User Guide

CRYSTAL CASTLE

for the BBC Microcomputer

GRAHAM NELSON
and TOBY NELSON
and ADRIAN NELSON

LOBSTERSOFT

Contents

1 About CRYSTAL CASTLE

This manual is written to explain the operation of CRYSTAL CASTLE on the BBC Microcomputer Model B.

CRYSTAL CASTLE is an interactive story of exploration and sorcery. You play the part of a powerful magician, who has always dreamed of Immortality. Years of adventuring took you to Sumatra, Shanghai, and Tibet in quest of eldritch wisdom,* and with your growing power, you grew rich. You built your own CASTLE, and set your own magical traps and guards within it, including the fearsome Shreves.

You thought yourself invincible. Every secret would be yours. And for so very long, it was. But you never had... IMMORTALITY.

And so you set out again, to the darkest corners of the world. Aztec temples. Icelandic monasteries. Desert islands. To locate the pure crystals of LIFE. To bring them back to your laboratory (located conveniently under your stairs). To combine them into one enormous life-giving CRYSTAL, bathe in its rays, and live forever.**

And it nearly worked.

But the last combination was too much. The CRYSTALS burst apart, scattering all over the CASTLE, with their dangerous magic rippling out. As for you, you were expelled to your gardens, to be reborn... as your pet snake, Mr Wriggles.

Do you have what it takes to regain your CASTLE? Every crystal has life within it. That may help. But the creatures you created have become monsters now. Even your guards. Especially your guards.

Good luck, once-potent one...

*** Not included in this edition.**

**** Also not included in this edition.**

2 CRYSTAL CASTLE controls

The Magician is controlled by the following keys, located on your BBC Microcomputer keyboard:

Left	**Z**
Right	**X**
Up	**:**
Down	**/**
Jump	**RETURN**
Transfigure*	**SPACE**
Sound off	**Q**
Sound on	**S**
Pause play	**P**
End game	**SPACE + ESCAPE**

*** After making a discovery early in the game, the Magician can Transfigure between Snake and Human form at will. The player of guile will know when it is best to be which.**

The Magician can carry certain objects, but no key press is needed to take, drop or use these. Simply bring them to where they are needed. But beware, only one object can be carried at a time!

3 The history of the CASTLE

i) May-June 1984: The Engine Demo

CRYSTAL CASTLE was born in July 1984, when the authors were 14 (Toby), 16 (Graham) and 18 (Adrian). Released from Great Baddow Comprehensive School to revise for O-levels, Graham had elected to spend the time more productively, playing Deep Purple's already-dated *Fireball* album (1971) and setting up the mechanics of a new platform game.

The stick-figure hero, whoever it was, could walk along, or up and down stairs, and could climb (though not descend) ladders. Some of the platform walls eroded underfoot — "crumble bars", as we called them — and could also be jumped through. There were conveyor belts moving left or right, though not animated ones, and as a one-off piece of furniture, tables to stand on. But all these design elements used the same graphics in every room. There was just one left-up staircase sprite, just one wall.[1] The object: to, for some reason, collect the crystals.

1 In the 2024 edition there's much more visual variety, and there are two new design elements: scenery (mere decoration which the player can't interact with), and spikes (like walls, but lethal on contact: we seldom use these). The humble table became a more general-purpose platform.

Most exciting of all was the ability to walk off one screen and onto another, in an adjacent "room". This was a clear steal from Matthew Smith's classic *Jet Set Willy*,[2] which Graham and Toby had played shortly after its release earlier that summer when visiting family friends in Spencer's Wood, Reading. *JSW* became a smash-hit on the ZX Spectrum, and was the best-selling UK game of 1984. But on the BBC Micro, for another few months at least, its trademark map-made-of-screens mechanic hadn't yet been tried. (Michael Jakobsen's splendid *Citadel* would not appear until 1985.) Fame and fortune beckoned.

JSW ran on a 48K Spectrum, with memory so abundant that its huge house design and its elaborate sprites could be stored uncompressed. The game deftly turned the limitations of the Spectrum's unusual video chip to advantage. The room grid and monster movements were aligned with the 8x8-pixel character cells of the screen memory. Since each cell had its own two-colour palette, natural-looking room designs at a 256 x 192 resolution used only around 7K of screen memory. With the sprites being two-colour, they could be stored as just 1 bit per pixel: a 2x2-cell four-state animated monster occupied only 128 bytes. With 40K of RAM free for game code and design, the program had plenty of space to breathe, and luxuriated in visual plenty.

CRYSTAL CASTLE, though, had to fit in a BBC Micro Model B with 32K of RAM and a much less accommodating video chip. We couldn't afford to burn 20K by running in graphics modes

---

2 So named because the main character, Miner Willy, had become rich after winning *Manic Miner*, and joined 'the jet set' (a term coined in 1949) of rich frivolous party-goers.

0, 1, or 2, so the only options were 4 and 5, and even those consumed a painful 10K. Mode 4 had the resolution we might have liked, and could have made for a stylishly monochrome game, but we went for the unloved Mode 5: 160 x 256 pixels at four colours. Two were always black and white, for a consistent background and player sprite (not counting his hair), but the other two could vary. Some rooms would be red and cyan, others green and magenta, and so on. It was a compromise, and it felt like it.[3] And even then we'd have to cram the game into around 20K of RAM, not 40.

If we had used the entire screen as play area, every room would have had gapingly empty spaces to fill. *JSW* had wisely confined itself to a 32 x 16 design grid for its rooms: in Mode 5, given the byte alignment of screen memory, the equivalent for us would have been 40 x 22, and we quickly realised this was far too much. Rooms with names like BEDROOM or CLOSET would have had the equivalent of sixty-foot-high ceilings. So the engine put in a cross-hatched border around the play area, trimming it to "only" 36 x 18 cells.[4] That was still a lot to fill, as we would soon learn. Unlike some other games of the era (*Pitfall*, *Rescue Esmerelda*, or *Jet Set Willy* itself), we didn't have swinging ropes which could fill voids in the room design with movement and something fun to do. Our 36 x 18 rooms would have to be filled the hard way.

That horizontal resolution of 160 also made the BBC Micro's built-in font unwieldy. Text drawn with it was so distorted that the whole screen was only 20 characters wide. *Citadel*, a year later, accepted this trade-off, and as a result its room names had to be very short ("The Prison"). We wanted names more like THE MONSTERS SECRET HQ[5], or OUT THROUGH THE CHIMNEY POTS. That meant burning further bytes on a custom font, though as an economy measure it contained only up-

---

3 The 2024 edition uses a custom screen mode, with timed interrupts to switch the palette multiple times each 1/60th of a second as the scan beam passes down the screen. This sort of video manipulation was a trick pioneered by *Elite*, which wouldn't be released until September. If you'd asked us in July, we would have said it was impossible.

4 The border is now absent because of the custom screen mode, except for a simple white framing rectangle, which survives from the original design.

5 It now has a different name. Originally Toby's idea in 1984, it is now the room he likes the least, but Graham refuses to remove it. Other renamings include THE PURPLE SNAKE ROOM.

per-case letters.[6] Another quirk of Mode 5 is that pixels are twice as wide as they are tall. Sprite design is faintly weird as a result, and sprites can't be rotated 90 degrees.[7]

All moving elements other than the player were "monsters".[8] These were very limited in 1984. They were what we now call "bouncers", meaning that they patrolled back and forth between fixed endpoints, either vertically or horizontally. And they all went at the same speed.[9] *JSW* had been a little more varied — it had arrows, for example — and even then it compensated for the relatively simple motion of its monsters with a marvellously-drawn set of four-state animations, often playing little games with height to play off against its grid layout. (The flying duck which lopes through the air, bobbing up and down, is one of its many design triumphs.) With less memory, and less resolution, and four-colour sprites to be stored, we just couldn't afford four-state animation, and so the Castle's monsters were and still are two-state animated.[10]

The only controls were Z, left, X, right, and RETURN, jump. Jumping and falling was fairly plausible, and the player couldn't steer his movement, in the style of *Super Mario Brothers* — a game which had not yet been written.[11] But

---

6 We still have our own font, though it now has stylised descenders, is less blocky, and is no longer fixed-pitch: take a look at the I in THE HALL FIREPLACE.

