Interview with Sean Ellis re: Graphic Adventure Creator

Aycock, John

Interview with Sean Ellis re: *Graphic Adventure Creator*

John Aycock
Department of Computer Science
University of Calgary
2500 University Drive N.W.
Calgary, AB, Canada T2N 1N4
aycock@ucalgary.ca

TR 2016-1086-05, August 2016

Preamble

This is an interview with Sean Ellis, author of *Graphic Adventure Creator* (1985), conducted via email on August 16–18, 2016. Interview questions appear in *italics*.

This work received ethics approval from the University of Calgary's Conjoint Faculties Research Ethics Board, file REB16-1235. Both interviewer and interviewee have agreed to release this interview under a Creative Commons Attribution-ShareAlike 3.0 Unported License.[1]

Interview

What was your first computer?

My first computer was a Science of Cambridge MK14. Our school had two which were not working, and I was given them to take home and play with. By swapping chips I was able to make one work. This was a single-board computer with a hex keypad and an 8-digit LED display, with the extra RAM expansion so that it had a full 384 bytes of memory. I gave it back to the school when I got my second computer, a ZX81, which was the first I actually saved up for and bought.

What was your education and background in programming?

[1]https://creativecommons.org/licenses/by-sa/3.0/

I had been interested in maths and science from an early age, and was a member of the school computer club from age 14. I took ‘O’ level computer science at age 16, did a standard Maths-Physics-Chemistry combination at ‘A’ level and then studied Computer Science and Cybernetics at Reading University.

To confirm, you wrote the original version of GAC yourself, which was for the Amstrad CPC, correct?

Correct.

When did you first start working on GAC? How old were you at the time?

I started to write an adventure game system (then called ‘ADVAL’) in my first year at University, which was 1984. I was 18 at the time. By coincidence, our University computer club invited the guys from Incentive Software to speak and I ended up having a chat with the founder, Ian Andrew, who asked me to bring it in and show it to them.

How long did it take to develop GAC?

About a year, including the porting work.

What was your development environment like? What computer, language, tools, and editor did you use?

All done with a hex editor, believe it or not. I had learned Z80 machine code and written a few games in hex on my ZX81, for my own amusement, and had got to the point where I didn’t actually have to write down the instruction mnemonics.

I managed to graduate this “style” of coding to the development of GAC, as the CPC464 was also Z80 based.

Doing this in hex was (in hindsight) a terrible idea. Here’s why.

In assembler you can mess around with the size of subroutines. I couldn’t. If I wanted to extend a routine I had two choices – either move it to the end of the code (freeing up the previously used bytes) or tack a jump onto the end and put the rest of the routine at the end of the code.

Changes to the code usually involved either strategic use of NOPs, rewriting or, again, tacking a patch routine onto the end of the code. This involved taking 3 bytes out of the main code, tacking in a jump to the end, where those 3 bytes were then added together with the patch, and a jump back. Example:

```
...
0440: 2A 03 01 ld hl, ($103) ; oops! forgot to ld a,(hl)
0443: 11 00 00 ld de,0
...
```

becomes:

```
...
0440: C3 A0 41 jmp Patch41A0
0443: 11 00 00 ld de,0
...

...
41A0: 2A 03 01 ld hl,($103)
41A3: 7E  ld a,(hl)
41A4: C3 43 04 jmp $0443
...
```

(30 years later and I can still do this from memory. My memory therefore needs a good clear out.)

I also had to manually recalculate the offsets for relative jump instructions. After a couple of months, you get very, very adept at counting backwards in hex.

Worst of all, when the time came for others to port the thing onto the Spectrum, Commodore 64 and BBC, I then had to write a disassembler (in hex) and spent the best part of a month fixing up the patches in a text editor and commenting the code printout in pencil. And there was a bug in the disassembler that meant that IX and IY instructions weren't handled properly.

What was the inspiration for GAC? Any additional inspirations for its design and user interface?

The original inspiration was an adventure game system for the ZX81 that was in a book. I'm afraid I cannot remember the name of the book, but the basic idea of representing locations, objects, counters and flags came from there. It also used "conditions" which were effectively small interpreted programs encoded as strings. *[Sean later recalled that it was the* ZX81 Pocket Book.*]*

All of this evolved massively under the guidance of Ian at Incentive, who understood the idea of "user friendly" a lot better than I did. It was his idea to add the graphical element, and he encouraged me to change the condition system from its original, Forth-like syntax to something more resembling BASIC.

