When to Not Trust Dynamic Text

Aaron A. Reed

Oct 24, 2019 · 6 min read

My upcoming novel *Subcutanean* is written such that each time it's regenerated for a new reader, hundreds of moments of textual variation get collapsed down into a single possibility. If you've written dynamic text before (for games, maybe) you're probably aware that this isn't as easy to get right as it might first seem. Errors both tiny and tremendous can easily appear. Some of the more insignificant ones around punctuation or spacing show up in the output like typographical mistakes:

> *great. I'll see you at the store.*
> *Have fun!. See you later.*
> *She loved chocolateand to swim in the sea.*

In each of these cases, the author probably forgot a detail of context around the dynamic text they inserted: that it needed to be upper cased because it would be used at the start of a sentence, or that it should include or omit punctuation or a space. These are common gotchas when you're working with two pieces of text that will end up next to each other but are written in different places.

Sometimes a join is less about a mechanical problem and more about natural phrasing. You might have written a clever expansion that can say "Yes" in a number of different ways: "Sure," "OK," "Sounds great," and so on. But then one of your Yes expansions might show up somewhere it sounds awkward:

> *"Are you Spartacus?"*
> *"Sounds great, I am."*

This kind of thing can lead to all kinds of awkward output texts, and a good chunk of the somewhat stilted, unnatural tone we associate with chatbots comes from these kinds of awkward insertions:

> *I gave the book to he.*
> *Hi, Nicole but you can call me Nikki lol 😛 , it's great to meet you!*
> *She loved chocolate bar and swim in the sea.*

These mistakes are even more awkward, by the way, once they're rendered in a fancy way and inserted into a context that looks like a human created it:

She loved chocolate bar.

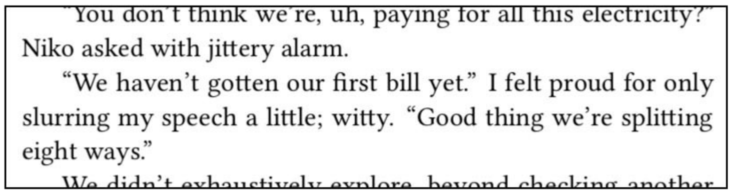
Background courtesy Pam de Butler from Pixabay

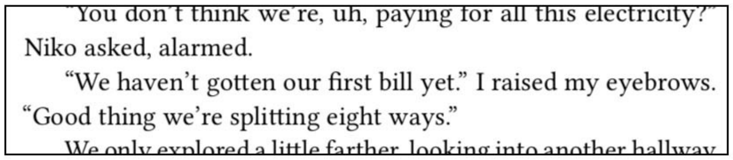
Sometimes dynamic text authoring mistakes can be more serious, like when a vital control symbol is mistyped:

> *"No," you say firmly, "I could never love you[Yes, kiss me, kiss me now!" you cry, throwing your arms around {love_interest}'s neck.]]]*

In a game, a Twitter bot, a web interface, or a NaNoGenMo project, these kinds of mistakes are generally forgivable by readers (if not their authors) as an interesting glimpse behind the curtain. This is in part because some of the charm of procedural text lies in the possibility of the unexpected. We're intrigued when we see oopsies like this, because it reminds us of the generative nature of the system we're interacting with.

Subcutanean, however, has very different goals. The output is meant to be a print novel — generated, printed, and shipped in an automatic process with no human review — that will not only remain on your shelf for (possibly) decades, but should also be as immersive and compelling as any other book. The variable possibilities are meant to interest you in reading it in the first place, but not be a distraction once you've actually settled down with a copy. In some ways this project is the precise *opposite* of most generative text works: I want to guarantee that nothing too unexpected can happen, maintaining complete authorial control over every possible output.


"You don't think we're, uh, paying for all this electricity?" Niko asked with jittery alarm.
"We haven't gotten our first bill yet." I felt proud for only slurring my speech a little; witty. "Good thing we're splitting eight ways."
We didn't exhaustively explore, beyond checking another


"You don't think we're, uh, paying for all this electricity?" Niko asked, alarmed.
"We haven't gotten our first bill yet." I raised my eyebrows. "Good thing we're splitting eight ways."
We only explored a little farther, looking into another hallway

Excerpts from two typeset renders of Subcutanean: in one, the narrator's drunk.

So I decided to take the nuclear option. For each piece of text that could vary, I would manually confirm it. In context. And each time any of the variants or any of that context changed, I would manually confirm it again.

