means of understanding and communicating about the underlying systems and structures of the games they are working to create (O'Donnell, 2014). The role of software tools themselves plays a similar role in terms of managing friction between different disciplinary understandings. O'Donnell (2014) describes how internal tools built by engineers – such as shortcuts that allow artists to more quickly integrate their assets into the game – become a commodity that acts with intention, purpose and priorities in terms of enabling and constraining certain activities over others. While noting the relevance of boundary objects, O'Donnell uses the metaphor of a 'creole' language to describe the consensus-building utility of these tools, which function as communicative devices in the heated 'trading zones' between disciplines. This shared focus on common tools and processes cultivated interdisciplinarity in the GAMEDEV team, with members learning along the way the rudimentary skills needed for the production of both technical and art assets. For example, the game designers learned to create art assets using art software such as 3D Studio Max and Photoshop, and integrated these assets and balanced gameplay using the game engine, Unity 3D.[3] These software tools become important boundary objects in the project.

In the case of the GAMEDEV project, the seven male and three female participants came from very different communities of practice. The composition of the team was notably diverse. None of the team members had worked together before, and their experience on game-making projects varied considerably. Three of the team had programming backgrounds, and three had art and animation backgrounds, while the remaining four members had backgrounds spanning history, dance, cinema studies and linguistics. Recruiting geographically diverse participants from New England to Northern Alberta meant that participants exhibited significant social differences, with a primary division between English and French-speaking participants in terms of both culture and politics, as well as by discipline in terms of art, design, programming and management. In short, it was apparent early on in planning meetings that each participant had a different vision of what a 'good' game was, and what they'd like to design, and that this team of 10 individuals was far more heterogeneous than other development teams (De Vaan et al., 2015).

Given their heterogeneity, their lack of experience in game development and their unfamiliarity with each other and professional workplace practices, one could assume that the project was doomed from the start. However, in 10 weeks, the team completely

designed, coded, implemented, polished, tested and launched a narratively driven cooperative multiplayer game from scratch, defying all expectations. The worlds of art, programming, design and management worked coherently together. *How did this happen?*

## Boundary objects and collaborating in the absence of consensus

Star and Griesemer (1989) introduced the concept of boundary objects in their now-classic historical study of Berkley's Museum of Vertebrate Zoology. Star and Griesemer examined how work can be effectively accomplished when heterogeneous participants – from professional scientists, to amateur naturalists, trappers, patrons, hired hands, taxidermists and administrators – inhabit different social worlds, and have conflicting viewpoints, training, skills and goals. Rather than take for granted that entrepreneurs are needed to align the vision and interests of all actors and/or impose top-down governance to cooperate and produce coherent work, Star and Griesemer instead asked the following question: *How do people collaborate in the absence of consensus?* They argue that, contrary to common assumption, consensus is neither necessary for cooperation nor for the successful conduct of work. What is necessary is the standardization of methods and the development of boundary objects. These boundary objects are able to inhabit several intersecting communities of practice at once, while satisfying the information requirements of each of them, simply because they are plastic enough to adapt to local needs yet robust enough to maintain a common identity. Through boundary objects such as a Museum's bird specimens, actors with vastly different goals, such as trappers, ecologists and preservationists, can productively work together despite their separation in time, place, world-view and training.

Boundary objects are important sites of study for a number of reasons. They help us analyse the nature of cooperative work in the absence of consensus, such as how GAMEDEV members managed to create a finished game, despite fundamental differences in how each wanted the end product to look like and function. Boundary objects are useful at an organizational level, shedding light on how people deal with ambiguity, abstraction, differing skill levels and even social worlds to work on a project. As discussed by Bechky (2003), boundary objects – and in particular material artefacts such as blueprints and prototypes – establish and maintain knowledge/power hierarchies in workplace organization. More specifically, focusing on game software as a boundary object helps emphasize the materiality and infrastructural properties of game development, making the work of creating play more visible and amenable to analysis.

A shared focus on learning the tools of game development (3D Studio Max, Unity 3D) and the supporting documentation (from online tutorials, to official forums, to face-to-face human support systems) all worked to tie the disparate team together. Returning to Star and Griesemer, trade between intersecting communities of practice was facilitated by these tools. Important sites of trade were the game engine, Unity 3D and the art tool, 3D studio MAX, both of which were used by all team members on a daily basis.

These tools helped the team define a 'lowest common denominator' which satisfied the minimal demands of each community by capturing properties that fell within the minimum acceptable range of all concerned communities (i.e. defining minimal fidelity

and functionality standards for assets); they allowed members to create versatile, plastic, reconfigurable, programmable objects that other members could then mould to their purposes locally; digital databanks for the team fostered the storing of complex of objects, such as individual game assets and levels, from which things necessary for each community could be physically extracted and configured for local purposes. Each participating community could then abstract or simplify the object to suit its demands; that is, 'extraneous' properties could be deleted or ignored, and thus work in each community proceeded in parallel except for limited exchanges of standardized sorts and was staged so that work was relatively autonomous (Star and Griesemer, 1989: 404).

