Writing Programs and Procedural Creativity: The Possibility of a Literary Platform Studies

Jason Boyd, Ryerson University
jason.boyd@ryerson.ca

Paper presented 6 June 2019 at the Canadian Game Studies Association

Note: Bolded numbers in square brackets correspond to the slides in the accompanying slidedeck.

[2] (Overview)

1. Procedural Creativity and Authorship: A Definition

My and Matthew J. Wells’ concept of and term ‘procedural creativity’ came about as a way of thinking of alternatives to Jeannette Wing’s advocacy for ‘computational thinking,’ an idea that was primarily motivated as a way of selling the teaching of Computer Science as essential to the social good: Computational thinking can solve the world’s problems! Therefore, everyone should learn to think computationally (or program)! University students should become Computer Science majors! Against the idea that the world operates (or should operate) according to computational models, and that therefore it is important that we all learn to think like a computer or a computer scientist, Dr. Wells and I offer the idea of procedural creativity, with a focus on the value of creative exploration, development and use of formalized procedures or processes, which may or may not be informed by a grounding in Computer Science or be computational at all. The concept is meant to be expansive, in a way similar to Anna Anthropy’s definition of a game in Rise of the Video Game Zinesters: “an experience defined by rules”--procedural creativity is learning how the creation of rules or procedures can be deployed for creative purposes, or learning how to be a procedural author.

The ‘Writing Programs’ of the title is intended to point to the procedural creativity that will be discussed in this paper, which focuses on programs or software that are designed for
creative, often gameful writing—software and writing which can involve programming or coding as a component of that creation. I argue that literary studies needs to understand these writing programs if it is to fully understand and appreciate works of procedural creativity like playable stories. I am thinking of such programs as Twine, Inform 7, Inkle Studio’s Ink & Inky, TextureWriter, and Bitsy (to name some).

2. Why Literary + Platform Studies

I approach this question from the position of a proselytizer within literary studies for the greater integration of electronic literature, playable stories and story-rich games into traditional established literary scholarship and learning—not just in specialist courses that focus on and contain computer literature and games from ‘contaminating’ ‘real’ literature courses. In her book Literary Gaming, Astrid Ensslin has noted the challenge to be faced: “[P]lay and games as forms of human interaction and entertainment have had a difficult time in being accepted in contemporary literary scholarship and tend to be viewed with a great deal of skepticism…. Similarly, games studies seems to be seen as a discipline more akin to media and cultural studies and is typically found in institutional isolation from English departments” (7). I believe, in order to effect conversion to the faith, it is not sufficient to only to study and teach works of procedural authorship, but that it is essential to expand this scope to a consideration of the ‘writing programs’ or procedural authoring platforms that are used to create these works. Traditional literary scholars and students typically have misassumptions and misperceptions about playable stories, based on a lack of awareness of the specific conditions under which these creations are produced. In particular, they are not well aware of the parameters of creative possibility in playable stories and the significance of the affordances and constraints imposed by the authoring systems on which they are created. They are also not trained to evaluate how interactant-driven components of playable stories are created and how these components are key in any poetics of playable stories. [4] It is all too easy (as self-contradicting
contrarian Ian Bogost has shown recently in relation to narrative games) to jump to the conclusion that playable stories are crudely constructed, narratively shallow, gimmicky, confusing to get through, and just too far away from the aesthetics represented by the standards of the Great Works of the Western Canon. These works are seen paradoxically as both cryptic (operating according to unfathomable mechanisms) and as superficial (having no narrative sophistication), thus exhibiting both the Eliza effect and the Tale-Spin effect discussed in Noah Wardrip-Fruin’s Expressive Processing. I think these judgments are due in large part because of a lack of understanding of procedural authorship and the platforms used for procedural creations.

