>Remember

By Hugo Labrande

Issue #2 : Text compression in Infocom games

After last month's article on the history of retro text adventures, let's switch it up and talk about a more technical topic: the way the text is compressed in Infocom's Z-Machine. It's actually a fascinating topic, and one I definitely became well-acquainted with when writing "Tristam Island" and its translation to French.

Infocom's scheme

Recall that the Z-Machine is a virtual machine, that is, nothing more but a specification, a set of standards on how code written for it should behave on a regular computer. It was a great idea by Infocom: you get everyone together, say "this is the low-level instructions and this is the behavior they should have", then everyone writes some program (an interpreter) that implements the behavior on whatever computer they want. Once the interpreters are written, tell your writers to write just one version of their game, in Z-machine language, and all these computers can now play the game. It's a great idea, definitely reused by other makers of text adventures, but also programming at large - see for instance Java's success and portability.

As part of the standard, Infocom came up with a scheme to encode text more efficiently. You could encode your text using the character set: the computer has a table to make bytes correspond to letters; sometimes these encodings are different from one computer to another, sometimes they can be redefined, but the point is, if you say PRINT"NEWSLETTER", the string uses 10 bytes of memory. But Infocom came up with a simple encoding scheme that is more efficient in practice.

Their scheme is based on "units" of 5 bits, encoding 32 different values. Three units are then packed into 2 bytes, with one unused bit at the end, like this:

```
11111222   2233333X
```

In the best case scenario, 3 letters are packed into 2 bytes; but not every letter fits, of course, and some characters will take up 2 or even 4 units. The general rule of thumb is that the encoding allows Infocom to pack 25-30% more letters (as in, a text of 100k letters in English will only take up 75k bytes); not bad for such a simple scheme, and definitely helpful when you want to write a computer novel.

If you want the technical details, here is how everything is decoded:

- if the value is between 6 and 31, it's a lowercase letter, a to z;
- if it's 0, it's a space;
- if it's 1, 2, or 3, it's an abbreviation (we'll talk about that later);
- if it's a 4, the next unit is a number between 6 and 31 corresponding to a letter in the uppercase alphabet;
- if it's a 5, look at the next unit; if it's not a 6, it's one of the symbols

  ^0123456789.,!?_#'"/\-:()

  with ^ representing a new line and ~ the quotes;
- if it's a 5, then a 6, the next 2 units (10 bits) hold an 8-bit value corresponding to a certain table. This table matches ASCII for numbers up to 128; for values higher than 128, it's actually unclear. For Z-machine version 3, Infocom never

looked at values greater than 128, thus using a 7-bit mask; but later (community-made) versions of the standard defined a table that held, most notably, European accents. Interpreters developed after Infocom's usually support these accented characters, but none of Infocom's do.

Finally, the Z-machine can hold up to 96 abbreviations, that is to say, 96 strings of any length that can be summoned using just 2 units. This is brilliant: for instance, " the " (5 units) can be abbreviated, thus saving 3 units (2 whole bytes) every time it appears. Over time, these savings add up, and amounted to an extra 10% reduction in Infocom's games, which is definitely appreciated.

Can this be improved?

At this point, the Z-Machine standard is fixed, so we're stuck with that encoding scheme, and all its inefficiencies (like wasting one bit every 2 bytes). It's actually interesting to note that at the time, there were other, more efficient schemes. For instance, the British company Level 9, who had to pack their games on tape instead of disks like Infocom’s because the European market was very much tape-driven, had their own program to compress text, and it was by their account much more efficient than Infocom’s. There’s not many details on their method (if you have some, let me know!), except what they said in an interview, in which they claim 50% savings using a scheme that, to me, just sounds like Infocom’s abbreviations. (I am a bit skeptical that it’s the only thing they did, as abbreviations have diminishing returns; though I guess if you define enough, way more than Infocom’s 96, you might have a good chance at compressing text quite nicely.) My source is:
https://www.filfre.net/2012/10/level-9/

