Conventions as Metacommentary game engine.

Ibragimova Dildora Shamsiddinovna
Samarkand state Architectural and Civil engineering institute. Uzbekistan, Samarkand

Raupov Kahramon Maxmudovich
Samarkand institute of physical culture and tourism.

Abstract: *In this paper, we aim to address this issue by discussing online communities surrounding consumer-grade game engines. We will examine how conventions and canons pertaining to the use of the engine emerge in these communities—and how they are then challenged and subverted. Building on an autoethnography of my own experience as a hobbyist game developer, we will describe how community members develop games that probe the boundaries of the engine's functionality and will discuss their motivation in doing so.*

Keywords: *game engine, game development, hobbyist games, participatory culture, platform studies, creativity, innovation, autoethnography.*

Introduction

It will argue that the act of subverting established conventions of using a game engine can be considered a metacommentary on the engine's expressive capabilities and intended use. This metacommentary can provoke wider reflection within the community and contribute to the development of the engine itself. On a wider level, the question we will address through the prism of game engines is this: how can a creator challenge the constraints imposed by the tool they are using and the community they are part of, and what is the cultural significance of such subversive practices? Before posing this question, however, we will first discuss what kind of constraints are at stake when using a game engine.

Main part

The game engine and its constraints recent decades have witnessed a dramatic transformation in the mediascape, involving the blurring of the dichotomy between producer and consumer and increased accessibility of media production tools. In response, a number of useful, if overlapping, theoretical paradigms have been put forward to make sense of this transformation: "prosumerism" and "participatory culture" "maker revolution" In the real of digital games, consumer-grade game engines (also known as "game makers") are one of the most prominent examples of our changing relationship with media. A game engine is software that accelerates game development, based on the idea of a "common substructure for similar games" .Substructure in this context refers to lower-level stock routines, such as video and audio output, player controls, saving and loading functionality, world physics, and so on, which can be abstracted from the game proper. As these routines can be vastly

different depending on the genre, hardware platform, and so on, game engines are typically geared towards a specific kind of "similar games".A game engine is thus a platform that enables and simultaneously constrains games made with it. Adventure Game Studio is called that because it incorporates most stock routines found in point-and-click adventures: a dialogue system, usable objects and inventory, mouse-controlled playable character, and so on.

Analyses

Puzzle Script is so effective for puzzle games because its entire syntax is based on an abstraction of the genre's typical features (two-dimensional tile-based map, four-directional movement, movable objects, etc). But when one attempts to accomplish something a game engine is *not* intended to do, they are likely to find it difficult or even impossible. You can, for example, create a platformer with Adventure Game Studio. But you would have to manually implement a lot of functionality not covered by the engine itself: gravity and jumping, attacking and taking damage, enemy AI, and so on. At the same time, you would not be taking advantage of many of the tools' built-in routines. Even more universal tools like and Game Maker can be challenging to use for purposes not intended by their developers: both, for example, would make it much more difficult to make a text-based game than, say, *Twine*, which is specifically designed for hypertext fiction. This focus on a specific kind of game is what to conclude that "game engines regulate individual videogames' artistic, cultural, and narrative expression" much more than, say, genre tropes do. But beyond the "material constraint" of the game engine's functionality there is also an immaterial constraint at play: that of conventions and canons associated with the engine. Some of them derive from official paratexts associated with the tool, such as the title and the manual. Tools such as it and make it clear what their intended use is through their very titles. This is reinforced and concretized by tutorials and examples shipped with the engine. As a result, many games made with a game engine are either recreations or modifications of examples provided by the developers. In between paratexts and the functionality of the engine itself are ready-to-use plugins, add-ins, and scripts included in the distribution. While not part of the tool proper, they are likely to be utilized by many of its users. For instance, many platformer games made with *Stencyl* have a very similar control scheme and "feel" because their authors made use of the tool's standard platformer movement behavior. Equally important to "official" paratexts are those made by community members. Online communities of users play a crucial role in regulating how "game makers" are utilized. Tutorials and examples created by community members and recognized by their peers as successful help establish a "canon" which subsequent contributions are judged against. When a newcomer asks for help, experienced members will often use that canon as a reference point, making adhering to it seem desirable. Sometimes, a failure to conform to the canon can lead to criticism from other community members. But perhaps even more important are actual games made with the engine. Typically, the first thing you do when you discover a game maker community is play some of the games made with it (usually those that community leaders selected to feature on the website, message board, etc.). Moreover, the more successful games made with a game maker can serve as "ambassadors" for the tool, bringing in newcomers interested in creating something similar. In such a case, their vision of the tool will initially be shaped by the game that attracted their attention. Thus, for example, if most games made with a certain game engine have been sidescrolling shooters, it can generally be expected that subsequent creations based on the engine will also belong to that genre. And