There is certainly a substantive body of scholarship on the poetics of procedural literature informed by literary theory. Janet Murray, N. Katherine Hayles, Marie-Laure Ryan, and Astrid Ensslin are just a few of the notable scholars in this field. However, this scholarship almost exclusively analyzes the creations and not what was employed to create them. Matthew Kirschenbaum, in “Editing the Interface: Textual Studies and First Generation Electronic Objects” [TEXT 14 (2002), pp. 15-51], in making the case for bringing the insights and methods of book studies to the study of born-digital literature, observes that “a bibliographical/textual approach calls upon us to emphasize precisely those aspects of electronic textuality that have thus far been neglected in the critical writing about the medium: platform, interface, data standards, file formats, operating systems, versions and distributions of code, patches, ports, and so forth” (27). Yet his discussion is confined to born-digital literary creations and how to edit them, which, and, while it does touch on questions of authoring software and their constraints (eg, HyperCard), is focused on editorial dilemmas when faced with works created in earlier technologies that are being edited by current technologies, rather than the creative affordances of the software platforms.

Noah Wardrip-Fruin, in Expressive Processing, made a significant step towards making a case for the study of procedural authoring systems, particularly in Chapter 5, on the Tale-Spin
Effect. He notes that both Espen Aarseth and Janet Murray misunderstand James Meehan’s story-generation program *Tale-Spin* because they only read the output of the system and not the system itself: “[A] close examination of the system’s *Tale-Spin’s* operations reveals something much more intriguing than either author [Murray and Aarseth] assumed—making a clear case for the necessity of developing an approach to reading what processes express.” [5]

Later in the chapter, he notes: “[A]dopting only the audience’s perspective makes full engagement with the work’s processes impossible. Some systems, through interaction, may make it possible to develop a relatively accurate hypothesis of how the internal systems operate (in fact, some works require this on the part of the audience). But this is a complement to critical engagement with the operations of the work’s processes, rather than a substitute. Failing to appreciate this leads to missed opportunities such as those of Aarseth and Murray with *Tale-Spin.*” While Wardip-Fruin makes a convincing case for examining authoring platforms to truly understand their outputs, all his examples in *Expressive Processing* are systems that were created to automate textual and other output—they are text generation or story simulation systems, not systems that are designed to enable humans to engage in procedural authorship.

[6] This survey of the literature led me to the field of Platform Studies and the monograph series of the same name. Platform Studies focuses on “the investigation of underlying computing systems and how they enable, constrain, shape, and support the creative work that is done on them” (Salter & Murray, Series Forward). Most people, when Platform Studies is mentioned, will probably think of the studies on the books in the series on gaming systems like the Atari, the Nintendo Wii, the NES and the SNES, or early personal computers such as the Commodore Amiga and BBC Micro. These individual hardware/OS systems are quite different from platform-independent software like *Twine.* [7] However, one work in the series is much more closely aligned with authoring platforms like *Twine:* Anastasia Salter and John Murray’s *Flash: Building the Interactive Web.* Salter and Murray note how their book differs from previously published books in the series: “platform studies is concerned with investigating
the connections between hardware, software, and creative works. Flash, consisting mostly of virtualized hardware, is inherently more generalized in its hardware needs. As a software platform, Flash exists in a more abstract space than consoles or personal computers.” The authors go on to state that "we will be using a case study model and looking at a few significant examples chosen for their suitability to illuminate Flash’s affordances and the role those affordances have played in constructing our experience of web multimedia." Salter’s and Murray’s study of the history and affordances of Flash resonates strongly with a platform like Twine, and their case study model seems most appropriate to study these authoring platforms. (I should add here that Salter has another useful case study model in her book *What Is Your Quest? From Adventure Games to Interactive Books* (2014), in particular, chapter 5, “Fan Games,” which examines the platform Adventure Game Studio)

