>Remember

By Hugo Labrande

Issue #18 : The new Inform 7

This month's huge news in interactive fiction is without a doubt the new version of Inform 7 that has been released by Graham Nelson, years after the previous version, with a lot of new tools and features, and now open source! This is really great news, and got a lot of people excited, with reason!

https://intfiction.org/t/inform-7-v10-1-0-is-now-open-source/55674

... But hang on. What is Inform 7? Why are we talking about it? What does this mean for retro text adventures? Let's back up, and explain!

This history of Inform

If you want to read about the history of Inform, there are a few resources, one of which being... Jimmy Maher's articles (bet you didn't see that one coming):

https://www.filfre.net/2019/10/new-tricks-for-an-old-z-machine-part-1-digging-the-trenches

https://www.filfre.net/2019/11/new-tricks-for-an-old-z-machine-part-2-hacking-deeper-or-follies-of-graham-nelsons-youth/

https://www.filfre.net/2019/11/new-tricks-for-an-old-z-machine-part-3-a-renaissance-is-nigh/

Here is the very short story: Graham Nelson, while a PhD student at Oxford, was an Acorn Archimedes user, and bought the 1992 Lost Treasures of Infocom for MS-DOS and ran the game files on the Archimedes using the InfoTaskForce interpreter; portable third-party interpreters had just started appearing and being disseminated online. While corresponding with Mark Howell, who wrote the "txd" disassembler, he set out to create a program that could create a story file that would be playable on the ITF interpreter: after much effort, Nelson has a Z-Machine assembler, which then grew into Inform, a programming language with a compiler producing Z-Machine files. The rec.arts.int-fiction newsgroup was a good place to chat and gather feedback; as he developed subsequent versions of Inform, Nelson also wrote his game "Curses", as a proof of concept and to entertain his friends. Curses was released there in 1993, which created a lot of interest; then came IFcomp in 1995, and the Inform Designer's Manual, and Inform 6 in 1996: the Inform 6 development scene became very vibrant for the better part of 15 years.

I6 as a language is object-oriented, with code looking like this:

```
   Object jukebox "jukebox" InPub
      with name 'jukebox' 'machine' 'slot' 'bal-ami',
      description "The jukebox is an old 1950s machine in a forgotten
corner in the pub.",
      before [;
            Plug: if (self has on) {"It is already plugged in!"; }
                  else { give self on; "As you plug the jukebox in the
outlet, it lights up!"; }
],
has scenery concealed;
```

But in 2007, Graham Nelson shocked the world with the release of Inform 7. Every update that came before had been an incremental tweak of the syntax; but here, the language's syntax had a major change towards English "natural language" syntax. This means the code became like this:

```
   The jukebox is in the Laboratory. Understand "machine", "slot" and
"bal-ami" as the jukebox. The description of the jukebox is "The jukebox
is an old 1950s machine in a forgotten corner in the pub.".

   Before plugging the jukebox when the jukebox is switched on:
      say "It is already plugged in!".

   Before plugging the jukebox:
      now the jukebox is switched on;
      say "As you plug the jukebox in the outlet, it lights up!".
```

Inform 7 had a massive impact: very quickly, most authors switched from Inform 6 to Inform 7, and Inform 6 was virtually extinct -- well, in the anglophone world, at least; other communities somewhat kept using Inform 6 (especially after it was opensourced) out of habit or because the natural language syntax was built for English but not so much for other languages. There are thousands of I7 games in English: a lot of newcomers were attracted by the natural language syntax, which made authoring very approachable and powerful; but I7 also had very powerful new features for advanced authors to leverage.

While I6 was object-oriented, I7 was a rules-based programming language. This meant behaviour could be triggered by rules that can be applied with a lot of flexibility; customizing the game's behaviour was a breeze, as you could insert as many hooks as wanted and control precisely when they would trigger, while doing the same in Inform 6 would have required rigour, discipline, and a very good understanding of the library. But you could also build complex systems, networks of rules triggering cascading behaviour, very easily; in my opinion, it is very unlikely that games as complex and systematic as Andrew Plotkin’s *Hadean Lands* and Emily Short’s *Counterfeit Monkey* could have been programmed in Inform 6! Inform 7 can run through sequences of rules, or rulebooks, and apply changes, or branch; this is very easy for the author, with a lot of flexibility, but also means the engine needs to check more possibilities and had more bifurcations.

The old Inform 7 and retro games

Inform 7 has never been used to write text adventures for retro platforms. You will however find games with a retro flavor and made with Inform 7: a couple of examples are Jim Munroe's *Guilded Youth*, and its BBS influences; Daniel Seltzer's *Scroll Thief*, which is heaven for *Zork* fans; and Jon Ingold's *Make It Good* takes Infocom's *The Witness* and runs much, much further with amazing results. You can make your own text adventure with Inform 7 and use CSS styling to give it a retro look, and even make a graphical adventure using Vorple (see Issue #8 for a tutorial!). But ultimately, if you want to make a game that runs on a C64 or an Amiga, you will not do it using Inform 7.