7 Though they *can* be reflected in the x- and y-axes or have their colours permuted, all of which the 2024 engine can do, but the 1984 one couldn't.

8 Today there are animated scenery elements such as rippling water surfaces and braziers with moving flames, but none of that was present in 1984.

9 Today they run complex little programs on a virtual processor called MASH, and can change speed, follow each other, execute strange dances, send each other signals, and so on, but there are still a few 80s-style bouncers around 2024's castle.

10 Though the 2024 edition has pixel-level movement, not character-cell, so it is very much smoother, and does at least have a four-state animation for the player. Its sprite-plotting is much faster than in 1984, and is also now timed to avoid flicker (caused by the scan beam passing through partially drawn sprites).

11 Jason Begy's 2010 paper *The History and Significance*

physics in the game could otherwise be a little odd. You could stand on top of a crystal, for example, even though touching it in every other way would cause it to be taken and disappear. There were no up or down controls because ladders were simply vertical stacks of platforms to jump up. We never really questioned the deeply strange consequence that the player had no way to get down again. Instead we had to provide downward routes consisting of successive crumble bars for the player to fall through.[12] Or else we made deliberate puzzles out of ladders stranding players in mid-air, as in THE HANGING LADDERS ROOM.[13] And there was no transfigure control either: Mr Wriggles, the snake alter-ego, did not yet exist. All this made the game simpler: left, right, jump.

One area in which the BBC Micro's hardware far exceeded its rivals was in audio, where its deluxe four-channel sound chip was perfect for gaming. Our Castle felt noisy, because there was a constant da-da-da-da as the player ran, and most of today's swoops and pings (for falling, or crumbling crumble bars, or getting crystals) are only modest refinements of the 1984 originals. Every monster bounce made a blip. The clatter was annoying enough through a BBC Micro's quite muscular speaker that the 1984 game opened by asking if you wanted to play in silent mode.[14] But there was no music, and really we could have done more.[15]

---

of Jumping in Games suggests 1981's *Donkey Kong* as the first mainstream game with a jump mechanic, with 1982's *Pitfall* cartridge for the Atari 2600 bringing it into the home. These jumping motions were mostly quite simple, without much concession to realistic physics, and it wasn't until 1985's *Super Mario Brothers* that this changed. We didn't make the Castle's jumps steerable in mid-air until 2020, and although trajectories are more parabolic now, there still isn't anything you could call a physics engine.

12 There are still a few of these in the Castle even today, for example in THE LONG WAY ROUND, whose design we've sentimentally preserved from 1986.

13 Now removed, though hints of its design live on in the west tower.

14 It also offered three play speeds, Slow, Medium, and Fast. We prided ourselves on always choosing Fast; the 2024 game plays at approximately Medium.

15 A music engine was one of the first 21st-century additions to the Castle. Tunes have come and gone: at one time *Greensleeves* was played in the garden rooms; and for sever-

Looking back at the demo, then, some of its limitations were down to video hardware and memory constraints. But in other ways, that first draft could easily have been souped up. Why wasn't it? Because another hindrance, to put it mildly, was that we had no development machine. The game was built in the same BBC Micro it would run in. Subroutines for the engine were written using the excellent 6502 assembler built into the BBC BASIC ROM. But the memory to store this textual assembly language — the engine's source code — would soon be needed for the level design and sprites. Nor could we have a series of loader programs which assembled the game from source: we had only tape cassette for storage, and no disc drive. So by July there was no source code for the engine at all. It existed only as opcodes in memory, at addresses we had to remember. Bugs could be fixed only by disassembling and patching, and new features were more or less impossible to add. And so, even though it now seems clear that the demo engine was a first draft, there wasn't going to be a second. For better or worse, we had our engine.

ii) July 1984: Denizens of La Villa Strangiato

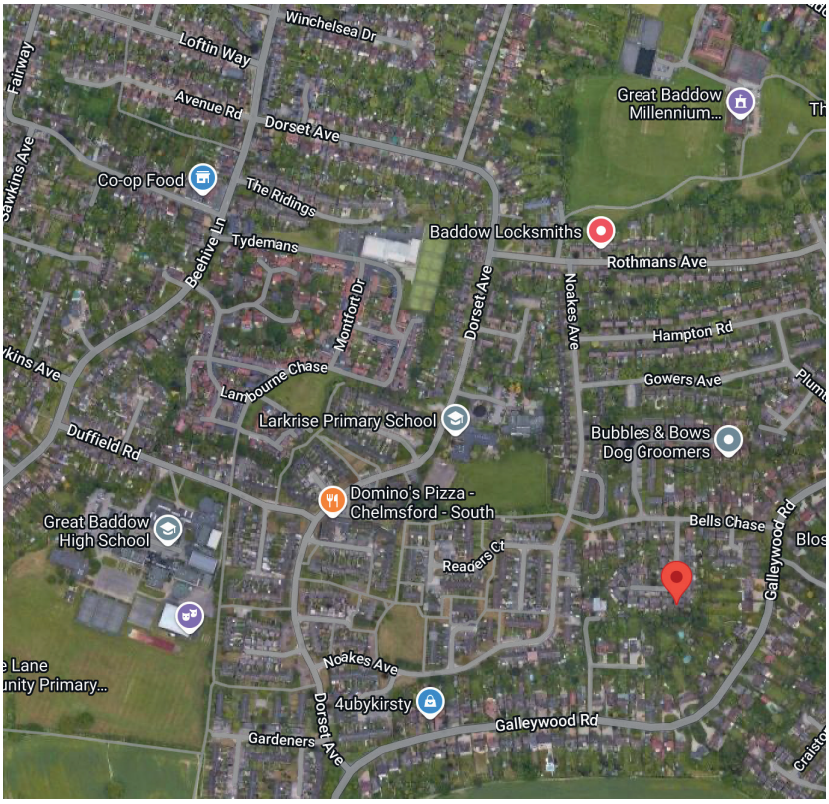
The demo version had only plaited two rooms together, the Great Hall and the Hall Fireplace, and both were quite bare with a lot of plain black background, but they promised a huge virtual geography to come. Like *JSW*, the Castle would have west, up, east, and down as its directions, making a vertical slice through a building and its grounds. And it would be stocked by monsters, though we always called them "denizens", a favourite word of cousin Adrian's. He had finished his A-levels that same early summer, and came over from Portsmouth to stay. Adrian remembers:

> "Almost as soon as I had brought my bags in from the car, Graham showed me this demo, it consisted of one (or maybe several?) screens [...] we decided to see what we could accomplish in the next two weeks, and see if we could complete the game, and maybe even sell it."

It was a halcyon summer, and we did play a little tennis,

---

al years the game opened with *Arrival of the Queen of Sheba* in the start room, and played the *Hallelujah Chorus* in the victory room. These have now gone. Elaborate musical scores take memory: still, the game does find space for a main title theme, Prokoview's *Dance of the Knights*, and little musical stings are heard from time to time, including one which hadn't even been composed in 1984.



Roughly a 1km square. Graham and Adrian at point marked in red; Toby at school in southwest of frame. Tennis courts (not then in the "Millennium" park, and grass, not clay, back in the day) in the northeast corner.

but otherwise it might as well have been a snowstorm outside. We were indoors. By day we level-designed in colour, scribbling on quadrille paper. Rooms were stored as a list of "strips": a staircase of length 4 rising right from (2, 3), a horizontal wall of length 7 starting at (10, 8), and so on. We reached the point where we could glance at a sketch of the layout and reel off the necessary bytes in hexadecimal. We listened quite a lot to the classic run of Rush albums, 'La Villa Strangiato' from *Hemispheres* (1978) being a favourite number: we named our daftest monster the Shreve in its honour.[16]

---

16 An instrumental based on a bizarre dream, `La Villa Strangiato' has twelve named sections, and Part VIII is `The Waltz of the Shreves'. Shreve is an archaic form of "sheriff", the "v" usually having become an "f" by around 1500, so goodness knows how Rush got hold of it. Rush fans may be pleased to learn that today's castle also has a room

Top row: Adrian, Toby, Graham (architects).
Bottom row: Helen, Violet, Pippa (girls).

