Casual Creator Cursed Problems or: How I Learned to Start Worrying And Love Designers

Adam Summerville[1], **Ben Samuel**[2], **James Ryan**[3], **Liz England**
[1] California State Polytechnic University, Pomona,
[2] University of New Orleans,
[3] Carleton College,
asummerville@cpp.edu, bsamuel@cs.uno.edu, jryan@carleton.edu, lizengland07@gmail.com

Abstract

A *cursed problem* is a problem that contains a contradictory set of goals. In this discussion paper, we discuss the trials and tribulations behind trying to create a language for social simulation aimed at a casual (or at least non-programmer) audience, while also still appealing to ourselves and other more expert users.

Introduction

In his 2019 GDC talk (Jaffe 2019), Alex Jaffe defines a *cursed problem* as a "problem with some inherent contradiction in goals." We propose that the construction of a casual creator Domain Specific Language (DSL) for complex phenomena is a cursed problem, and invite discussion towards understanding how best to design a DSL that can be useful and usable by designers – programming and non.

In her dissertation (Compton 2019), Kate Compton discusses Casual Creator (CC) programming languages and discusses three components that CC programming languages might (should?) have:

- **Scaffolded** – The output of the program should be visible during editing, to speed the 'grok' loop
- **Unfoldable** – The language should have a compact form that beginners can use, with advanced features available for power users
- **Little** – The language has a compact vocabulary useful for a small domain (as opposed to a general purpose programming language like C, Javascript, or Python)

The authors of this paper have been developing a CC-like DSL capable of producing worlds for social simulation title *Kismet* (Summerville and Samuel 2020). Previous attempts at social simulations have either relied on bespoke code in a general purpose programming language (Ryan et al. 2015; Adams and Adams 2006) or DSL's that are perhaps only arguably human readable (the XML representation of Comme il Faut (McCoy et al. 2014)). Taking inspiration from Inform 7 (Nelson 2006) the language in question uses natural language like representation for constraints:

```
if Drinker is drunk and
  Drinker and Target don't like each other
```

or

```
if Gossiper heard GossipedAboutAction and
  Gossipee didn't do GossipedAboutAction
```

With the intention being the construction of a DSL that hits the following design goals:

1. **Free Text** – Can be edited in anything that can edit text (Word, Notepad, Atom, GMail, IRC, etc.) and does not require a special editor
2. **Natural Language** – The act of writing rules is closer to writing prose than a standard programming language
3. **Simple Simulation** – Instead of simulating the complete richness of human (or inhuman) society, the language is designed for "apophenia hacking" (Ryan 2018) wherein humans use their innate pattern matching skills to read more into it than there might be

These designs seemingly hit on some of the points set forth by Compton

- Free Text → Little
- Natural Language → Unfoldable
- Simple Simulation → Scaffolded

but in practice, these designs do not necessarily hold up, or at least not for all circumstances and users. The free text nature does lend itself more to being a little language that could exist inside of a more general purpose system, but it is not enough for it to be little. The natural language aspect enhances readability, but it hurts writability (E.g., is "doesn't like" equivalent to "dislikes"?) in that natural language is not a formal language (Graham Nelson notes in his 2019 Narrascope talk (Nelson) "that the benefits [of natural language] are double-edged. But they are real."). Finally, while the simple simulation is closer to something that can be updated and shown in real time to a user, in practice even the simplest of simulations takes at least a few seconds to run for long enough to show interesting emergent properties, which is much slower than the fraction of a second for something like procedural text generated by Tracery (Compton, Kybartas, and Mateas 2015).