The reason is very simple. Inform 7 does not compile to Z-Machine directly; it generates I6 code, then the I6 is compiled through the regular compiler. This means it has to generate I6 code and make this work using generic and complex constructs: this is very tricky! As a result, if you look at the Inform 6 code generated by Inform 7, you will find a maze with many generic constructions, lots of jumps and function calls, and complex problems sometimes resolved through code repetition. Inform 7 are very far from optimized from a low-level code standpoint: they are much larger than I6 games, and they are very slow (probably because of all these function calls). In fact, Z-Machine games written with Inform 7 are very frequently z8 games (just because of their size, 256-512kb) and are frequently unplayable on a retro machine, even one supporting z8 games (like Ozmoo on the C64). In fact, it's not just the speed: the numerous nested function calls means the call stack needs to be at least 16KB large, which on an 8-bit machine is just not feasible given the available RAM; as a result, an I7 game is likely to run out of memory at some point when playing on an 8-bit computer. And there is not much to do about it: the code that is generated is tortuous and inefficient from a pure performance perspective; optimizing it would most probably required a bunch of profound changes, and without any obvious benefits since very few players in 2007-2017 played on anything else than a modern computing platform with gigabytes of RAM and tons of processor power to throw at the text game.

To give you an idea of the relative efficiencies (and I've already mentioned this in a previous issue!), Ryan Veeder's *Craverly Heights* is a 258KB game file compiled with Inform 7 version 6G60, and, Jason Compton tells me, with each turn taking dozens of seconds (and frequently over 1 minute) on a C64 *with REU* (meaning it would be much worse using disk!). However, a reimplementation with PunyInform gives a 49KB game that's also much faster (a couple of seconds on a C64 with disk between each command), and 48KB with ZIL with similar speed. That being said, it goes without saying that, on a recent computer, you will not notice the difference between these versions!

The new Inform 7

On April 28th, the new version of Inform 7 (the first one in 7 years) was announced. Most of the changes consisted in the fact that it was open source for the very first time, and a brand new approach to the toolchain: Inform 7 code is now compiled into an intermediate representation, "inter", and various tools allow you to transform inter code into other types of code, like Inform 6, or C, which forms the final code to be compiled. This is a very interesting approach, which was inspired by LLVM, which similarly uses an intermediate representation on which optimizations are performed, before compiling to machine code. The thought of being able to compile I7 code directly to JavaScript, Unity, or pure C or even, who knows, 68k assembly is a tantalizing one indeed.

But on its face, the new version of Inform essentially proposes to have another layer of translation between I7 code and Z-Machine code. And usually, the more abstractions, genericity, and layers of translation, the worse the performance is - as in, your productivity as a programmer is higher, but the end code runs slower. And indeed, the preliminary results tend do show that the new Inform 7 version only makes matters worse in terms of code efficiency: while a simple "Hello World!" game compiled to Z-Machine took over 400KB on version 6L38 of I7, and later went over 580KB in 6M62, the new version of I7 produces code that is 792KB large. For those who were thinking of using C instead, this generates 86k lines of code and a 1.2MB program (and that's in basic mode - including the parser and world model gives 200k lines and 3.6MB of code!).

I don’t think I need to spell this out: the new Inform 7 cannot be used to write games for old computers, and generates larger and slower code than before. Some might say it’s a bad thing; I wouldn’t. It achieves its vision of opening Inform for integration in many more settings, tools, and eventually games, and very few modern game development engines will balk at adding one more “external library” that weights 3MB. The rule-based system is as powerful as ever, and the syntax is easy to understand; this is all part of Graham Nelson’s vision, and it works! It all goes with a general pattern in software development these days of leveraging extraordinary clock speeds to improve programmers' productivity and allow them to manipulate powerful abstractions at an industrial scale, instead of the artisanal approach of hand-crafting assembly. I know some decry it, and it is not for everyone; however, I have no doubt that we will see some interesting uses of the system in the years to come, and the code generation inefficiencies will likely not be noticed.

That being said... call me an optimist (and maybe this is completely unrealistic!). I like optimization problems, I like elegant constructs, I like fast code. So I can't help but wonder: how optimized could we make code generated by I7? Sure, there are a few translation layers, but GCC and LLVM both have translation layers too; does this mean they generate inefficient code in the end? (I haven't followed the story of compilers throughout the ages; does anyone know if modern-day C compilers are that much worse at generating, say, 68k assembly than the compilers of yore? Are they maybe even better) LLVM is able to perform a lot of optimizations on a LLVM-IR representation; how optimized is Inform 7's inter, and could we create these kinds of optimization strategies? Does I7 generate inefficient code because of a fundamental limitation, or is it because generating compact code just hasn't been a priority, since nobody will notice any slowdown on a modern platform?

This is an interesting question, but most importantly: Inform 7 is now open source, so we can look into it! Right now, the new Inform 7 version is not relevant for retro text adventures. However, I doubt Graham Nelson (& team) has made it a priority during the last 7 (or 20) years to spend extraordinary amounts of time optimizing the generated code: getting the project to work is already ambitious enough as it is without having to worry about invisible inefficiencies! But now that Inform 7 is open source, some people could look into it, and might even expand and improve the toolchain to generate better code. So who knows, maybe one day Inform 7 will be relevant for retro text adventures! But the first step would have to be having brave adventurers step forward and contibute improvements to its tools. Could it be you?