yet, despite these material and immaterial constrains, there are plentiful examples of game engines being put to unexpected and improbable uses that challenge conventions associated with all the features expected of their genre and then some: gravity and jumping, weapons and combat, stealth… None of these are built into *Adventure Game Studio*, so they must have taken considerable effort to implement. This did not discourage the author despite tools better suited for platformer development being readily available online. *r*, a tool known for its limited text capabilities, and seems to use raster sprites to emulate DOS text-mode. Several *RPG Maker* enthusiasts have developed complex scripts enabling the use of 3D graphics despite the engine itself having strictly 2D capabilities.

Similar examples can be found in nearly every online community centered on a popular game engine. But why are people willing to invest time and effort into making them? We will attempt to answer this question through an autoethnography of my own experience as a hobbyist developer.

Autoethnography refers to "research, writing, and method that connect the autobiographical and personal to the cultural, political and social context. As a method, autoethnography embraces the subjectivity of the researcher's experience by recognizing its grounding in wider sociocultural context. Thus, through a reflexive account of their personal experience, the researcher is able to unpack the meanings and complexities of this wider context The following section is a brief retrospective autoethnography of my engagement with a Russian-language text adventure development tool called URQ. I became an Internet user in my pre-teens, shortly after the turn of the 21st century. For the following few years, my modem became a narrow peephole into paces of Web 1.0, a brave new world comprised of personal webpages with inconsistently encoded text, blocky JPEG images, and outlandish GIF animations. our early experiences of being online were full of excitement, but also filled with frustration and anxiety. Frustration, because there was never telling when the connection would drop, rendering several hours' worth of downloaded data useless as the progress bar stopped short of theal mark. Anxiety, because there was no telling how my parents would react to another astronomical phone bill. Like most kids my age, the thing we was interested in most were games. Yet downloading even old graphical games on a dial-up connection was an ordeal, and contemporary ones were decidedly. ("Universal Ripsoft Quest")[1] was a Russian-language engine for "Choose Your Own Adventure"-style menu-based text games (also referred to as interactive fiction). While the original version of URQ had been developed for Windows, the newer interpreter was, counterintuitively, a console application originally written for DOS. This meant no graphics were supported. The text-mode games looked no-frills even by the day's modest standards, yet at its peak URQ was perhaps one of the better known game creation tools in Runet (the Russian-speaking Internet community). There were two reasons for the tool's popularity. First, the absence of graphics resulted in very small download sizes. Unlike their graphical counterparts, URQ games could be downloaded by the dozens in a matter of minutes—and the selection, if not necessarily the quality, was good enough. The second reason was the simplicity of URQ syntax, which made it easy even for a beginner with no programming knowledge to make a game of their own. All they needed to do was open a text editor and start typing something like this.

And, of course, URQ was much more than a programming language and an interpreter. It was a virtual community of interest with its own traditions, classics, and celebrities. And its sociality was

[1] At the time, the URQ community largely revolved around the now-defunct official site urq.ru. The current official site is urq.plut.info.

perhaps what made it more appealing than more advanced game creation tools lacking a full-fledged community.

That said, the enthusiasm of many authors was not perhaps matched by the quality of their output. The gameplay was often tedious, too diligently recreating daily routines such as getting up in the morning and packing for school. Poor writing was exacerbated by poor spelling. On top of that, many authors seemed to take a kind of pride in violating at every opportunity: many seemingly reasonable choices would unexpectedly lead to death or the game closing off with no possibility to win. At times, the typically short length of URQ games could indeed seem like a relief. In a 2007 interview with SPAG (an English language magazine dedicated to interactive fiction), one of the "old-timers" of the URQ community, a Ukrainian who went by the nick Goraph, reflected on his experience of those early days: "I just did what I liked. The very process of game creation was fun for me, I had no concerns about whether someone would actually play it (and if he would be able to play it at all)."[2]

The lack of established canon did not help, either. Some players still remembered old Russian Choose Your Own Adventure books by Dmitri Braslavsky and others. The few who understood English would have played some foreign works of interactive fiction. But for the majority of URQ enthusiasts, there was no milestone to look up to.[3] The only standard to judge one's works against were the better URQ games, most of which were still flawed. Oftentimes, it was the sheer volume of the game that won the community's respect, signifying the author's determination and the effort that went into the game. Larger games often prompted comments like: "The filesize is impressive, 400 kbytes of text is no joke"[4][5]. And if the prose happened to be readable, the game was guaranteed to become an instant classic.