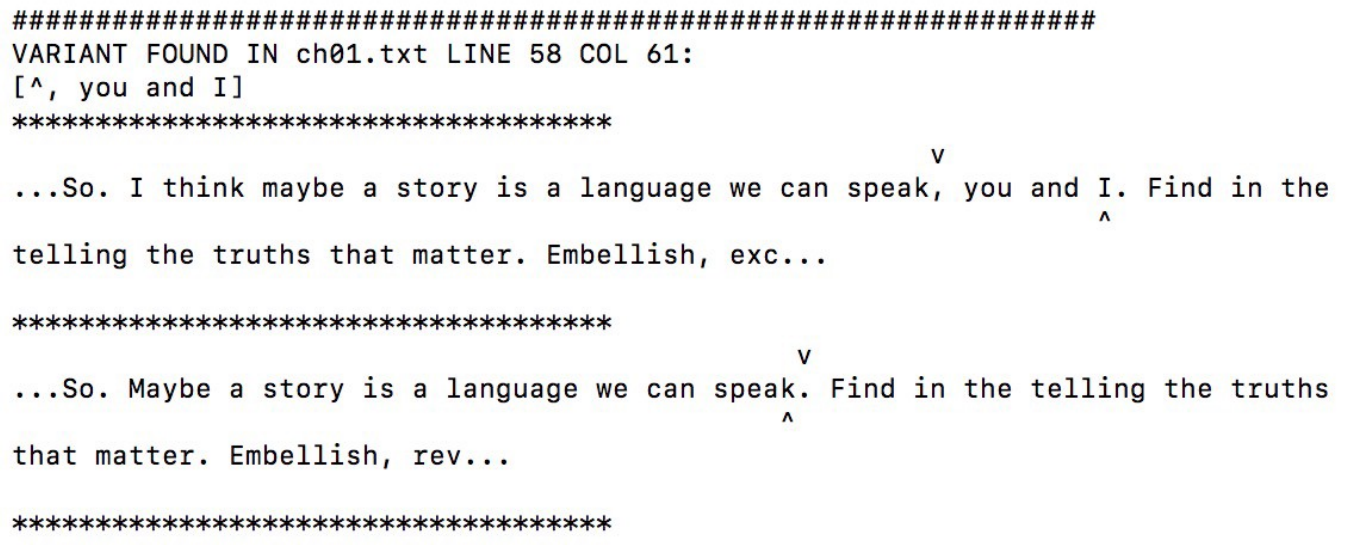
• • •

To understand why this is even possible, I should explain that *Subcutanean* only rarely contains nested expansions (I wrote about why here), as are common when working with something like Tracery. Nor is the text unbounded in the way a procedural text project that uses Wordnet or a Wikipedia corpus might be: each moment of variation is deliberately and thoroughly curated. So this wasn't as impossible a task as it might sound. Showing the author a range of sample outputs, as many and as quickly as possible, is of course a useful approach adopted by other procedural text authoring systems like Tracery or Character Engine: the difference here was that I wanted to not only show *all* the variations, but require that each one be manually validated.

The key challenge was to keep thisfrom being a slog, a chore so laborious and annoying that I'd stop doing it, and risk immersion-breaking mistakes slipping through.

To achieve this, I developed a module in my rendering pipeline called Confirm. Each time I export the text, this module shows a couple possible ways bits of text can be rendered, like an aide discretely handing over a couple documents to the boss for a signature. The boss is too busy to sign the whole stack, but if the aide gets a few signatures each time there's a free moment, eventually everything will be processed.

The Confirm module intervenes before the formatting and output-generation part of the pipeline. First, the whole master text is scanned and each bit of variable text is given a unique key, made up from the line before, the variants themselves, and the line after. The module then renders the "before" and "after" text (including rendering any variants within those, if necessary), glues this together with each way the current variant can be rendered, and shows these outputs to me. Little carets are added between the lines to show exactly where the variant starts and ends. I can take a quick look at each version in a basic double-spaced ASCII rendering (with all formatting and other control characters removed), reading the output as normal prose and only paying attention to the carets if something looks wrong.