Accordingly, this software maximized communication between members, while simultaneously allowing for the autonomy of different communities of practice (in this case, the four 'worlds' of art, design, code and project management). Only those parts of the work essential to maintaining coherent information exchange were pooled (e.g. art assets, tools to simplify the Unity interface, diagrams of the artificial intelligence (AI) directing enemy behaviour patterns). The rest was left alone, remaining the private domain of individual specialists. What one could do with the game was necessarily limited by the design of the software itself and the constraints this imposed. Differences in approaches and beliefs, and higher order questions such as the role of play and games in society, or the role of games as methods of critique and reflection were backgrounded by a focus on how to make a game with these tools and constraints.

Star and Griesemer introduce a rough taxonomy for boundary objects that can help us differentiate between their forms and uses. While not meant to be an exhaustive listing, each of the following four types is present in game development:

1. Repositories, such as libraries (both physical and code), order heterogeneous objects in a standardized way, allowing each member to borrow what they like. For example, game assets were stored and organized on a central server, allowing each member to copy and alter whichever files they currently needed. The team also used TortoiseSVN, a version control system that allowed team members to each access the most recent copy of the game as well as past versions with logged changes. This allowed members to work on different versions of the game simultaneously and then merge their separate branches back together, generating a list of what edits were made to each version of the game, at what time and by whom, allowing developers to pinpoint edits that cause the game build to malfunction. When the current version of the game became inoperable, team members went to the log to see who had added assets or lines of code that 'broke the build' and to 'roll back' to earlier working versions of the game.
2. Ideal types, such as diagrams and maps, genres and classes, constituted another category of boundary objects. Ideal types are abstract and not accurate, but are useful in terms of pointing to things and thus becoming a shared basis of communication, thus serving as a general road-map for all parties. For example, in the GAMEDEV Project, design documents on the team wikipage acted as ideal types, with rough descriptions of character occupations (which acted as a shorthand for their physical appearance, powers and motivations), as well as more specific directions as to how non-player characters should behave and how levels

should be mapped out. Less tangible, but very influential ideal types, were the constant references to other successful games. For example, team members went back and played the original game prototype throughout the project, looking for clues as to enemy behaviour and movement patterns (AI), to direction on how to balance length and general mechanics, to tips on how to incorporate humour.

3. Coincident boundaries, such as temporal or spatial boundaries, have the same boundaries but different internal contents, allowing parties to share a common referent. In Star and Griesemer's example of museum-work, the state of California served as a coincident boundary. The maps each member drew of the state had the same outline but differing contents. In the case of the GAMEDEV project, temporal boundaries were most salient, such as weekly sprints, monthly milestones and the programme's duration. What each participant worked on within these periods differed, yet the group's schedule remained consistent and all shared the same deadlines – particularly the Friday morning showcase to PlayHouse's Creative Director. Accordingly, the flow and affect of the space took a predictable shape with anxiety levels and working hours increasing from Tuesday until early Friday morning when it peaked as the team scrambled to get a working and playable build ready for presentation. Inevitably, Thursday nights were very late nights but also brought the team together, alone in the studio.
4. Standardized forms are the fourth category of boundary objects, allowing for the creation of 'immutable mobiles', objects that can be transported over long distances without changing the nature of the information conveyed.[4] The standardization of methods, in particular, allows amateurs and professionals to find a common ground in their tools (without necessarily needing to understand the 'what or why' – the theory – behind them). In game development, standard specs and formats (e.g. formats such as .jpeg or .tiff or maximum polygon limits for art assets) are common examples.

Software tools operate as a translation method, enabling developers to ideally create immutable mobile assets that retain form regardless of whether they are imported into a scene during development, or downloaded by a player thousands of miles away and years later. However, game software also operates as a boundary object. For example, Unity 3D was flexible enough that all participants used it on a daily basis, regardless of their area of expertise, yet robust enough that despite differences in their skill levels, role and their familiarity with the software, each developer could add assets to the game.

## Boundary objects in flux, boundary objects as agents

In returning to the question of how a heterogeneous and largely neophyte team developed a full game in such a short period of time with very little oversight or instruction from the professional developers around them, it becomes clear there were key boundary objects that played the role of project leads and instructors, such as Unity 3D and 3D Studio Max. They became key agents in both 'apprenticing' developers (indicating what skills each needed to create and combine assets, providing help messages and cues when things went wrong) and 'socializing' developers (creating a focus for team discussion

and aligning team members with a common purpose). Unity 3D and 3D Studio Max played a very active role in shaping not only the game but also how the team interacted with each other.