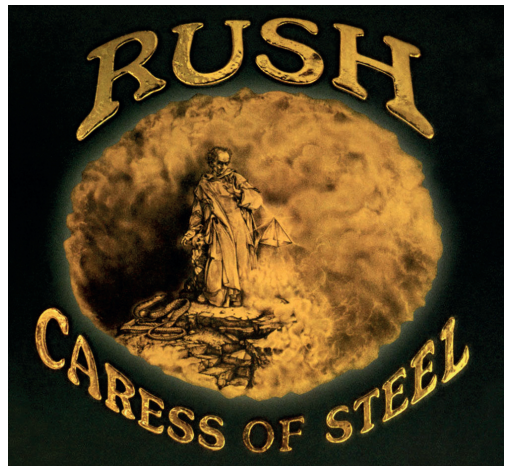
While Graham and Adrian were laying out architecture, Toby, to his enormous annoyance, was still in lessons.

> "I well remember the frustration of those long, dull school days. The moment the lunch bell rang I would sneak back home, and for a fleeting hour I could see how the Castle was progressing. Mostly I would play-test the latest version, trying out newly added map locations, looking for bugs, and generally making suggestions of widely variable degrees of helpfulness. I recall playing a new screen called XXX XXX XXXXXX XXXX[17] for the first time, immediately finding its secret exit on the very first attempt - much to the annoyance of Graham and Adrian. Otherwise, my input would be to de-

---

named after Part VI, THE GHOST OF THE ARAGON, and might want to listen out for the track's "Monsters!" leitmotif, which the game occasionally plays. Rush themselves subtitled this work `An Exercise in Self-Indulgence' (which on an album like *Hemispheres* was saying something), but they were in fact exceedingly pleased with it, and played it at 956 subsequent concerts (including one we attended) before their final performance in 2015.

17 Spoilers.



These albums were all massively out of fashion in 1984. The authors played them over and over on the portable cassette player otherwise used for saving programs.

> sign some animated sprites (the candlestick was a favourite of mine, I think?). In my defence there's only so much you can do with two state animation, a handful of pixels and just four colours. The secret was not to try and design a specific object, but to put down a few pixels almost at random, see what it started to look like, then develop it from there. Often, the object you ended up with was completely different from the one you intended to design!"

And so we all became possessed. By night, bewildered adults having returned to the house, we moved to the kitchen table and a black and white portable TV screen, where we simply carried on, fuelled by frozen Mars bars. Here it was, at two-thirty one morning, that a horrid engine bug turned up... and the adrenaline began to flow. Adrian:

> "Graham disassembled the machine code in his head and directly entered hex codes over the previous contents, rewriting subroutines on the fly. Time seemed to have no meaning, but suddenly real progress was being made, and dawn crept up on us about 4.30am, by 5.30am it was fixed and we could go off to bed, only to awaken at 10am to continue."

The Castle grew in all directions, with sudden growth spurts as we got interested in particular areas: we spent about two days on the gardens, for example. It was easy to lose sight of larger-scale design issues as a result, especially as we hadn't started from any real plan. Why was there such a gigantic chimney cutting the whole Castle in half, for example? Wasn't that a little disproportionate? But we found ways to join these regions together, through means like THE BALUSTRADE (another Adrian-pleasing word). Since we all three played the game as obsessively as we wrote it, the map lived in our heads, and we never really kept a paper version. There was no question of changing the fundamental layout. Once a room was in place, it seemed quite real to us, as if we had discovered it rather than invented it.[18] And somehow, just as the fortnight was up, we had our 64-room castle, and its denizens, and we all three of us knew every inch of it.

---

18 For the 2024 release we felt much more able to rebuild the castle, to remove tedious areas (THE ATTIC and THE LIBRARY, for example), and give it more coherence. Even so, the two map PNG images, if placed side by side, are remarkably similar.

iii) 1985-99: The Lost Years

So, that only left the fame and fortune part. Graham:

> "I had sold an Acorn Atom version of the arcade game *Frogger* to A & F Software two years earlier, so I considered myself a pro, obviously. This time I wanted to sell to Program Power, then the top BBC Micro games firm, whose back-cover ads in magazines like *Acorn User* seemed the very pinnacle of fame. As it turned out, they did take something of an interest, and we exchanged a few letters. They rejected it for not having enough animation — the four-colour graphics, two of them always black and white, didn't help — but gave the impression that they'd sneakingly liked playing it. I don't think it was entirely a matter of being kind to a couple of kids."

And so the game never made it out of the Program Power slush-pile, and was never commercially released.

There was occasional talk of reviving the Castle, mostly because we had fond memories of creating it, but we had universities to go to. Once we had moved up to the Acorn Archimedes (1987), the whole thing no longer made any sense. Trading up from 4 colours at 160 x 256 to 16 colours at 640 x 512, and from 32K of RAM to 512K, and from a 6502 processor to an ARM chip, all made the Castle and its constrained little world seem a complete mismatch for the computer it would run on. The ability to have, say, another three thousand rooms, and hundreds of new sprites, simply made us realise that we would need artists, and that we wouldn't ever get the necessary forty or so summer fortnights to design it anyway.

So the Castle was history. The quadrille notebooks were all lost. The source code had never existed at all, and even the program was gone, and nobody was even looking for it.

iv) August 2000: The Two Tapes

But fifteen years later, Toby found a cache of 1980s cassette tapes: half of them music, half computer programs, some going back to the BBC Micro's predecessor, the Acorn Atom. (Graham had had an Atom, with a mighty 12K of RAM, and had learned 6502 coding on it.) Toby remembers:

> "The tapes that caught my eye were clearly labelled *Crystal Castle* on the inlay. From them, the shaky, but unmistakable tonal patterns of the (now antiquated) BBC micro cassette format were played back and recorded

into a modern day PC. Could it be possible that every single binary bit of data from the original files would still be correct on these tapes, and could be correctly decoded? It seemed unlikely, and my initial tests seemed to confirm this unhappy thought. I downloaded a program that could read BBC tapes from the internet, and tried it. Practically every block of data was a 'bad block' and could not be read. All I managed to get were the filenames of the four files on the tape. I left it for a while, resigned."

But the BBC Micro design, so full of good intentions that it was sometimes impractical for what its users really wanted (see discussion of screen modes above), may now have saved the Castle with its diligence. The BBC Micro wrote cassette tapes in a well-specified format known as the Kansas City standard, 1976, which assumed that its hardware and media would always be fallible, and the BBC further divided files up into blocks which had checksums. The result is a resilient medium where it's possible to be *fairly* sure whether any given data has loaded correctly or not.

"About a week or two later, I came back to it, and tried the second tape. This time I had some success. About 60% of the blocks were readable. I managed to get the three short loading programs correctly but the all-important main file called 'work' remained. I altered the tape reading program (fortunately it came with source code) to save out the correct blocks of data in their correct file positions, with bad blocks padded out with zero bytes. I cleaned up the tape signal in different ways with an audio editor, each time running the results through the program, getting a few more of the blocks correct."

But even that didn't do it: a few stubborn blocks held out, and in the end Toby rerecorded the original tapes into WAV files hundreds of megabytes in size, and reconstructed the 20K of actual data on them by examining the waveform directly. By 29 September the job was done:

"Even now I was sceptical. Could they really be correct? I downloaded a BBC Micro emulator from the internet, and scrambled for ages getting my raw files into the right format for the emulator. Eventually I got to the stage where I could type CHAIN"CRYSTAL" and hit RETURN. This was the moment of truth. I was fully expecting the dreaded 'Bad Program' error message to appear, or for the screen to just go blank. But within a few seconds, the title screen appeared, large as life. The whole game worked perfectly first go."

"PS.", he wrote in a contemporary email, "The sound effects are really bad." On the other hand, he could still remember how to play. "I just instinctively know where all the right jumps are."