The user interface design details were mostly mine – I wanted to make it easy to remember so for the most part the menus were driven by single-letter keys which were easy to remember.

When you say that the original syntax (design? implementation?) was Forth-like, does that mean that the internal stack-based code GAC uses is actually the original syntax, or were additional changes made beyond that?

The internal byte-code originally had a 1:1 relationship with the original syntax, but is (of course) a tokenized form of it. I'm pretty sure that I made changes as I went along, though. The original ADVAL system was effectively a prototype – it was written mainly in BASIC and consisted of 3 separate programs – the editor, the compiler, and the runner. As you can imagine, this wasn't very friendly.

Converting to machine code allowed it to be a lot faster and more compact.

Had you used similar adventure game creation tools? Which one(s)?

Apart from the type-in one, I hadn't seen anything else when I started. The Quill came out shortly before GAC, which was in some ways more powerful but seemed to me to be less friendly. PAW came out afterwards, but by that time I was (belatedly) concentrating more on my degree and didn't really have much opportunity to look at them in depth.

Did you have any background in creating software tools?

Not at the time. This was my first commercial software project.

What was the game made with GAC you saw that impressed you the most and why?

Incentive was a company that ran a lot of competitions – the prize was the incentive for playing – so we had a lot of entries for the GAC competition. The best ones were published but didn't really do very well commercially. I remember "Winter Wonderland" as the outstanding title, though. Good use of the graphics, a decent plot and some nice puzzles.

Tell me about any uses of GAC you saw where it was made to do things that you hadn't intended, or where it was made to do things that you didn't realize could be done with it.

The odd thing here is that I do distinctly remember someone using it for a very odd application – training, perhaps – but time has removed all my memory of it apart from the fact that it existed. Sorry.

The order of operands for the "in" operator seem to have been flipped around. What is the reason for that? (The initial manuals showed it as "o in r – Is object o in room r?", but later errata said it was actually "r in o" instead.)

No reason – that was a straight bug that we didn't catch until it was too late.

Looking back, what would you have done to improve GAC?

Buying an assembler! This would have given me a lot more flexibility and made it easier to spot mistakes and correct them. The extra time would have allowed me to do things like implement a proper flood fill command and optimise the dictionary used for descriptive text.

One of the most impressive GAC-related things was the GAC optimizer, which was developed entirely by people reverse-engineering the code with no help from me or anyone at Incentive. They managed to repack the data (including the dictionary) and make larger adventures than we had thought possible.

What was the part of GAC that gave you the most difficulty during its development?

The condition compiler and decompiler. I had got the idea of a decompiler from the Forth implementation on the Jupiter ACE, which I also owned at the time.

The GAC's BASIC-like conditional language was compiled to a stack-based byte code, inspired by both Forth and the byte code system used by the ZX81 to implement floating point. The original source code was then thrown away.

If you wanted to edit the code, it was decompiled from byte code back to the BASIC-like form for editing. Getting both transformations correct was difficult; we hadn't at that point done any compiler design in our computer science courses so I had to invent infix to postfix transformation (and back) from scratch.

As it was, I didn't get it completely right, hence the requirement for left-to-right evaluation order for mathematics in conditions.

By the time I wrote STAC for the Atari ST, I had learned a lot more and so the condition compiler was a lot better.

Did the problems that led to the requirement for left-to-right evaluation order and other bugs relate to the manuals not mentioning some features, like parenthesized expressions and nested IF statements?

No, the problems were down to me trying to derive infix-to-postfix compilation methods from first principles, rather than reading about it first.

Both parenthesized expressions and nested IF statements were used by various GAC games on the Spectrum, although they don't seem to be overtly advertised capabilities.

Yes, they were there but as you say not well advertised. Identifying brackets was easy; dealing with relative priorities of arithmetic operations was not.

The parser was hand-coded and having now looked at the code it's very simple. It just classifies each command as having 0, 1 or 2 arguments. 2-argument instructions were always infix, and did a bit of recursing to work properly, but the recursion was very simple and had no notion of operator precedence or left/right binding. The lexer and parser are about 100 bytes each – not much space for intelligence!

The decompiler does something similar but works backwards from the end of the program, and is about 3x the size.

Under the time and space constraints I had to decide that "good enough" was good enough.