Not that any of these shortcomings mattered. The burgeoning community churned out dozens of games on a monthly basis. A few of them were good, after all. There were a number of official, semi-official, and unofficial websites dedicated to URQ games. Competitions were organized regularly. Community leaders (the "inner circle" whose position was reinforced by their admin status on URQ's official IRC channel) collected and posted reference materials on improving one's writing and storytelling in an effort to raise overall game quality. There was a sense of movement, a feeling that the community was making progress and a vague hope for a bright future which, just possibly, would involve mainstream recognition.

> We were very optimistic about it, wrote articles, warked on two games simultaneously, tried playing English text adventures again: it seemed to me the only problem of the genre was the lack of publicity, and that in our 'world's most-reading country' IF [interactive fiction] was just doomed to be successful.

For a long time I was a "lurker" in the URQ community: an invisible, passive participant who did not actively contribute to it . I played URQ games, read others' posts on the tool's message board without creating my own account, and made a number of short, unfinished games, which I never showed to anyone. It was only when I seriously started working on a game of my own that I had to

[2] The full interview is available here: www.spagmag.org/archives/backissues/spag48.html.

[3] In the same SPAG interview, when asked about URQ games' perceived lack of quality, one of the community's most prolific authors known as Korwin pointed to "the absence of massive IF [interactive fiction] traditions in Russia".

[4] Comment by ifn00b, posted 27.12.05 at http://urq.borda.ru/?1–0-0–00000133–000–0-0–12086

[5] . (All the forum comments are translated from Russian by the author.)

announce my existence: some of my more difficult code was not working and I turned to the community for advice. Gradually, I found myself more actively engaged with the community, first through the message board and then through the tool's IRC channel.

I then followed it up with another unfinished game that featured a more dramatic departure from URQ's built-in interface. It was a parser-like game with no buttons at all, where the player had to use the arrow keys to construct simple sentences out of a set of verbs and nouns. This change meant I could no longer use URQ's convenient *btn* operator and, in essence, had to write a custom engine with its own interface—inside an existing engine. At that point URQ as a programming language stopped being simple and effective and became clumsy and esoteric. What was supposed to be clean and straightforward code ended up looking like this. Still, I persevered and released a functional demo version. Despite the incompleteness of the story, the game came first out of four entries at the yearly Summer Competition[6], largely thanks to the innovative interface. In fact, one of the more critical comments explicitly stated that in *Thoughts Corporation* there was "an amazing engine but no game." The relative success of my technical gimmicks motivated me to go even further. Inspired by roguelike-style walking demo by a user known as Terracon, I decided to use URQ to make something nobody had attempted before: a turn-based tactics game. While the algorithm itself is straightforward, implementing it in URQ proved a challenge for both myself and the interpreter, resulting in a delay of up to two seconds before each enemy move. The initial release was warmly received by the community[7] and the feedback prompted me to add more features: multiple playing characters, ranged attacks, improved enemy AI. The experiment never got beyond the stage of a tech demo, but it still became a reference point for what one could do with the URQ engine. Following the advent of broadband, URQ players started to drift away, switching their attention to the more appealing Flash and downloadable graphic games. The gradual rise in the general quality of URQ games seemed to have hit a ceiling. The erstwhile enthusiasm about URQ's future started to wane as it became increasingly apparent that in the era of fast Internet connections, text adventures were doomed to remain a niche form of entertainment. Even the release of a new Windows interpreter supporting rich graphics and eventually skin customization could not reverse the tide. The ambition of "making it big", which many authors had cherished, led them to other, more popular genres and platforms. Some of them are still part of the community for the social aspect of it, but do not produce any new URQ games. The URQ community joined forces with other text adventure engines in order to survive and in this form still exists in the more obscure corners of Runet.Personally, I drifted away, despite having been promoted to community and website administrator. I am, however, still in touch with some of the friends I made while active in the URQ community.