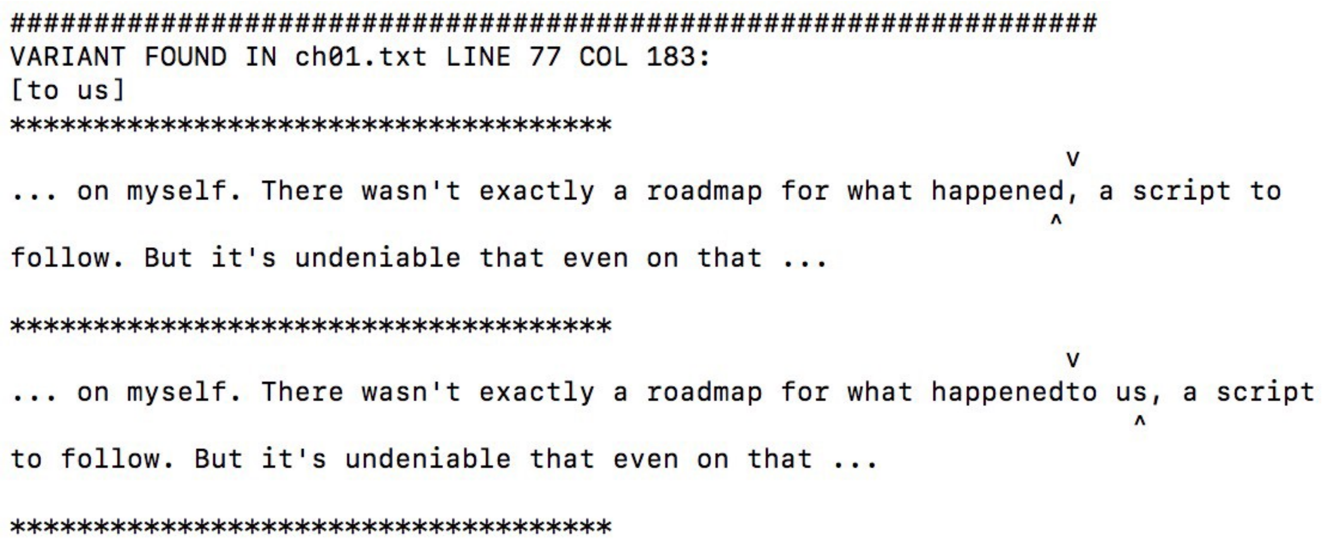


```
##################################################################
VARIANT FOUND IN ch01.txt LINE 58 COL 61:
[^, you and I]
************************************
                                                       v
...So. I think maybe a story is a language we can speak, you and I. Find in the
                                                                 ^
telling the truths that matter. Embellish, exc...

************************************
                                               v
...So. Maybe a story is a language we can speak. Find in the telling the truths
                                              ^
that matter. Embellish, rev...

************************************
```

Example of a very simple confirm with an optional phrase "you and I." The text before and after is also slightly different; the system picks one way to render before and after context for each individual confirm (though these can be regenerated with a keypress to see other possibilities).

If everything looks okay, a keypress confirms that set of variants in that particular context — and as long as neither change, the system won't bother me about it again. But if I edit the surrounding text, the variants, or add or remove options, the key has changed: and therefore that chunk goes back into the review queue. When performing final renders for the eventual book deliverables, the system won't move on to PDF rendering until each unique key has been confirmed — but I'll have been working on it piecemeal as I go, one change at a time, rather than saving that huge stack of documents for signatures on the very last day.

The bit about the before and after context is key: sometimes a variant itself is fine, but sounds awkward given what came earlier. A word repetition, for instance, is hard to catch if it's split between an introduction and the start of the third option in a set of multi-paragraph variants. This kind of thing is easy to catch with Confirm, as well as all the simpler errors around spacing or punctuation, which also stand out clearly.



```
##################################################################
VARIANT FOUND IN ch01.txt LINE 77 COL 183:
[to us]
************************************
                                                                v
... on myself. There wasn't exactly a roadmap for what happened, a script to
                                                               ^
follow. But it's undeniable that even on that ...

************************************
                                                                v
... on myself. There wasn't exactly a roadmap for what happenedto us, a script
                                                                     ^
to follow. But it's undeniable that even on that ...

************************************
```

Catching optional text that needs a space before it.

Of course, the next step is rendering a print-ready PDF from a procedurally assembled LaTeX file, and making sure that nothing's gone wrong in that process — no overfull boxes have spilled text into margins, no misplaced control characters have erased or distorted huge chunks of text. But that's a whole new set of problems, and I'll save those for a future post.

> **Subcutanean** *is an upcoming horror novel that changes with each new copy. Find out how to get your own unique copy. You can also follow the project on Twitter, Facebook, or Goodreads, or check out more design posts.*

Typography Procedural Text Text Generation