Were the graphics done using routines you created yourself, or were you using a reference of some kind for the algorithms? If so, which one(s)?

The line drawing routines were standard in the CPC ROM, but the weird not-quite-flood-fill operation was all my own work.

How did GAC evolve over time as versions appeared for different platforms?

It didn't really evolve – the ports were straight feature-for-feature copies of the original, with the exception of the Acorn Electron version which couldn't handle graphics at all.

But see below about follow-on products.

Does the source code for GAC still exist in any form?

I think that I still have the hand-annotated listing around somewhere in the attic.

Tell me a story about GAC development.

I remember getting lost when first trying to meet with the Incentive guys at their office in London Street in Reading, and ended up literally trudging through miles of snow before giving up and trudging right back home again. Despite this terrible start, they agreed to see me again the next day.

Most of the other amusing stories don't involve development, but more the environment in which the development happened. The Incentive shop was also Ian's home, a somewhat dilapidated 4-storey Victorian terrace full of pokey little attic rooms where you could often find freelance programmers hunched over a BBC micro and a fan heater. I remember staying there over the holidays, in what was both Ian's spare room and the stock room for Incentive. I had a single bed, a desk, a computer, a Tempest arcade machine and several hundred boxes of Moon Cresta (Spectrum version) for company.

Are you still in contact with the other GAC authors?

Not really. I did meet up with Ian Andrew from Incentive a few years ago for an evening of reminiscence, but we're not in regular contact.

Thinking of the other GAC authors credited, what roles did they and you play in development of GAC?

The other authors ported the code to the other platforms. By that time the GAC was mostly finished. The most influential person on the design of the GAC, as I have already mentioned, was Ian Andrew. That's not to say that their role was not significant – there's no way that GAC would have been anywhere near as successful without a cross-platform launch.

How were the sales of GAC when it was released?

Excellent! It came out for Christmas 1985, and we also had a deal with Book Club Associates that made it the default choice for their software club.

I think we ended up shipping about 100,000 copies overall, but that's from memory so I may be way off.

Were you surprised by how many commercial games were created using GAC? Did this exceed your expectations?

I thought that there would be more. Only a very few were published, and those that were did not have great success. The same applied to the other adventure creator programs too.

People tended to see it as a hobby, I think, rather than as a way to make and sell games.

(Although looking at World of Spectrum I see that they have 117 games listed – I had no idea it was that many!)

What follow-on products, if any, did GAC lead to?

The next thing I did was STAC – the Atari ST Adventure Creator – which was much less constrained and had a better parser, conditions and graphics.

By that point, however, the text adventure craze was more or less over, and the suggestion that we create an Amiga Adventure Creator (AMAC?) never got off the ground.

Did you create any full-length games using GAC yourself? Which ones?

No. I did write the little example adventure, but I'm more interested in the technology of the tools than the final product. I'm the kind of guy who goes to see Pixar movies and gasps in all the wrong places because I'm watching the rendering of the grass rather than the plot.

What did you do after your time working on GAC? What do you do now? Are you still involved with games in any way?

While I was working on STAC, Ian's brother Chris had invented the ground-breaking Freescape 3D graphics system for the Spectrum and Commodore 64. They invited me to join them full time to port it to the ST and Amiga system, as their existing 16-bit programmer was leaving.

Incentive then had a number of games based on Freescape, including Driller, Dark Side and Total Eclipse. The system was then used as the basis for the 3D Construction Kit.

Chris, Paul Gregory and I then worked on the design of a replacement, "super" 3D system which inevitably became known as Superscape. The company changed direction seeking more serious 3D work, renaming itself first Dimension International and then Superscape. We developed the Virtual Reality Toolkit (VRT) which went through 5 versions.

VRT was also used for the TV programs "The Satellite Game" and "Cyberzone".

After that, we tried to get 3D onto the web with the VWWW (Virtual World Wide Web) and a VRML browser before moving to mobile phones and kicking off the Mobile 3D Graphics on Java working group (JSR184).

I worked on mobile graphics at Superscape until 2006, when I moved to ARM where I now work in their media processing group as part of the GPU architecture team.

Do you have anything to add?

I am amazed that something I dreamed up in my first year of University became so successful, spawned a dedicated set of fans and still generates questions 30 years later.

Epilogue: Source Code

Sean kindly scanned in his source code listing and notes and, with agreement from Ian Andrew, has made it available at `http://moteprime.org/article.php?id=62`.