It was only years later, when discovering some of the URQ games I had made on an older computer that I (by then a PhD student) asked myself: why had I been so interested in experimenting with the tool instead of using it as intended? Why had I been so willing to write and debug line upon line of unreadable code in order to achieve something I could have implemented much more easily with a better suited tool? Subverting the game engine's conventions from a utilitarian perspective, this does not make sense. If you use a sledgehammer to crack a nut, you most likely do so because there is no

[6] http://urq.borda.ru/?1–0-0–00000210–000–80–0-1212442099.

[7] Full discussion available here: http://urq.borda.ru/?1–0-0–00000200–000–0-0–1191393926. 13 For example, it was cited as such in this thread: http://urq.borda.ru/?1–0-0–00000384–000–0-0– 1316245957.

nutcracker at hand. Yet, with game creation software, this is clearly not the case. There are hundreds of tools readily available, and it is usually possible to find one specifically tailored for the type of game one has in mind. So why would you, presented with a choice between a nutcracker and a sledgehammer, still choose to crack a nut with the latter? When I started to reflect on my own experience in the URQ community, I could think of three main reasons.

We have technologically ingenious game is a way to gain recognition in the community and establish yourself as a "tech guru." This allowed me to circumvent the traditional path to recognition in the URQ community: create an engaging full-length game. My entire URQ output was shorter in terms of text length than some of the more ambitious games for the platform and it mostly comprised unfinished demos. Yet, technological creativeness was a social currency in its own right, quickly elevating my status in the community. Soon, other members were seeking my advice on how to implement this or that feature. (The same way as I had previously sought advice from the experienced "gurus" who had written technically complex games before.) This connection to less experienced members enables continuity within the community: newcomers are inspired by existing technically ingenious works and can learn from trying to recreate or better them.

This also leads to a level of expectation on the part of the community for one to keep coming up with new ideas. When a member of the English-language *RPG Maker* community presented an early demo of a 3D-enabling script for the tool, the first response they got on *RPG Maker*'s message board was: "Awesome. If anyone would do it, it would be you."[8]

This is not to say the response to experiments with the engine's functionality is universally positive. Consider, for example, the following comment I received for my turn-based tactics game: "Folks, let us not forget that our games should emphasize the literary aspect."[9] To this, another commenter immediately responded: "Why's that?"[16] This goes to show that the "appropriate" uses of URQ were a contested terrain. On the one hand, URQ was positioned as an engine for interactive fiction, leading to the valorization of the literary dimension of games in the community. From this standpoint, games with little writing were bound to be seen as marginal. On the other hand, there was a discourse of URQ as a flexible engine with richer capabilities than were immediately obvious. This discourse became particularly prominent when the advantages of URQ were preached to outsiders, such as users of game engines seen as URQ's rivals. And what better demonstration of the engine's power than proof it can be used to make a technically complex game in a very different genre? I believe a similar tension between the normative discourse and the spirit of experimentation is at play in most game engine-centered communities.

Insisting on using a certain tool regardless of its suitability for the task is also a sign of loyalty to its community. After I released *Xlomidomanad's Experiments*, at least two players questioned the choice of the engine for the game. One of them (who was largely positive about the game itself) wrote: "I would obviously suggest you rewrite your experiment in C, but you would obviously refuse."[10] I had, in fact, learned C before discovering URQ and could probably rewrite my games in it, which would likely be more efficient both in terms of computational resources and my own effort. Yet I chose not to

[8] Comment by meustrus, posted on 21.04.2009 at http://www.rpg-palace.com/categories/game-maker-s-guild/projects/full-3d-rmxp-already-works.

[9] Comment by Belial, posted 11.01.07 at http://urq.borda.ru/?1–0-0–00000200–000–0-0–1191393926. 16 Comment by Goraph, posted 11.01.07 at http://urq.borda.ru/?1–0-0–00000200–000–0-0– 1191393926.

[10] Comment by Goraph, posted 10.01.07 at http://urq.borda.ru/?1–0-0–00000200–000–0-0– 1191393926.

do so, because it seemed that making URQ games was the surest way of staying in the comfort of the familiar community where I had already established myself. Even the comment itself implicitly recognized that my choice of URQ was deliberate ("you would obviously refuse").

Contribution to the tool's development

Probing the boundaries of a game engine's functionality can be the author's way to enrich the tool with capabilities they feel it is missing. In my early URQ days, a member of the "inner circle" called Evgeny released a set of animated text effects for URQ (one, for example, mimicked movie-style credits) that could be reused in any game for the platform. As URQ provided no animation capabilities out of the box, this was a quick way for authors to jazz up their games visually.