4. Case Study: Twine

[8] In the limited time I have today, I propose to discuss the hypertext/hypemedia authoring software *Twine*, particularly in the context of teaching *Twine*-created works and using *Twine* for the major assignment of an undergraduate course, “Narrative in a Digital Age,” as one way to approach a literary platform studies. *Twine* was created by Chris Klimas, and from the interviews with him that I have read, he is not particularly illuminating about what his motivation and vision was in creating *Twine*. Although the title of an essay by Astrid Ensslin and Lyle Skains, “Hypertext: Storyspace to *Twine*” [*The Bloomsbury Handbook of Electronic Literature*, edited by Joseph Tabbi. Bloomsbury, 2018, pp. 295-309], might lead one to suppose that Klimas intended to create a more accessible version of Eastgate Systems’ hypertext authoring platform, he never mentions *Storyspace* or the now canonical works of ‘serious hypertext’ by authors such as Michael Joyce, Shirley Jackson, or Stuart Moulthrop. Indeed, his own creative work predating *Twine* was Interactive Fiction, parser-based works created using programs like *Inform*. He also seems not to have envisioned or understood *Twine* as a game-making platform
until Anna Anthropy’s championing of *Twine* as such in *Rise of the Videogame Zinesters*. In a 2014 interview with Leigh Alexander, Klimas recalls that "At the time, I had been experimenting with ways to create hypertext that were strongly code-oriented. I was studying interaction design, so *Twine* was my attempt to make something that would be friendly to people who were writers more than coders." (Klimas, qtd in Alexander). He later observes that "...moving from a codeless story to one that, say, has a little conditional logic is painless. You can add it when you're ready, and you can dive into as deeply as you like, even to the point where you are writing your own JavaScript to extend the runtime itself" (Klimas, qtd in Alexander). So, one inference that might be made is that Klimas conceptualized *Twine* as a authoring platform that did not necessitate, but could facilitate a gradual incorporation of programming into what was structurally a hypertextual story, making a story more gameful or playable.

So, the question that I would like to tackle in this case study is this: How could a post-secondary literature course effectively teach (that is, convey the full measure and richness of) something like, for example, Porpentine’s *howling dogs*? [9] It is all too easy (however one might appreciate the skill of Porpentine’s writing) when transversing this work to peremptorily conclude, without knowledge of the *Twine* platform, that, on a procedural level, it’s just a ‘bunch of links.’ A first step to a more informed appreciation of *howling dogs* or any other *Twine* work is to point out that there are three levels that need to be examined. [10] The first level is the browser-rendered or published view, the most opaque from a construction point of view; [11] the second is the *Twine* authoring interface view (in which one can see what the author actually created using the *Twine* platform), [12] and the third level is the ‘Build’ view, where one can see the HTML version, including the *Twine* ‘engine’ script.

[11] From a high-level node map of *howling dogs* (in the *Twine* authoring interface), a reader can quickly see that there is probably something more interesting going on than ‘just a bunch of links.’ Clearly, there is a meaningful structure of passages and links as indicated by the clustering and lines. Since a creator can drag passages to any place on the map view, the node
map can often reveal the author’s own conceptualization of the narrative structure. [13] *Twine Garden*, a *Tumblr* blog curated by Porpentine, is a gallery of Node Maps created by *Twine* authors that contains some highly elaborate maps that often double as abstract works of geometric art. In the opening and core setting of *howling dogs*, the space in which the player character is confined and whose movement is constrained in various ways (particularly the sequence in which one uses the facilities), operates and degrades according to variable tracking and conditional statements, not just a linked series of passages. [14] Each return to the space is not a completely new repeated text, but the same text in which certain states are triggered according to variables such as how many times the player has woken up in the space (likely using Twine 1’s visited() function with the <<if>> macro).

One approach I have taken to understanding *Twine* works like *howling dogs* and *Twine* as a platform is to approach creating on *Twine* starting not with a story idea but with a consideration of the affordances available on *Twine*. [15] An unfinished *Twine* story/tutorial I created, “Agent of GLAM,” tried to take as many macros and functions available in *Twine* as possible and find ways to include them in a story. [16] So, instead of asking, “How can I use *Twine* to realize my story idea?” I was asking “How can I incorporate this *Twine* macro or function in a story?” In essence, it is a code-driven story.