While tangential, perhaps more interesting are the elements that *did not* become boundary objects. In particular, concepts such as the 'target player' and 'fun' could have ostensibly played an important role in the design of the game, given that they occupy such a large part of game development literature. However, these terms were commonly evoked to settle arguments, and otherwise ignored in favour of constructs that were more easily defined and operationalized. For example, in deciding which weapons and attacks to implement in the game, 'fun' and what the player might like did not arise as topics of discussion, whereas the time schedule, ease of programming, and how similar mechanics were employed in other games all played important roles in the decision.

Game software becomes a key site of social collaboration. But, as the opening vignette demonstrates, it can also be a site of messiness, negotiation and resistance. Following both Lee (2007) and Griesemer (2016), much can be learned by examining the moments where boundary objects break down. This messiness occurs for a number of reasons. First, developers redefine and modify software applications after development. Boundary objects such as Unity and 3D Studio Max are not 'stabilized' after release. While marketing materials, game design texts and online tutorials depict them as static and complete artefacts with a built-in array of fixed and predictable structures and tools, the assumption of technical stability, completeness and predictability breaks down during ethnographic investigation.

Not surprisingly, developers modify the technology in ways that are prescribed (see Akrich, 1992) by the software designers; for example, in Unity, the programmers created additional layers of 'plug-in' tools, scripts for shortcuts, and simplified interfaces for designers that obscured the underlying complexity of Unity and instead highlighted desired usages (e.g. creating interfaces that allowed designers to quickly modify the AI of characters, enabling them to 'balance' their speed, strength and health by toggling a few sliders). Developers also work around software constraints, employing tricks and hacks to adapt available technology to their needs (Montfort and Bogost, 2009).

While the constant updating and modification of software tools contributes to some of the messy sociomateriality of learning game development, a further site of negotiation and resistance is when boundary objects themselves push back and resist the commands of the developers, becoming active agents in the game design process. On site, again and again, Unity 3D, 3D Studio Max and other tools such as Photoshop are anthropomorphized by the team, especially when things were not going according to the user's plans. For example, Jessie speaks quite vehemently about Photoshop and Unity both being 'retarded' when they are forced to work together. Every day, software is depicted as recalcitrant, wilful and needlessly abstruse. However, these negative characteristics are balanced by framing the software as something magical in the sense that it is so complex as to be fully unknowable, definitely not something simply taught or learned quickly, but something arcane that one must tread lightly around (e.g. both Sam and Erin were both extremely careful not to touch or even go near certain 3D Studio Max menus – as if they are Pandora's box). Returning to the vignette that opened this article, let us return to story of the shark. When we left, it was a sad, deflated accordion:

When Clark cannot re-inflate the shark, Alex, its creator, is quickly brought in to troubleshoot. Manipulating the shark, he comments quietly '*It's a flipper baby*'. The mutations, instead of being fixed, are growing worse. Alex knows that the scale is somehow distorted, but none of his mouse clicks are successful in reverting the shark to its natural form. Half the team now crowd around Clark's monitor.

Erin suddenly speaks up: Oh I know why. It's because you didn't reset your scale.

Alex: Can you show me how?

They move back to Alex's desk and I follow, leaving Clark and Martin to experiment with the shark on their own. Erin saves the shark and opens a new version to experiment with de-boning the rigging skeleton and using modifiers instead. She replicates some of her work (resetting vertexes, etc.).

Erin: [mumbling to herself] … I wish I knew how to … no …

She says to Alex: How did you do it in the modifier?

Erin starts experimenting with benders, showing Alex how she can move the shark without using animations. Alex likes the benders. '*It's more subtle*' than the bones and animations he embedded in the shark. The tail bends much further and realistically. They continue experimenting. The problem seems to be solved. Alex states that he'll work on it tomorrow. Erin suggests that they should make one quick version that moves to check if it's compatible in Unity. She guesses it's compatible, but she's never done it before. She starts preparing the shark she's working on for export.The shark suddenly deflates.

Erin: Wow. [She starts laughing.]

Erin: It's a magic part. I dunno. It's like very weird. My god. I've never saw that before.

Alex: This thing is bugging all the time.

Erin: Oh my god. … Let me open it on my computer. Just to see. [clicking. The shark suddenly reverts back to its intended form] Oh. did I correct it? I think I … I dunno what I did, but … okay.

Alex: So magically it's not happening anymore?

Erin: Yeah, it's magical.

Me: It's magic?

Erin: It's magic. I just grabbed the key … I'll save it under another name. Always keep different versions. That's very very weird. I've never seen that before. Wow.