The Castle's survival had hung on just two cassette tapes, recorded in an obsolete format, degraded to the point of unreadability, and then lost. It had survived all the same.[19]

And where better to live forever than the Internet? Toby created an amusing download page on his personal website, www.tnelson.demon.co.uk. His final service to the game, or so he thought, was to screenshot the 64 rooms one by one, edit out the player sprite from each, and then painstakingly glue these together into a giant map image.[20] That, once again, seemed an end to the story, and for sixteen more years it was.

v) 2016-2017: The reassembly and the CONTINUUM

Onward to the autumn of 2016. CRYSTAL CASTLE was now 32 years old and its teenage authors now had an aggregate age of 144. Toby had continued to host the game on his public website, but Demon, his Internet service provider, had been sold several times and was being wound down by Vodafone. This meant the demise of www.tnelson.demon.co.uk.[21] Somehow, the Castle had moved back into private ownership. And, at some point around September or October 2016, Toby began to poke at the binary copy of CRYSTAL CASTLE:

> "Firstly I was curious. I wanted to understand how it worked. I also wanted to be able to extract the long lost original room data, perhaps thinking of recreating it on modern hardware. But to extract the room data it was necessary to understand a great deal of the code..."

This turned out to be a fateful moment, since it eventually led to him becoming "TobyLobster", BBC Micro hacker.[22]

---

19 This is more than can be said for the two cassette tapes, which have once again disappeared.

20 And also showing a few misalignments in the original, where our rooms hadn't quite matched at their joining boundaries, most notably on THE BALUSTRADE.

21 Though, of course, the Wayback Machine will provide.

22 The *-lobster* suffix has something to do with fellow

His projects would later include a commentary and disassembly of the MOS ROM, the machine's 16K operating system (2020);[23] a performance analysis of all known multiplication and square roots algorithms using 6502 machine code (2022); a catalogue of sideways ROMs for the platform (2024); and "reassemblies" of a number of classic games, with "remastered" and improved versions to follow.[24]

Though it was done in private, disassembling and then reassembling the Castle was Toby's first tentative step in this direction. "Reassembly" meant more than "disassembly": it involved disassembling a game (or a language ROM, such as Acornsoft LISP) into source code which could then be run through a modern 6502 assembler to produce an executable BBC Micro program bitwise identical to the original. He learned, or relearned, 6502 assembly along the way, and fell back in love with a computer which (ironically) he himself had never actually owned.[25]

By November he had put together an annotated source code for the game which could be assembled to a byte-for-byte identical copy of the original binary. But while the engine was now fairly comprehensible (despite the mess left by its chaotic writing and debugging history), the data structures for the sprites and the room layouts were still large globs of hexadecimal data. It was at this point that Adrian, too, returned to the Castle, and wrote a Javascript viewer for the sprites. Room data was extracted to an ASCII-art representation, and Adrian wrote an initial draft of a utility to encode that text into strips, and another to compress text. Suddenly, the Castle had its own build tools.

Adrian's ASCII notation was an inspired idea and is still essentially in use today. Here is the Great Hall:

---

team members at Electronic Arts calling him "the Tobster".

23 It has its flaws, but the MOS revealed itself to be an exceptionally sophisticated work given its time and place.

24 Among them the BBC ports of *Jet Set Willy* and *Manic Miner*.

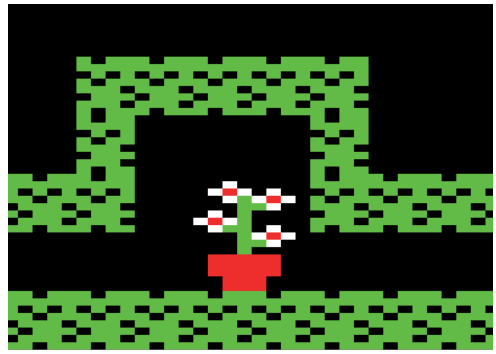
25 Though he did own an Acorn Electron in 1984, and was to buy a reconditioned BBC Micro Model B in March 2017. In 2024, it's still working: the BBC Micro, whatever else can be said about it, was built to last.

```
; ------------------------------------------------------------------------
Room THE GREAT HALL
; ------------------------------------------------------------------------
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 1 2 3 4 5 6 7 8 9 0
a #!. H . . . . . . . . . . . . . . . . . . . . . . . . . . . . H . . #!
b . . H . . . . . . . . . . . . . . . . . . . . . . . . . . . . H . . .
c . . H . . . . . . . . . . Sk. . . . . . . Sc. . . . . . . . . . H . . .
d . . H . . . . . . . . . . Sk. . . . . . . Sc. . . . . . . . . . H . . .
e #f#f#f#f#. . . . . . . . . . . . . . . . . . . . . . . . . . . . #f#f#f#f#f
f . . . . . . . . . Sb. . . . . . . . . . . . . . . . . . . . Sb. . . . . . . .
g . . . . . . . . . Sb. . . . . . . . . . . . . . . . . . . . Sb. . . . . . . .
h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
i . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
j . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
k . . . . . . . . . . . . . . . . . . . . . . . . . . . . ^0. . . . . . . . . .
l . . . . . . . . . . PaPaPaPaPaPaPaPaPaPa. . . /i#g#g\j. . . . . . . . .
m . . . . / # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
n . . . / # [ . . . . . . . . . . . . . . . . . . . . . . . . . . # # # # #
o . . / # [ . . . . . . . . . . . . . . . . . . . . . . . . . . # [ . . .
p # # # [ . . . . . . . . . . . . . . . . . . . . . . . . . . # . /i# #
q # # MaMbMb. . . . . . . . . . . . . . . . . . . . . . . . . . # . # # #
r # # MaMbMb. . . . . . . . . . . . . . . . . . . . . . . . Fh# . # # #
```

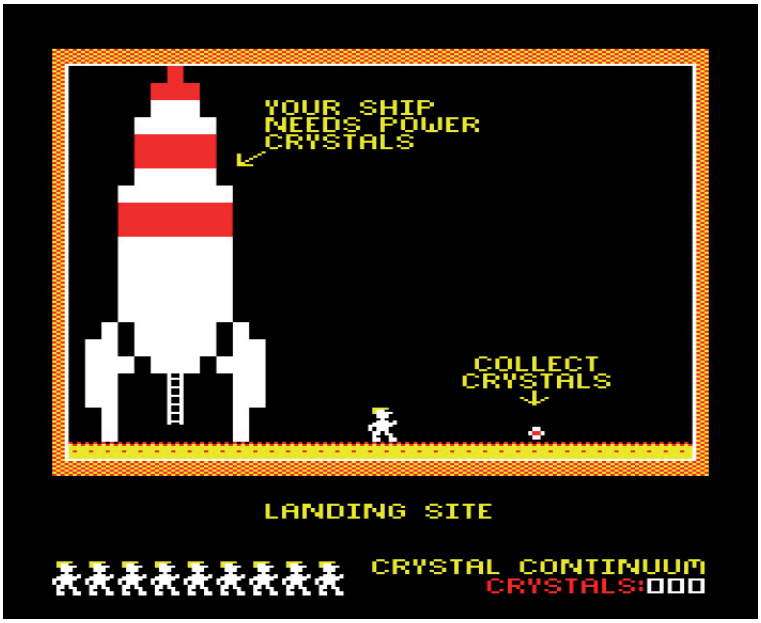
Each cell of the 36 x 18 grid is represented by two ASCII characters, and an absence of content is represented by a dot, not by white space. The use of coordinate axes at the left and top also turned out to be wise.[26]

Sprites also had an ASCII-art source. Here's a flowerpot:

```
Sprite <FlowerB> (2x2)
-  -  -  -  -  -  -  -
-  -  #  -  -  -  -  -
-  #  ?  #  -  #  -  -
-  -  #  o  #  ?  #  -
-  -  -  o  o  #  -  -
-  #  -  o  -  -  -  -
#  ?  #  o  -  -  -  -
-  #  o  o  -  #  -  -
-  -  -  o  #  ?  #  -
-  -  -  o  o  #  -  -
-  -  -  o  -  -  -  -
-  ?  ?  ?  ?  ?  -  -
-  ?  ?  ?  ?  ?  -  -
-  ?  ?  ?  ?  ?  -  -
-  -  ?  ?  ?  -  -  -
-  -  ?  ?  ?  -  -  -
```



26 Rows use lower-case letters, and columns upper case. Adrian's decision to label the columns A to Z, then 1 to 9, *then* 0, was... less inspired, but also lives on.