There are three import features of *Twine* that are important to take in account in a Literary Platform Studies account. [17] The first is the Story Format, which is independent from the *Twine* editor (although the editor has a default format included with it). Because these Story Formats come with a CSS and thus a particular visual aesthetic, it is often assumed that this is their primary function. This is not the case: they “determine the markup and code you use to write your stories. They allow you to add conditional logic, display effects, and other interactivity” (*Twine* Wiki). Story Formats have been created to add RPG functionality, for stretchtext, for chat-style stories, and for interactive comics. In a 2015 interview with Tory Hoke, Klimas observed: “To me, *Twine* 2 is a common editor with a set of story formats, each with their own
set of capabilities. And honestly, a lot of the interesting parts are in those story formats… It’s a much different situation from Twine 1, where choosing a story format was not very perilous. In Twine 2, the formats are so vastly different that it’s no longer a question of ‘what do you want your story to look like?’ but ‘what do you want your story to be able to do?’” (Klimas, qtd in Hoke). Story Formats come with their own built-in functions and macros and scripting conventions that will not always (if at all) function in other formats. Therefore, understanding the affordances of the Twine editor is shaped by the selected Story Format.

[18] The second important feature of Twine is the use of CSS to change the default CSS included in Story Formats. Twine 1’s default CSS is white text on a black background. [19] But this can be drastically changed, and have important narrative effects, as shown by this screenshot from “Agent of GLAM.” [20] The third feature is the macros and functions that can be employed by an author in a Twine passage. These two examples show how much content of a passage can be coding and not the text that the reader encounters. [21] These two passages plus a third end up like this [see slide 21] in the browser level.

While this code-driven approach to Twine-based procedural creativity is not necessarily conducive to great literature (assuming that category doesn’t cover campy space opera), it does have the value for scholars of developing a quite sophisticated understanding of the affordances of Twine (as constrained by the selected Story Format). Once one understands what is involved in constructing a choice-based menu that tracks variables, it becomes impossible to perceive and dismiss a Twine browser-rendered page as ‘just a bunch of links.’ [22] A more thorough and probably more useful approach can be seen in Melissa Ford’s Writing Interactive Fiction with Twine (2016) (which uses Twine 2), where chapters focus on specific authorial/literary goals—e.g., “Creating a Vivid Setting,” “Constructing Believable Characters,” “Developing a Strong Player Character,” “Balancing Pacing and Action”—which describe how certain functions and macros can help to achieve these goals within the context of playable stories. In effect, Ford’s manual is a type of Platform Studies book, in that it provides a systematic and
comprehensive description of Twine 2’s functionality and how these functionalities can be employed to achieve literary effects. In other words, it is a practical study of the potentiality of Twine 2 for procedural authorship.

A Literary Platform Studies approach, I have recently realized, would enable me to deal with a problem I have had in using Twine and other authoring platforms as an essential component of the major assignment of “Narrative in a Digital Age.” I felt it was necessary to reserve a not inconsiderable part of class time to tutorials on how to use various authoring platforms, but not as much as might have been needed, given that the course primarily focused on studying works of procedural creativity and relevant secondary literature. I now see that I can make a platform like Twine itself an object of scholarly study and discuss it in class along with the works created on it, perhaps asking students to try and create themselves certain procedural aspects in the Twine works being studied as a preliminary exercise prior to class discussion.

In conclusion, I think that a Literary Platform Studies that takes as its objects of study software and platforms intentionally created to enable the procedural authorship of playable stories addresses an unmet need and is one way of addressing the previously mentioned skepticism that Astrid Ensslin has noted literary studies scholars, teachers, and students have about computer-based literatures. Such a focus can reveal the craft and the capacity of playable stories as they are shaped and enabled by these platforms.