Erin to Alex (referring back to the technical issues that Alex has had since day one, when his computer refused to boot): You're not very lucky with computers. They hate you.

Alex, glumly: yes.

In this case, Alex initially assumed that he made a mistake or that the software had a bug. But as more experts, both programmers and artists, are enrolled to unsuccessfully discover the problem, the interaction becomes imbued with 'magic'. The computer in

general, and the software in particular, is depicted as alive and actively resisting Alex's instructions.

Early studies of both children's and adult's interactions with computers evidence how technology users often personify the technology they interact with (see Fogg, 2003; Turkle, 2005). As Suchman (2003) argues, because computers immediately respond in reaction to input, we tend to see them as a purposeful, and by association, social object. Moreover, the internal complexity and opacity of the computer invites an intentional stance, even for users who possess technical knowledge of the computer's internal workings. As Galison's account of Norbert Wiener and the birth of cybernetics evidences, there is a long history behind the 'gremlins' in our machines. In Wiener's own words,

> The semi-humorous superstition of the gremlin among the aviators was probably due, as much as anything else, to the habit of dealing with a machine with a large number of built-in feedbacks which might be interpreted as friendly or hostile. For example, the wings of an airplane are deliberately built in such a manner as to stabilize the plane, and this stabilization, which is of the nature of a feedback … may easily be felt as a personality to be antagonized when the plane is forced into unusual maneuvers. (Wiener as cited in Galison, 1994: 246)

In short, we commonly respond to self-stabilizing, black-boxed machines as if they exhibit consciousness of will and purpose.

While anthropomorphic machines have precedent, this *is* a new understanding of a boundary object: as something that is alive and recalcitrant. And rather than operating according to rational dictates, when 'misbehaving' it was seen as acting in ways that were 'magical', mysterious and indecipherable – a ghost in the machine. This was the role of 'voodoo software', a term arising from this fieldwork which resurfaced again and again when developer's input into the software resulted in inexplicable forms of output.

While its origins are unknown, 'voodoo software' is an actor's category arising from developers to refer to software seemingly exhibiting a mind of its own and operating in a manner counter to users' input and goals.[5] 'Voodoo' is generally applied when a developer's input results in unanticipated output, and the cause-and-effect result of interacting with the software boundary object is not predictable. Voodoo occurs more often, it seems, with intern developers but arises across domains, teams and experience levels. Voodoo software is not the same as a software bug, in that bugs are the result of an error or fault in the software itself. The cause of the voodoo is unknown, but is believed to be operating under rules that are hidden, unknowable, undiscoverable or unfathomable at the current time (and with the developer's current level of enlightenment).[6] However, voodoo must be dealt with before the developer can continue with their task. After the fact, voodoo occurrences are rationalized and attributed to a bug, human error, software incompatibility issue or something else entirely. But in the moment, the voodoo frame directly shapes developer's interactions.

Commonly, voodoo is linked to 'magic', mysterious forces that influence the course of events. In this sense, 'magic', a common referent, seemed to be applied to smaller effects that could not be readily explained but did not exhibit the same degree of personification, intent or agency. Categorizing software behaviours as 'weird' often preceded the labels of magic and voodoo, and the label was quickly applied to many phenomena. Generally, 'weird' was used when members initially noticed something was not working

out as planned, but had to investigate and determine the cause (i.e. human error or software malfunction). Often, 'weird' effects were just as quickly deciphered and fixed, with no agency being ascribed to the software or hardware. As we see in the excerpt above, once the shark resists Sam and Clark's interventions as well as Alex's, it shifts classification from 'weird' to 'magic', and the computer gains intentionality.

Ascribing magic and voodoo agency to software makes sense when we think of Suchman's (2000) *Plans and Situated* Actions:

> Intentional explanations [i.e. the computer intends to do this action] relieve us of the burden of understanding mechanism, insofar as one need assume only that the design is rational in order to call upon the full power of common-sense psychology and have, ready at hand, a basis for anticipating and construing an artifact's behavior. At the same time, precisely because the mechanism is in fact unknown, and, insofar as underspecification is taken to be characteristic of human beings (as evidenced by the fact the we are inclined to view something that is fully specified as less than human), the personification of the machine is reinforced by the ways in which its inner workings are a mystery, and its behavior at times surprises us. Insofar as the machine is somewhat predictable, in sum, and yet is also both internally opaque and liable to unanticipated behavior, we are more likely to view ourselves as engaged in interaction with it than as just performing operations upon it, or using it as a tool to perform operations upon the world. (p. 606)

When unexpected software effects occur in-the-moment, the first questions asked by the interns were 'Is someone doing this? Did I make an error?'. If human error was not quickly determined, the next question was 'Is the software doing this?', thus ascribing agency to the technology. This led to further investigation and efforts to isolate the issues (e.g. compatibility, formatting/exporting, human error). As we see in the final vignette, if the issue is still not resolved, fault is ascribed to the software or hardware, the work is abandoned, and new, less conflictual, development routes are sought out:

> Erin continues manipulating and experimenting with the shark model. There are now three sharks on screen as she bends them in different ways to later compile them into one moving shark. The shark she's clicking on suddenly flattens.
>
> Alex: What the fuck?
>
> Erin: [laughing.] Wow. I don't understand. Your sharks are so weird. There are weird things happening here. Magic.
>
> There is silence as Erin collapses the sharks into one moving one, saves and exports it as a test version for Unity 3D. They move over to Clark's desk to try it out, nominating him, a programmer, to import the shark. They already have one set of recalcitrant software to deal with – 3D Studio Max – and don't want to deal with another. Clark opens it. It's a massive, flat shark. It's not moving. Dead. But they have no idea what the issue is. Alex guesses that there are multiple issues.
>
> Alex imports another shark. This one looks okay. The right size. No deformities. But the shark 'teleports' – appearing far away from the gridded spot on the 3D plain that Clark is working on. It is an entirely new problem, but it is still dead. No animations. Back to square one.

Despite spending days of effort, Alex ultimately chooses to abandon his work on the shark. It is removed from the game and not replaced. 3D Studio Max 'wins' this battle. This not only changes the game but also slows the whole team's progress. Ultimately, other design and narrative elements in the game are cut to make up for the absence of this central enemy class, but also for lost time.

## Discussion

The above series of vignettes evidence the role of software as a boundary object uniting the team but also highlight how boundary objects can become a site of negotiation. Importantly, the negotiation at stake was not only between developers but also between developers and machines. Alongside the human developers, 'magic' and 'voodoo' took an active role in game design, circumventing intended plans and complicating any assumption of designers-as-omniscient-gods, as well as the belief that software tools are easily black-boxed. Game making, from this perspective, is mysterious and magical because designers need to interact with this software, which itself acts as a key boundary object.

Most often, development took the trial-and-error approach of team members coaxing recalcitrant agents – their software – into working in alignment. While this is true of seasoned professionals as it is of interns, it is by looking at interns we gain insight into how one learns to channel voodoo as both a material and social accomplishment. This is because harnessing and ideally exorcizing voodoo facilitates meeting deadlines and problem-solving, as well as developing reputation and respect within the team.

The pace at which one develops reputation and respect within the team is distributed unequally according to the community of practice one is affiliated with. Learning 3D asset creation, rigging and animation for the first time, perhaps it's not surprising that Alex experienced difficulties, as complex and powerful tools such as 3D Studio Max can become a house of cards. When playing with options not fully understood, such as touching a wrong button or saving at the wrong moment, the illusion of a perfect asset – what appears to be a healthy shark – can fall apart or disappear altogether and days of work is lost. This was often experienced by artists. In contrast to programmers, who could use debugging tools and go through their code line-by-line when attempting to find out why the software acted in unexpected ways, artists' only step-by-step log of events was using the 'undo' button to revert changes. Going back and changing something earlier in the history often meant abandoning all the work that followed. While code could often be corrected, art was not. This is echoed by O'Donnell's (2014) work on tools engineers and their respective importance in both debugging and uniting the work of artists, designers and programmers. Importantly, this links to Bechky's (2003) finding that boundary objects may reinscribe pre-existing power relations, ascribing interpretive power to engineers and programmers more often than other users.

However, none of the team members ascribed the issue with the shark as being the result of Alex's inexperience. This provides a counter-example to Bechky's (2003) finding that when boundary objects fail, blame is often displaced onto those of lower status in the workplace hierarchy. Rather than blaming or asking Alex what he may have messed up, team members took an entirely different approach, instead quickly granting

intentionality and capriciousness to the software itself. This difference in approach may be rooted in the shift from more hierarchically based organizations to the comparably 'flat' structures that now characterize small-scale software development and the need to preserve working relations in the absence of overt authority. Regardless, these scenes illustrate how 'voodoo', 'magic' and 'weird' happenings can be used in-the-moment to corral the negative social effects that result when one member's contribution to the game refuses to work or to fit nicely within the larger structure, thus preserving team cohesion. These terms denote that the events are uncontrollable and unavoidable. Effectively, voodoo allows the developers to save face, asserting that the undesired outcome is not their fault, nor is the resultant delay in work due to their own inexperience. Categorizing something as 'voodoo' allows team members to move on as a group, dealing with software issues without needing to ascribe blame to a team member's error or to expose another member's lack of expertise.