The notation here is - (black), o and ? (colours 1 and 2, which vary from room to room), and # (white): this is a sprite which will display well only in a room where o comes out as green, and then it will have white-rimmed petals, and come out of a flower-pot of some description.

Toby, meanwhile, saw no harm in tidying up the source code. Room data in memory had been in a fairly random order,[27] and he rearranged these more logically. The engine code was also a jumble of subroutines, which would benefit from a certain amount of reorganisation.

By December, with what amounted to a Castle developer's kit in existence,[28] Toby and Adrian were no longer pretending to be only interested in a faithful restoration of the original game. They even contemplated a sequel:

> "Once we had a way of creating rooms, Adrian and I created a spin off project called CRYSTAL CONTINUUM. Set on an alien planet, the main character had to collect crystals to power up his spacecraft to get home. Only

---

27 That is, rooms occupied memory in the order in which they were designed in summer 1984.

28 Though Toby's development machine was a Mac, these converter tools were largely written, at that point, in C#: Adrian was a Windows user, and Toby liked C# too, though the Mac wasn't its natural home. Graham, having never used C#, later contributed further tools in Perl.

> a few rooms were created, but it had teleports, monsters that followed paths, and multiline formatted text with choice of two border styles, printable on the game area. Key cards and doors were a todo item."

But this demo revealed the limitations of the original engine, which, after thirty-three years, was finally about to get its second draft. In January 2017, Toby rewrote it to give the monsters and the player pixel-level movement (that is, they were no longer plotted only at character cell boundaries, but moved smoothly in between), and redrew the player sprites, giving the player four-state animation. And whereas the 1984 engine had allowed only allowed square monsters — occupying a sprite area of either 2x2 or 1x1 cells — 2x1 monster support was now added.[29] The engine also finally got the more careful bug-fixing revision it had long needed.

This process continued, off and on, until May of 2017, but it wasn't entirely clear where it was going. Was the aim to produce a new BBC Micro game, or to extract the Castle design for a game on a more contemporary platform? A brief experiment with porting the engine to Unity in October and November didn't get very far, and the reassembly project came to an end. But, it had succeeded in revealing the secrets locked inside the one surviving binary copy of the game, improving the play experience, and establishing a modern build environment for any future work.

vi) 2019-2021: "The 35th Anniversary Edition"

The Castle slept unattended, as it so often had, for two more years. During 2018 Toby had taken an interest in using the BBC Micro as a music player, of sorts, and had written a tool which could transform an XML representation of a musical score into a BBC Micro disk image of a program which would play it.[30] In May 2019, a reduced form of this, using only two of the BBC's four sound-chip channels since the game was already using the other two for sound effects,

---

29 Thus making the woodworm monsters in THE RAFTERS notably fairer, since collision detection between the player and the monsters was then done on the basis of sprite boundaries, and so even though the woodworm occupied only the lower half of their 2x2 boxes, it was death to touch even the upper half. (1x2 monsters, incidentally, remain unsupported today.)

30 Philip Glass's *Mad Rush* was particularly memorable.

allowed the Castle to have background music for the first time.[31]

Endless pianola-style repeating music had been a feature of early platforms-and-ladders games: *Manic Miner* had run *Hall of the Mountain King*, while *Jet Set Willy* had played *If I Were A Rich Man*. Perversely, though, the effect of a continuous musical backdrop was a more intrusive on a BBC Micro, simply because its sound hardware was so much more robust. Toby therefore intended music to be played only in certain rooms, and the début theme was *Greensleeves*, heard in the garden rooms.[32]

To this point, though, the game had still not really been changed: the engine was better implemented, it had a more interesting soundscape and smoother graphics, but the mechanics and Castle design had not altered at all. That somehow seemed more of a transgression.

And yet, the 35th anniversary was approaching.[33] Collaborating with Graham this time, after the two of them had had a vague conversation about the idea while walking across the Golden Gate Bridge the previous year, Toby now set up a GitHub repository, putting the Castle under source control for the first time. The build tools would print, on every build, the number of bytes of BBC Micro RAM free, a number which dominated everything we now did to the game.[34]

---

31 The 1984 game had played some form of Handel's *Hallelujah Chorus* on the victory room screen, but there had been no music as backdrop to game play.

32 Later dropped, alas. Even when heavily compressed, music costs memory: the 2024 title theme, the *Dance of the Knights* from Prokoviev's Romeo and Juliet, consumes 132 bytes — as much as a typical Castle room.

33 Admittedly... not the most compellingly round number. Fleetwood Mac's *Rumours* album had a 35th-anniversary rerelease (in 2013), but most of the classics did not.

34 It wasn't quite so simple, because there were several pools of memory to exploit. The stack, in theory occupying all of addresses 0x0100 to 0x1ff — or rather, vice versa, since it descended from higher addresses to lower ones as it filled — tended to have spare storage room underneath it, for the brave, provided nothing too recursive was happening. What memory above that was available for use then depended on how many operating system (or BASIC) features needed to carry on working. An inflexible rule was that, no matter what, a subroutine to handle interrupts had to be

Whatever graphics, sound, engine features or room design we added would reduce this number: whatever savings we could make through more devious encoding would increase it. Five years of pushing and pulling would follow, with the count of bytes free sometimes pushing 2000 and sometimes in single digits, or even, on a few occasions, zero.

As an example of a push from 2019, Toby added animated braziers to the engine, making it possible to have flaming torches on the wall for the first time. This meant adding a 1x2-cell sprite plotter, and of course finding the memory for animation states of the flames.[35]

As an example of a pull, though, this was when the map table almost evaporated to nothing. The 1984 game allocated 4 bytes to each room which stored the room numbers of their neighbours to left, right, up and down: altogether, 256 bytes. Eventually Toby realised that, *provided* the rooms were numbered consecutively from the top left of the map to the bottom right (which at that time they were not), and *provided* the layout was always physically consistent (which they were),[36] and *provided* there were no gaps at any given height (which there were not), almost nothing needed to be stored at all. For example, suppose a room layout like so:

```
      00
   01 02 03 04
05 06 07 08 09 10 11
   12 13
```

In all cases, subtract 1 from the room number to go left, add 1 to go right. To go up or down, all you need are the magic sequence of numbers (2, 5, 6) — the numerical differences between horizontally aligned rooms one level apart. So the 256-byte map table reduced to just 9, a huge saving. On the other hand, of course, the lookup code to interpret those bytes took more instructions, growing the engine. But it was still a net win of around 200. This was always the story for future savings: the trick is not to find new ways

---

located at 0x0D00. Once out of the lower few KB of memory there were fewer constraints, but of course the game still had to share with the screen memory.

35 Rippling water surfaces and spikes lunging up from the ground were added in October 2020, sharing code with the braziers.

36 Interestingly, *Jet Set Willy*'s is not, with several east-west routes impossibly short.

to compress, but to find ways whose benefit will not be lost again because of the code needed to decompress them.

The only road to greater savings was to economise on the single largest data structure in the Castle: the "strips". One of the few good technical decisions of the 1984 demo engine had been to store rooms as a list of strips of content — walls, ladders, staircases — starting at a given character cell and continuing for a given length and direction. This was simple to decode, and far from inefficient. Each strip occupied three bytes: type and length, R (row) origin, C (column) origin.