Discussions

Several years later, when URQ got a new Windows interpreter that supported graphics, I joined forces with Evgeny to create a game with a custom graphical user interface which did not use any of the interpreter's built-in GUI capabilities. The game, called *Peanut Orchestra*[11], was somewhat clumsy to control—but we did not mind. This was our way of putting pressure on the interpreter's developer (who was initially skeptical) to include skin support, which would make interface customization much easier. Eventually, he did agree to add this feature.

Similarly, above-mentioned scripts enabling *RPG Maker* to display 3D graphics may be a response to the developers' insistence on sticking with retro 2D visuals. Sometimes, such subversive ideas can give birth to a whole new game engine.

As the Rube Goldberg machine demonstrates, being impractical can be expressive. Similarly, probing the boundaries of a game engine's functionality is a valid way of self-expression. In this context, the game maker's functionality becomes the ultimate creative constraint, prompting the developer to look for hitherto unknown ways around the tool's limitations. I found a lot of enjoyment in thinking how to trick URQ into working the way I wanted it to—and not the way it was designed to. I was not the only one: there was, for example, a user who initiated a discussion on the URQ forum about "Various tricks and contrivances with URQ code"[12] where users were invited to share their technically ingenious code.

Results

It is perhaps this process of creative negotiation of a platform's technical constraints that most clearly illustrates why programming can be regarded a form of self-expression (Schulte 2013: 21). This view also problematizes the distinction between the artist (whose work is attributed aesthetic value by virtue of belonging to a recognized art form) and the artisan (whose work is based on the application of skill, with the expressive properties of this work seldom given consideration).

Intrinsically connected to the creative aspect is the ludic dimension. The main reason hobbyists put time and effort into developing games in the first place is that they enjoy playing games. Experimenting with a game engine is itself a form of play, or more precisely meta-play, which builds on

[11] The game was called *Peanut Orchestra* and is available at http://ifwiki.org/index.php/Peanut_Orchestra.
[12] http://urq.borda.ru/?1–0-0–00000384–000–0-0–1316245957.

play's close link to curiosity and exploration.

Ingold and Hallam (2007) also make a case for creativity as "cultural improvisation" rooted in a particular social context. Ingold (2001: 17) also argues against the division between "art" and "technology," which he demonstrates is a modern Western construct, pointing out that both are parts of the same continuum: that of "skilled practice" (ibid.: 20). The practice of subverting game engine conventions, which defies categorization as either purely "artistic" or purely "technological," perfectly illustrates his point.

In her seminal *Cyborg Manifesto* (1991: 164), Donna Haraway wrote that "the boundary is permeable between tool and myth." I understand the "myth" here to refer to the conventions and assumptions surrounding the tool. Taken a step further, this idea can be extended to a McLuhan-esque "*the tool is the myth.*" That is to say, our understanding and usage of a tool is never independent of the beliefs and values surrounding it. But the "myth" of the tool is not static: it evolves overtime thanks to innovation originating from within the community of the tool's users. This article, I believe, instantiates this point.

Conclusion

In an age of technological determinism, we need to remember that while technology has undergone dramatic changes in recent decades, the mechanisms of technological innovation in communities of practice have not in fact changed that much. We thus should not ignore the broader social and historical contexts when discussing digital media communities and innovation therein. But we should also look at their specificity and localize them in narrower contexts. Experiments with game engine conventions can (and should) be linked to such practices as modding and homebrew development; they can be considered through the ideological prism of indie games; they can be linked to wider contemporary practices such as remixing. It is by establishing these connections and learning more about the practices they bring together that we can start to fill in the voids on the vast canvas of contemporary digital media production.

References:

Anderson, C. (2012). *Makers: The New Industrial Revolution.* New York: Crown Business.

Bogost, I. (2008). *Unit Operations : An Approach to Videogame Criticism*. Cambridge, MA: The MIT Press.

Bogost, I., & Montfort, N. (2007). New Media as Material Constraint. An Introduction to Platform Studies. *Electronic Techtonics: Thinking at the Interface. Proceedings of the First International HASTAC Conference*, 176–193.

Ellis, C. (2004). *The ethnographic I: A methodological novel about autoethnography*. New York: Altamira Press.