Another possibility could be to use an encoding scheme like Huffman's encoding scheme, which determines (using a binary tree) a set of encodings for the given symbols that depends on their frequency, in such a way that decoding is never ambiguous and can be done reading the text bit by bit, and is optimal size-wise. The efficiency can be very large, although most figures I could find quote around 35-45% savings - significantly better than Infocom's scheme. I don't really know why they didn't use that method, invented in 1952; but perhaps decompression was too expensive. (Infocom's encoding scheme was invented with version 1 of the Z-Machine, in 1978; perhaps the 8-bit micros of the time couldn't do this very efficiently, or perhaps it was wasting too much space in RAM. Anybody with more 8-bit programming experience than me wants to chime in?)

However, there are a few opportunities in the realm of abbreviations. From what I can see, Infocom calculated the abbreviations they needed to declare using the following procedure: break down the text into words, count each word's frequency, multiply by the number of units it takes (minus 2, to get the savings of each abbreviation call), then sort and take the top 96 abbreviations. This is certainly a nice technique, as it doesn't take too long to compute; but this technique is missing all the abbreviations that start in the middle of words. In "Tristam Island", the following abbreviations were quite efficient: "ing ", "n't ", " th", "ould ", "ight", "ed " "ough", "tion", etc. (and, of course, full words like " the ", "You ", " you", " and ", " to ", etc.).

Inform 6’s compiler has a switch, "-u", which goes over the game text to determine the best abbreviations for your game. I've always liked using that switch and squishing my text, but I've never been sure it was computing optimal abbreviations. So, as I was making "Tristam Island", I created my own script, with the goal of being thorough but still fast. It's made in Python, and it checks every substring up to a certain length; pruning strategies and score updates ensure you find at each step the

abbreviation that saves the most space, greedy-algorithm style. You can find this script on my Github:
https://github.com/hlabrand/retro-scripts

So, how much space could Infocom have saved with better abbreviations? (This is all hypothetical; 1980s computers wouldn't be able to run my Python script in a reasonable time.) It turns out, a nice chunk. I ran some tests with Henrik Åsman on the intfiction.org forum, on a thread about finding optimal abbreviations:
https://intfiction.org/t/customized-infocom-game-wont-fit-on-version-3-z-machine/48608

Here are some figures to realize how much space is saved by my script:

Zork 2: 89,400 bytes	(with Infocom’s abbreviations: 92,192 bytes)
Planetfall: 105,016 bytes	(with Infocom’s abbreviations: 108,594 bytes)
Plundered Hearts: 125,274 bytes	(Infocom’s: 128,830 bytes)
Trinity: 257,068 bytes	(Infocom’s: 261,484 bytes)

So it turns out that using better abbreviations can free around 2.5kb of space in Infocom's story files! What they could have done with that space is left to our imagination; more funny responses, another puzzle, a fun feature?

But this is more than daydreaming about the "could-have-beens": this has very tangible applications if you are making a retro text adventure, like I did with "Tristam Island". My savings totalled 19 kb (compared with Inform's 15.7kb), and I used every single byte of that space to pack in more features, more vocabulary, and more responses. I am quite proud of how large "Tristam Island" is, and although other factors definitely played a role (like PunyInform's compactness and Fredrik Ramsberg's advice on optimal code patterns - that could be a whole article in itself), aggressive abbreviation-finding freed up space at no cost, which allowed more content to fit. I definitely recommend using my script if your own game is getting close to the 128kb limit!

Finally, let me close this discussion with mentioning that, after the original discussion spurred by my script, Matthew Russotto implemented his own algorithm in C++ using suffix trees, a very appropriate data structure for this kind of job. My algorithm is, in comparison, rather crude, and his script runs in a few seconds instead of the minute or so mine takes; and he was able to find more optimizations and bring “Zork 2” down to 89,368 bytes! You can see the discussion on intfiction.org:
https://intfiction.org/t/highly-optimized-abbreviations-computed-efficiently/48753
And the good news is, it now benefits users of ZILF, as Jesse McGrew implemented this algorithm in their own tool. Hooray for better text compression in text adventures!