But clearly bits were being wasted. Graham sketched out a more concise scheme based on a virtual processor, called MASON,[37] which had six registers: R, C, T, N, S and F. A program for MASON is a stream of opcodes encoded as:

```
bits: 7    6 5      4 3 2 1 0
      DO   CHANGE   VALUE
```

A single opcode causes one of the four registers R, C, T, N to be changed (the CHANGE segment determining which, and the VALUE saying what it changes to), and can optionally, if DO is set, cause a strip to be plotted using the current register values.[38]

At first sight this is less efficient, not more. Here's a staircase occupying 4 bytes not 3:

```
Change R to 10
Change C to 12
Change N to 3
Change T to "stairs down and left" and plot
```

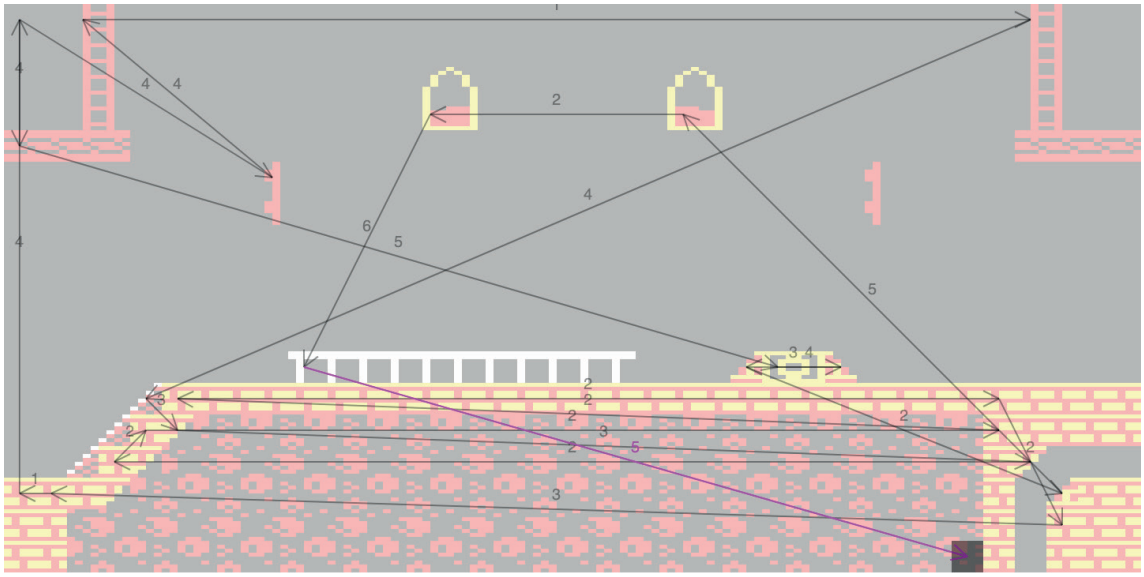
But the benefit comes because not every register needs changing for every strip. Consider:

```
Change R to 10
Change C to 12
Change N to 8
Change T to "wall" and plot
Change R to 11 and plot
Change R to 12 and plot
```

Now we've made a solid 8x3 block of wall material with 6 bytes. Another enhancement is that N = 0 means infinite, or at any rate, unlimited length: that is, the strip continues

37 "Memory-Augmenting Stream Of Numbers".

38 There are accommodations to get around the fact that C values need to exceed 31, and to allow S and F to be set.



The build tools currently choose this sequence of plots to lay out the Great Hall in MASON. The numbers on the arrows are the cost, in bytes, of making the next plot (the one which begins at the arrow's end) given that the registers start with the values of the previous one. Note that the right-hand ladder, platform and brazier are plotted as reflections of the left.

until it hits the room edge. Values of N of 64 or greater mean that a left-right reflection of the strip should be plotted, and in a reflected position on screen. Thus:

```
Change R to 13
Change C to 5
Change N to 4
Change T to "stairs down and right" and plot
Change N to 68 and plot
```

And this produces two symmetrically-placed staircases, one moving down to the right of column 5, the other moving down to the left of column 30. A surprising number of rooms have one or more architectural features which are symmetrically placed like this.

So, then, MASON potentially saves many bytes, but the code for a given design has to be ingeniously written to do it, making the fewest possible register changes. Our build tool therefore had to turn the ASCII-art drawing of a room into an optimal set of strips to achieve it, placed in an optimal sequence to minimise register changes between plots. This is a slight spin on the travelling salesman problem, and pre-computing the answer is by far the slowest part of the build process. On an Apple M2 Studio Ultra with 24 cores, which is clocked about 45,000 times faster than a

BBC Micro Model B, the process takes 8.25 seconds. On 1984 hardware, that would have been around five days.[39]

Moreover, the two registers S and F allowed new forms of plotting. S is the sprite number, and F holds flags which modify the sprite being plotted (to turn it upside down, or flip it left-right, or permute its colours). S typically held 255, a special value meaning "the standard sprite for this type" — thus, the ordinary wall sprite would be used for any walls, and so on — but could also be 254, "invisible", or else an arbitrary value, allowing anything to be plotted as anything. This, all at once, made the Castle much more textured.[40]

Roughly simultaneous with MASON was the development of a second virtual processor: MASH.[41] Just as MASON helped to make the rooms look a little different from each other in architecture, MASH gave their monsters different motions.

For example, the ghost in THE GREAT HALL now emerges from a skull and drifts erratically about, though in fact the course is pre-programmed. It's running this program:

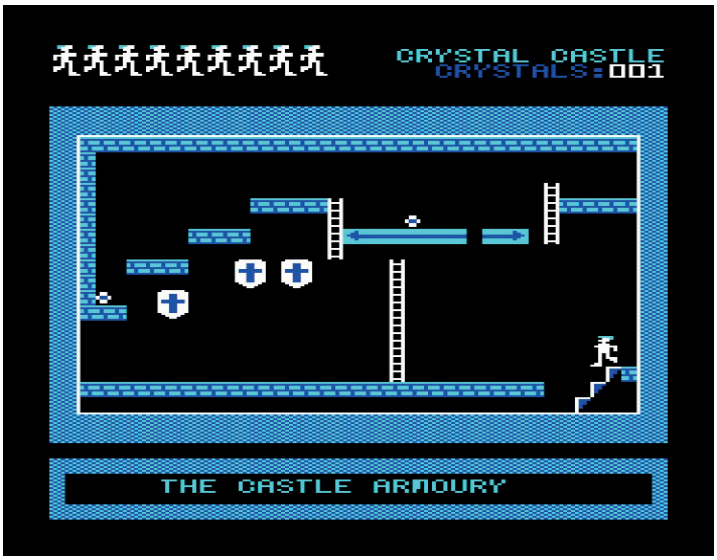
```
Monster Mb <Ghost> {
    WAIT RANDOM 1 TO 7
    MOVE TO D
    RIGHT-UP TO o
    RIGHT TO H
    WAIT RANDOM 1 TO 7
    UP TO k
    RIGHT-DOWN TO o
    RIGHT TO 2
    WAIT RANDOM 1 TO 8
    RIGHT-UP TO h
    REBOOT
}
```

---

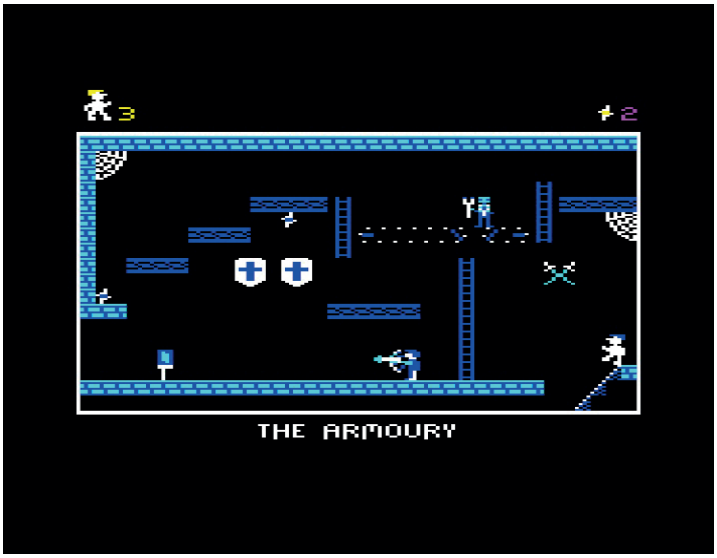
39 And even this vastly understates the difficulty, because the optimiser also requires more memory than even a 1980s mainframe could provide, and because the 6502 processor in a BBC Micro could only provide integer addition and subtraction of 8-bit values by way of arithmetic.