Rather than just assuming the role of an inanimate tool that is wielded at will, the role of software in game design becomes something much more complicated – a finding that only emerged through ethnographic observation. In the field, when the team members were huddled in a semicircle of rolling chairs, fixated on screens showing mutated sharks, 'ghost' images of teddy bears or objects falling endlessly through the scene, 'magic' and 'voodoo' are commonly bandied about with smiles and laughter. No one has a solution, hence it is simply voodoo. That's okay. In exit interviews, this voodoo disappears. It is nowhere to be seen and is completely masked or overlooked by participants who describe more rationalized depictions of the events. This process of tidying up accounts is illustrated in the exit interview with Tyrone, the project manager, who in a conversational tone was listing a laundry list of unplanned 'features' that arose from code, such as inexplicable enemy AI movements. After chuckling over a couple of other examples, he reflects,

> 'Code is weird. Code has a mind of its own. It does whatever it wants and you just …'
>
> Here, Tyrone abruptly stops. He glances at my notebook. He continues, again, but this time in a different, quicker tone: 'Well, no. That's not true. It does *exactly* what you tell it to do and you have to seriously sit down and think about why it's doing it that way because you can't take any steps that – you can't skip any steps in your head 'cause if you skip steps it's gonna create a bug.'
>
> He pauses, confident now, with a nod of his head: '*That's my opinion'.*

Perhaps the experience of voodoo, and then afterwards rationalizing and tidying up accounts of the development process, is more common with interns – they are currently looking to prove their expertise to gain employment. Therefore, admitting that they have not mastered boundary objects such as Unity 3D would seem counterproductive, and mentioning 'magic' or even 'weirdness' would seem downright silly. Presumably, as developers gain experience, they get better at learning the situated trouble-shooting and problem-solving processes required to re-align recalcitrant technology. For example, seasoned developers are more likely to have run into a similar problem, read more forum posts on the common issues with that software, and know which team members are more

prone to human error. Thus, they are able to quickly pinpoint cause-and-effect sequences without needing to apply the 'voodoo' label. This process seems to align with the experienced programmers described by O'Donnell, who when confronted with 'system failures', use debugging tools to observe game behaviour and make visible the broken feedback loops and improperly integrated pipelines. This ability to make work visible is granted by engineering tools, but not art tools, meaning that artists run into unsalvageable issues more often. Importantly, O'Donnell notes that younger developers often make many minute changes to the code with little analytical foundation. These changes often backfire, ending up in an 'unreturnable past' and thus a 'death of a 1000 cuts' (O'Donnell, 2014: 107–118).

However, voodoo is not restricted to new developers. In speaking with experienced PlayHouse developers, 'voodoo' was a term they all applied to their work. As a senior designer put it, voodoo does occur, but it appears less often with in-house made tools because the studio's own game engine is produced in-house. Thus, when the seemingly inexplicable happens, he goes to the person behind the technology to figure out what caused the unanticipated effect. It's not a ghost in the machine, but rather a man *behind* the machine. However, another senior developer in a separate interview had a slightly different experience of voodoo. While stating that tools built in-house generally work as intended, he felt PlayHouse's internal game engine exhibited voodoo behaviour as a function of its age. The engine was over 12 years old, and there were 'parts no one's touched in a million years', and in the entire company no individual person existed that understood how the entire thing worked, indicating that as system complexity grows, so too do occurrences of voodoo agency.

When creators and experts are absent, then 'voodoo' becomes more commonly applied, highlighting for brief moments how boundary objects can push back on their users, and in doing so are granted agency and will. Yet, a third senior designer posited that turning to magic and voodoo is a natural instinct for game developers in particular, given a near universally shared love of science fiction scenarios and fantasy worlds in the media they consume as well as produce. Rather than believe that everything can be rationalized, they would rather ascribe some magic and intentionality to the machines they use every day. Personally, I rather like this interpretation.

## Conclusion

Interpreting game development via the concept of boundary objects allows us to explore the everyday materiality of game development practices. These 'studio studies' give us an anchor from which to investigate creative practice in game studios, which are largely uncharted waters (Ash, 2015). In particular, boundary objects are useful tools that help us theorize how different communities of practice can come together to work coherently on a shared project, despite having different visions of what this project should look like. In this sense, tools such as Unity 3D and 3D Studio Max provide a shared focus and language, helping teams dictate the scope, form and schedule of game development. Technocratic rationalization and instrumentality are commonly associated with software production. But, rather than a mutely obedient tool, software exerts agency of its own and is seen to exhibit magical, even agential properties during game development. This

allows for – and even forces – collaboration in the absence of consensus, uniting the team together. Importantly, while agency is attributed to non-human actors in many Science and Technology Studies approaches, the observation that boundary objects themselves may exhibit agency is something relatively novel (Fleischmann, 2006).