In looking to improve this language, we sought the feedback of a professional game designer. The designer notes that "a good tool removes cognitive overhead from the designer and puts that information directly in the tool, so that the designer can keep more of their focus on high level design goals and not get bogged down with lower level concerns like syntax or data setup, or even just remember what the system can or cannot do." In discussing syntax, the designer states "My philosophy is that I don't ever want to have to worry about syntax. If I am spending time on syntax, then I'm not spending time on design. If you think of it like Maslow's hierarchy of needs, syntax is the 'am I being fed and clothed?' layer, and I want to work on the 'self-actualization' layer where all the interesting design is happening." The designer has expressed an interest in the language being entirely behind the scenes, with interaction being handled via its own one-off graphical editor (a la the non-public editor for Comme il Faut). However, this goes in contrast with the other goals – Free text explicitly and Natural Language implicitly (as there is no reason for natural language when the language is hidden from the designer). This also goes in contrast to the goals of other designers such as Reed (Reed 2020) "Other tools were also unsuitable for various reasons, such as requiring IDEs rather than support for text files." or Compton "the structure editor was a perversion of what casual creators ought to be" (Compton 2019). However, that is not to say that the idea of an editor is entirely opposed, as Compton does go on to say "...but the constraint of making JSON errors impossible was a necessary safety support for a large percentage of my users." I.e., the limitations of a tool can be helpful if they also introduce useful safety. Similarly, on the topic of simulation richness, an expert in Social Simulation expressed that the expressiveness of the simulation was not rich enough and needed higher order modeling (e.g., agents being able to reason over memories and form meta-memories where they reminisce about those memories). The professional developer has noted that the currently planned features are already too much: "I can tell that once locations and more complex characters (with roles, age, and other arbitrary defined information) are ready to be implemented, I'm probably going to be lost." But in coming from a world of bespoke tooling for AAA games they note, "I find that [the language] covers half my needs, and the other half are a mix of missing functionality (now that I have a specific use case) or extra functionality I don't need (because I can't see what problems it solves)."

This brings us back to the idea of a cursed problem – a problem (in this case design goals) that have opposing goals. Different users (the language authors, the non-programming game designer, and a social simulation expert) each have different features that are desirable (perhaps even necessary) for them.

1. **Free Text** – Important for the authors and expert. Unimportant for the designer.
2. **Natural Language** – Important for the author. Useful for the designer for readability, but a hindrance for writability. Unimportant for the expert.
3. **Simple Simulation** – In some cases too complex for the designer and in others not rich enough. Not rich enough for the expert.

Perhaps there is some set of tooling and *foldability* that makes this Gordian knot cuttable, such that it is usable and interesting to both a non-(to-lite)-programmer and a social simulation expert, and that is what we would like to discuss.

The Discussion

We now note questions that we the authors have that we think can lead to good discussion:

- Is this a cursed problem?
- Is there a way to please all (most?) users across a wide range of programming comfort? (text editor for programmers, Scratch-like GUI for beginners, drop-down boxes for intermediate?)
- Is there a way to please all (most?) users across a wide range of simulation desires?
- What human-in-the-loop accessible feedback can be provided in real(ish) time for something that could take 5-100 seconds to generate?

References

Adams, T., and Adams, Z. 2006. Dwarf fortress. *Game [Windows, Mac, Linux], Bay* 12.

Compton, K.; Kybartas, B.; and Mateas, M. 2015. Tracery: an author-focused generative text tool. In *International Conference on Interactive Digital Storytelling*, 154–161. Springer.

Compton, K. 2019. *Casual creators: Defining a genre of autotelic creativity support systems*. University of California, Santa Cruz.

Jaffe, A. 2019. Cursed problems in game design.

McCoy, J.; Treanor, M.; Samuel, B.; Reed, A. A.; Mateas, M.; and Wardrip-Fruin, N. 2014. Social story worlds with comme il faut. *IEEE Transactions on Computational intelligence and AI in Games* 6(2):97–112.

Nelson, G. Opening inform.

Nelson, G. 2006. Natural language, semantic analysis, and interactive fiction. *IF Theory Reader* 141:99–104.

Reed, A. A. 2020. A minimal syntax for quantum text.

Ryan, J. O.; Summerville, A.; Mateas, M.; and Wardrip-Fruin, N. 2015. Toward characters who observe, tell, misremember, and lie. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Ryan, J. 2018. *Curating simulated storyworlds*. Ph.D. Dissertation, UC Santa Cruz.

Summerville, A., and Samuel, B. 2020. Kismet: a small social simulation language. In *Summerville, A., & Samuel, B.(2020, September). Kismet: a Small Social Simulation Language. In 2020 International Conference on Computational Creativity (ICCC).(Casual Creator Workshop). ACC*.