40 Though of course sprites cost memory, even textures.

41 Which was named after the title of Bobby Pickett's 1962 novelty single 'The Monster Mash', a Halloween-themed dance number. (It was a graveyard smash. It caught on in a flash.) We knew it, though, from the Bonzo Dog Doo-Dah Band cover version on their 1969 album *Tadpoles*, memorably voiced by Vivian Stanshall, who later went on to introduce the instruments in *Tubular Bells*.



1984



2024

Spot the differences... Change is everywhere: of the sprites, only the wall spritelet and the shield are the same. In 2024 there's scenery (the webs, the crossed swords) and animation (with archery going on, and the conveyor belts in motion). A crystal has moved, and so has a ladder, and a new platform has appeared. The surround is much lighter, and the status bar above the play area more graphical — to make room to display an object being carried, which wasn't a thing in 1984. Text is more stylised, and better spaced (note the M, especially). Even the room name is slightly altered. But this is recognisably still the same place.

This is a simple set of movements and pauses, but even so it's far more than 1984's engine could do. The ghost can move in eight directions, not four, and is not endlessly bouncing back and forth. In fact, a typical 1984 guard bouncing up and down endlessly between two fixed heights would be this program:

```
Monster Ma <Guard> {
    UP TO c
    DOWN TO g
}
```

And so on. MASH could also change a monster's sprite, which made some pleasing set-pieces possible. In December, for example, we added an archer firing an arrow at a target in the ARMOURY. All this was in service of visual story-telling, of a sort. The other ARMOURY monster is a guard trying to stand on duty, but who keeps being dragged away from his post by conveyor belts. It was very satisfying to be able to add these little looping stories with only a handful of bytes each. We would have loved that ability as teenagers in 1984, and having it now felt like being able to go back and do it all again.

MASH iterated rapidly as further features were added,[42] demanding more and more of the engine. Using a custom screen mode (i.e. programming the video chip directly to break out of the BBC Micro's standard Modes) trimmed away unwanted screen space and liberated a precious 2K, but it pretty well all went on further MASH features.

By September 2019, with MASH only a month old, programs were more than simple loops: they had interrupts, of a sort, called "events". For example:

```
Monster @harmless Mb <LeverUp> {
    ON COLLISION:
        IF D >= 1, STOP
        DO D = 1
        SPRITE <LeverDown>
        SOUND leverMSound
        SIGNAL Ma
}
```

This monster does nothing until the player collides with it. That would ordinarily result in the player's death, but because the monster is @harmless, we run the ON COLLISION

42 In 2024 we are using version 19, but versions 1 to 10 came and went within the first two days. The first mature version (if any of this can be called "mature") was 14.

interrupt instead. The lever is pulled into the down position, if it hasn't been pulled already, and makes a clunk. (It uses a variable D to store its state, D = 0 meaning the lever is in the up position, D = 1 in the down.) The lever in fact causes something else in the room to behave differently, and it does this by sending a "signal" to that other something (the monster "Ma"). There's no data in a signal, it's just a prod, but MASH code can also allow monsters to move or reboot each other.[43]

In effect, then, the Castle engine had become a simple time-sharing operating system, running the monsters in a room as tasks which could send signals to each other. The engine maintained the shared world-state, while the monster tasks looked after their own states, and could grab control with interrupts when certain things happened. Critically, these programs had to be *small*. The above lever occupies only 12 bytes: 3 bytes of header (initial position and sprite, and room for its variable D), 1 byte identifying the event ("ON COLLISION"), and 8 bytes for the five instructions in the program, three of which occupy two bytes and two of which one.

The game logic for the 2024 Castle ended up at just 1643 bytes of MASH, whereas the architecture took 4755 bytes of MASON, a roughly 1:3 proportion. A typical room then consumed around 170 bytes in all, of which about 16 might be data other than MASH and MASON, such as its name and colour scheme.[44]

By now, we were writing up proposal documents for new features, and keeping documentation for MASH, and finding its limitations. Because there were global game-state flags, something done in one room — a lever pulled, say — could already affect what happened in another. But only in a quite limited way. How about a key which the player could find and take from one room to another?

---

43 A "reboot" restores the monster to its original state when the room was entered. For the lever, it would be back in the up position, and would be drawn accordingly and with its D variable restored to 0.

44 Text is expensive. In 2024 the game contains around 620 bytes of it, stored as a stream of nybbles in a form of UTF-4 encoding with character codes arranged to place most frequently used letters (A, space, E, O, S, ...) first. We could find no form of Huffman coding which could be decoded with few enough 6502 instructions to make further savings beneficial.

We quickly realised that an object such as a key needed the same sort of facilities as a monster, and began to use the word "mobject" for the now-joint concept. A monster was a mobject found only in one room, which the player could not take: an object was a portable mobject.

Another priority was for game events to be able to change the layout of a room. For some time we experimented with an ever-more complicated MASH instruction devised by Graham called BOMB, because it essentially blew up a little clump of design cells, thus allowing doors and walls to be removed. Making BOMB more flexible needed more and more fiddly engine code, until we dropped it in favour of Toby's alternative solution, REVEAL. This was a much more elegant scheme in which MASH interacted pleasingly with MASON, activating MASON to add new strips to a room. So, now we could add walls and staircases, for example, not simply remove them.[45]

All of this allowed some pleasing puzzles to be added. There was much redesigning of the Castle to do, but we had 778 bytes free, and in October a round of optimisations of the engine and tools raised that further. The obvious next source of savings was from the sprites, which occupied nearly 3000 bytes. Could they be compressed? Many schemes were tried, but for once the answer was no. They were already high in information density, and in every scheme we looked at, the decompression code would be bulkier than the savings. Except for a cutesy trick to allow sprites to overlap in memory if the relevant pixel rows were coincidentally the same,[46] sprites are uncompressed to this day.

At any rate, by November we had 1751 bytes free. Times were good. It was a land of plenty. Many rooms could be made more deluxe, and those where we couldn't think of anything interesting we removed. We dropped the idea of having to have exactly 64 rooms and decided instead to have only the number of rooms we could make fun: it ended up at 61.

That bounty of spare memory wasn't all spent on room and puzzle design, though. The player's movements up and down stairs were made smoother, and the conveyor belts were animated for the first time. Envelope support for the sound effects allowed for subtler tings and crashes: those in effect were another virtual processor. An assembler for

---

45 Though it was a headache dealing with crystals which were hidden behind a "revelation", as these were called.

46 You wouldn't think this would happen often, but it saved 120 bytes, enough for a modest extra room.

these (MINIM) wasn't added until 2024, so sound effects had to programmed in hexadecimal, but still, they weren't long programs. Here for example, in MINIM code, is the sound effect for taking a crystal:

```
Sound crystal {
    Priority high
    Delay 40cs
    C#7 for 85cs
    Modulation { repeat: up 1 wait 5cs, up 1 wait 8cs,
up 1 wait 8cs, down 3 wait 1cs }
    Volume { to 15 wait 13cs, repeat: down 1 wait 5cs }
}
```

So it's a high note (a C-sharp in octave 7, near the top of the piano keyboard) with vibrato and a slow fade. Of course, everything comes at a cost: this effect occupies 23 bytes.[47] Yet a further round of memory savings became necessary, this time rewriting the engine to move as many state variables as possible into zero page.[48]

Enhancement to MASON gave it an ability to flood fill areas of a room with wall, or scenery. The crescent moon and the stars were added to the exterior scenes, clarifying for the first time that the game took place at night.

By now, it was Christmas 2019: the last normal one before the COVID pandemic. We'd missed the deadline for a 35th anniversary, but also, we couldn't quite convince ourselves that we were done. We were starting to have the guilty feeling that the story of the game was... opaque. Who was this guy? Why did he want the crystals? Why were they in the castle? Was it his castle? When does all this happen: in the Middle Ages? The far future? 1978? The three original authors mulled it all over as the family gathered, but didn't really get very far. Graham's wife Emily, one of the world's leading experts in the field of narrative design for games, looked on with a kindly expression.