While 'voodoo software' is an extremely useful concept to understand the messy work of game development, as well as the fact that software tools simultaneously act as both boundary objects and agents, this means that the existing literature on boundary objects may need to be re-conceptualized. Their role in game development has additional implications for our understanding of creativity and authorship in the game industry, as the final shape of games is often the result of unintended workarounds initiated not by solitary designers/authors but by the team in response to recalcitrant technology. The more removed the software tool author is from the team developing the game, the more voodoo agency is ascribed to the machine itself.

Ultimately, learning how to develop games is a situated, collaborative practice that is enabled by software tools. Interacting with software engines and art and sound tools is how one makes the transition from a player and consumer of games, to a modder and maker of games. With each world they create, developers accumulate expert knowledge, often brokering this 'knowing how' to interact with both teammates and machines into full careers. The release of low-cost and increasingly powerful game development tools has led to a 'democratization' of game development (Anthropy, 2012), if not more full-time game development careers. Free engines such as Twine, GameMaker, Unity and now Unreal engine, coupled with a plethora of digital distribution channels such as the Apple App Store, Google Play, Itch.io and Steam, to name a few, mean that game-making and publishing is more accessible than ever before, no longer limited to skilled hobbyist and homebrew groups, post-secondary game development program graduates and fully fledged game studios.

This has implications for games culture at large. Tools such as the Unity Engine and their associated paratexts (help forums, tutorial videos and community sites) are boundary objects within individual teams, but they are bringing increasingly wider social worlds into contact with each other as shared protocols and practices are developed – and come into conflict. These tools are agents – our mentors, producers and community organizers – bringing together different communities of practice. They structure how we interact and collaborate with each other and shape what can be said and done, both in the workplace *and* in the game space. However, as these game technologies become increasingly user-friendly and multifunctional, more and more of their complexity and functionality is black-boxed and removed from user view. Accordingly, as the distance between the developer and the tools-developer is increased, we can expect an increasing influence of 'voodoo' software on the design of our games. This, in itself is not a bad thing. However, given the agential role of these software tools, the politics of platforms (Gillespie, 2010) becomes an increasingly salient focus of study, as the creators of these tools design-in prescribed and proscribed uses, influencing the forms of the games we can make and the types of political statements that can be made by them (Harvey, 2015). Ultimately, the tools we choose to use structure how we enter into the world of making play, how we meet and interact with other communities and social worlds, and how we come together to collaborate.

## Acknowledgements

The author would like to thank the two anonymous reviewers, as well as the editorial staff of *New Media & Society* for their insightful comments in revising this manuscript. Any errors or omissions are mine alone. The author would also like to thank Bart Simon, Aphra Kerr, John Banks, Casey O'Donnell, and Jason Della Rocca for inspiring work and conversations on studio studies.

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Social Sciences and Humanities Research Council of Canada (grant number 756-2012-0209) and the Graphics Animation and New Media Network of Centres of Excellence (GRAND NCE).

## Notes

1. All proper names of the research site and participants are pseudonyms to protect the anonymity of participants.
2. Following Schell (2008), I use the term 'game developer' to refer to 'anyone who has any involvement with the creation of the game at all' (p. xxv). This includes engineers, animators, artists, writers, designers, programmers, modellers, managers and musicians.
3. 3D Studio Max is a three-dimensional (3D) graphics software for making animations, models and images that can then be imported into a game engine. Game engines such as Unity 3D economize and streamline game development by providing a set of middleware tools (e.g. a rendering engine for graphics, physics engine for collision detection, and tools for sound, scripting, animation, artificial intelligence, networking, memory management and porting) in one software framework. This allows developers to reuse/adapt the same engine to create different games without having to code the game from the ground up.
4. Star (2010) provides a corrective to the full-scale adoption of immutable mobiles. Standardization only works as a boundary object when there is space to tack between ill-structured hacks and well-structured aspects. If there is complete standardization, then these objects operate more as infrastructure. This addresses Lee's (2007) critique, which is rooted in the assumption that boundary objects must be highly standardized to completely satisfy the informational requirements of all communities of practice involved.
5. The term might have arisen from the 1980s geek cyberpunk culture, referencing both William Gibson's (1986) novel, *Count Zero*, wherein Haitian loa created from the fractured remains of AI reside in computer networks, and Masaune Shirow's (1991) manga, *Ghost in the Shell*.
6. Interestingly, in the medical field, 'voodoo' is used in a very similar way – researchers apply the label to drugs whose effects are inexplicable (Timmermans and Berg, 1997).

## References

Akrich M (1992) The de-scription of technical objects. In: Bijker W and Law J (eds) *Shaping Technology/Building Society: Studies in Sociotechnical Change*. Cambridge, MA: The MIT Press, pp. 205–224.

Anthropy A (2012) *Rise of the Videogame Zinesters: How Freaks, Normals, Amateurs, Artists, Dreamers, Drop-outs, Queers, Housewives, and People like You Are Taking Back an Art Form*. New York: Seven Stories Press.

Ash J (2015) Theorizing studio space: spheres and atmospheres in a video game design studio. In: Farías I and Wilkie A (eds) *Studio Studies: Operations, Topologies & Displacements*. London; New York: Routledge, pp. 91–104.

Bechky BA (2003) Object lessons: Workplace artifacts as representations of occupational jurisdiction. *American Journal of Sociology* 109(3): 720–752.

Bogost I and Montfort N (2009) Platform studies: frequently questioned answers. In: *Proceedings of the digital arts and culture conference*, University of California Urvine, 12–15 December.

Chen M (2012) *Leet Noobs: The Life and Death of an Expert Player Group in World of Warcraft*. New York: Peter Lang.

De Vaan M, Stark DC and Vedres B (2015) Game changer: the topology of creativity. *American Journal of Sociology* 120(4): 1144–1194.

Fleischmann KR (2006) Boundary objects with agency: a method for studying the design–use interface. *Information Society* 22(2): 77–87.

Fogg BJ (2003) *Persuasive Technology: Using Computers to Change What We Think and Do*. San Francisco, CA: Morgan Kaufmann.

Galison P (1994) The ontology of the enemy: Norbert Wiener and the cybernetic vision. *Critical Inquiry* 21(1): 228–266.

Gibson W (1986) *Count Zero*. New York: Arbor House.

Gillespie T (2010) The politics of 'platforms'. *New Media & Society* 12(3): 347–364.

Griesemer JR (2016) Sharing spaces, crossing boundaries. In: Bowker GC, Timmermans S, Clarke AE, et al. (eds) *Boundary Objects and beyond: Working with Leigh Star*. Cambridge, MA: The MIT Press, pp. 201–218.

Harvey A (2015) Twine's revolution: democratization, depoliticization, and the queering of game design. *Game: The Italian Journal of Game Studies* 2014(3). Available at: http://www.gamejournal.it/3_harvey/#.VUO3LRceXOo (accessed 1 May 2015).

Kerr A (2006) *The Business and Culture of Digital Games: Gamework and Gameplay*. London: SAGE.

Latour B (1986) Visualization and cognition: thinking with eyes and hands. In: Long E and Kuklick H (eds) *Knowledge and Society: Studies in the Sociology of Culture past and Present*. Amsterdam: Elsevier, pp. 1–40.

Lee CP (2007) Boundary negotiating artifacts: unbinding the routine of boundary objects and embracing chaos in collaborative work. *Computer Supported Cooperative Work* 16(3): 307–339.

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

O'Donnell C (2014) *Developer's Dilemma: The Secret World of Videogame Creators*. Cambridge, MA: The MIT Press.

Pickering A (1995) *The Mangle of Practice: Time, Agency, and Science*. Chicago, IL: University of Chicago Press.

Schell J (2008) *The Art of Game Design: A Book of Lenses*. Burlington, MA: Morgan Kaufmann.

Shirow M, Schodt FL and Smith T (1991) *The Ghost in the Shell*. New York: Kodansha Comics.

Star SL (2010) This is not a boundary object: reflections on the origin of a concept. *Science, Technology & Human Values* 35(5): 601–617.

Star SL and Griesemer JR (1989) Institutional ecology, 'translations' and boundary objects: amateurs and professionals in Berkeley's museum of vertebrate zoology, 1907–39. *Social Studies of Science* 19(3): 387–420.

Steinkuehler C (2006) The mangle of play. *Games and Culture* 1(3): 199–213.

Suchman L (2000) Organizing alignment: a case of bridge-building. *Organization* 7(2): 311–327.

Suchman L (2003) Excerpt from plans and situated actions. In: Wardrip-Fruin N and Montfort N (eds) *The New Media Reader*. Cambridge, MA: The MIT Press, pp. 600–611.

Taylor TL (2006) Does WoW change everything? How a PvP server, multinational player base, and surveillance mod scene caused me pause. *Games and Culture* 1(4): 318–337.

Timmermans S and Berg M (1997) Standardization in action: achieving local universality through medical protocols. *Social Studies of Science* 27(2): 273–305.

Turkle S (2005) *The Second Self: Computers and the Human Spirit*. Cambridge, MA: The MIT Press.

## Author biography

Jennifer R Whitson is an assistant professor in the Department of Sociology & Legal Studies at the University of Waterloo. In addition to studio studies and ethnographies with game developers, she researches digital media surveillance, social influences on software development processes, gamification, and governance in online domains. You can find her work in journals such as *Surveillance and Society*, *First Monday*, *Economy & Society*, and *FibreCulture*.