We began to toy with a victory room which would in some way

---

47 Though envelopes for pitch and volume can be shared between sound effects, and MINIM automatically seeks savings like this.

48 The 6502 processor was chronically short of registers but compensated somewhat by making accesses to the memory locations 0x0000 to 0x00ff (page number 0, pages being 256 bytes long) take fewer bytes of code. This made them perfect locations for variables. Unfortunately, it also made them perfect locations for the operating system's variables, and some care was needed to avoid clashes.

explain what had just happened, but never really found anything satisfactory. Eventually, though not until 2020's deeply strange summer of quarantines and isolation, we made a more radical change: we removed the Mode 7 textual opening pages — expensive on memory, mostly for the text they contained — and replaced them with a prologue room, which was then just called ENTER THE CASTLE TO BEGIN.

2020 wasn't an enormously productive year for the Castle, but it had its moments. The most interesting idea, for a procedurally-generated cavern maze using pseudo-random numbers, came to nothing because experiments came out unsatisfying. But useful work was done enhancing MASH into super-MASH, SMASH.[49] Mobjects could now share "library programs", making it possible for, say, six monsters with the same complex behaviour to be more efficiently stored in memory. And monsters gained the ability to WANDER, that is, to move until they hit an obstacle, then turn.

The big development in the engine for 2020 was, however, behind the scenes. Months of preparation led in September to the final break with the operating system. When running, Crystal Castle now acts as its own OS for the BBC Micro, never calling the built-in MOS. It reads the keyboard itself, and programs the video and sound chips itself. It displays its own text, with its own fonts.[50]

For players, the most visible change came in October, with the birth of Mr Wriggles, the snake-familiar into which the player transfigures. This too was an attempt to make a story from the origin room, but it quickly became a way to add variation to the ways in which rooms had to be solved. Mr Wriggles has a smaller jump than the Magician, but doesn't die from long falls;[51] on the other hand he can't climb ladders. Being smaller allows him into crevices, and so on.

Design work picked up again around Christmas. The tools

---

49 Git commit name: "For MASH get SMASH". If this reference escapes you, you are either not British or weren't alive in the early 1980s. But if so, why are you reading this?

50 The gain from this is that RAM used by the built-in OS is now free for the game to use instead, and this includes a lot of precious zero-page space. But it's a demanding technique, and was hardly ever tried in the 1980s, *Exile* (1988) being an exception.

51 Snakes can survive surprisingly long falls, though it seems to be partly from being naturally springy on impact, and is not just because their terminal velocity is low.

were tidied up to allow for multiple castles to be developed: in effect, we now had a constructor kit for games using the now quite elaborate engine. A Happy New Year game demonstrated this, with the player walking all over a giant number 2021. MASH developed a little further, once again, to allow monsters to support the player: and so we had moving platforms for the first time. Mobjects were also allowed to be "trailing", which meant that they could leave a pixel trail behind them: this is how the spider can come down or up a strand of silk, for example. And, because we had noticed how many rooms involved monsters doing one thing until something happened and then doing another, MASH added the concept of a "secondary loop", an alternative program which a monster could switch itself to.

This surge of activity over New Year 2021 culminated, in February, with the ability to climb ladders. For the first time, the game had UP and DOWN keys, and the player (if in Magician form) could shimmy up or down. Ladder cells became different from platform cells: a ladder can support the player even at pixel heights in between cell rows, whereas a platform cannot. And so, for the first time, it was possible to come *down* a ladder; which made a lot more sense, but this, in combination with Mr Wriggles's ability to fall safely, forced us to look again at the design of every room.

Over the second summer of COVID-19, with fewer mandatory lockdowns but more self-isolation, a little more work went on between August and October. Monsters gained the ability to SEEK or FLEE each other (or the player), and could EXPIRE: that is, vanish from play for the rest of the game, not just until the room is re-entered. Memory was by now appallingly short; and, although we still had ideas for improvements, it seemed improbable that we could find the space for them.[52] As September 2021 came to an end we put the game aside, still not really having worked out a satisfactory plot for it, and gave it a rest for a while.

---

52 A sign of the extreme nature of our optimisations by this point is that the state of which crystals have been taken was moved into screen memory, where it uses alternate bits of the pixels in the apparently blank spacing between the room and its title: because the engine, using a carefully timed interrupt, changes the screen palette to black, white, black, and white there, the player doesn't see the data bits, since the relevant 0s and 1s only change pixel colours from 0 to 2 or from 1 to 3.

vii) 2024: The 40th Anniversary Edition

For quite a long while, as it turned out, but in October 2024 we realised that the fortieth anniversary year would soon run out: wasn't it time, finally, to finish?

Throughout that autumn, the bytes free count rarely exceeded 100, and the build tools would typically report fewer:

```
    Total RAM in BBC Micro Model B          | 32768 |
    - Engine code and variables             | 14675 |
    - Screen memory                         |  6992 |
    - Font                                  |   160 |
    - Music                                 |   495 |
    - 154 sprites of 376 spritelets         |  2833 |
    - 61 room(s)                            |  7601 |
      - of which MASON code                 | (4809)|
      - MASH code                           | (1611)|
    = Free space left                       |    12 |
```

Every addition to the Castle meant a subtraction somewhere else, and sometimes those were painful. The low-hanging fruit in terms of savings (compressing the data, shortening the engine code) had long, long gone.

But we did manage one major change to the game's mechanics. In the 1980s, the player had nine lives, and when they were gone, they were gone. In December 2019 we had added "medkits": these were monsters, implemented in MASH, which when touched would vanish and give the player another two lives. At one point the Castle also had a Fountain of Youth, restoring all the original number of lives, which — because the game really is quite tricky — we had raised to 16. But we started to feel that this wasn't working. Better was to start the player with a modest number of lives, the traditional arcade-game 3, and then steadily award extra ones: and so we had the idea of giving an extra life for every third crystal taken.[53] What made this a happier idea was that the crystals were now, according to the story we were finally working out, the keys to immortality. So, it made sense that even getting a small number of them gives life.

And with that, we were done. What we thought would be the final remaining bytes were used on scenery, with the crossed swords and the hanging paintings being one last motif, and THE HAUNTING[54] being the last room to have its ar-

---

53 Implementing this required the engine to allow for a global MASH program to run whenever a crystal is taken.

54 Formerly THE HAUNTED GUEST BEDROOM. There was also,

chitecture spruced up. But then we saved some more bytes, and spent them on... and so on. The positively last denizens added to the Castle were the snails. (If you even find them, you've done well.) The final released binary had 0 bytes free: to the best of our knowledge, we used every last byte in the BBC Micro RAM.

- * - * - * -

Of course, it was one thing to have a theoretically complete game, and another to test it. We quickly realised that we needed a way to end-to-end test the solution.[55] This led to one final M, MEANDER. Toby devised an ingenious recording and playback mechanism for keystrokes: for testing purposes only, the engine can call an extension piece of code in sideways RAM,[56] which can either record the keys being pressed (also into sideways RAM), or simulate keys being pressed by going back through such a recording. That made it possible to teach the testing harness how to complete a given room. But great long runs of binary data of keypresses were difficult to stitch together, or pick apart, and so MEANDER was born: this provided a textual way to describe test scripts. For example:

```
ENTER THEMAGICIANCASTDOWN AT (B, q) AS SNAKE
→29 JUMP→4 →42 JUMP→3 →23
WAIT38
→26
```

That made it bearable to put test scripts together, and the first complete end-to-end recording of a victorious play-

---

for many years, THE JESTER'S BEDCHAMBER (with a jester's hat as monster) in which nothing worked as it should, conveyor belts went backwards, you could not die, and such. But we ended up taking it out because, as a joke, it relied on the fact that the Castle behaved exactly the same everywhere. Once that wasn't true, it wasn't a joke.

55 If there even *was* a solution: we weren't sure that our puzzles could all be completed in sequence.

56 Sideways RAM was a hardware tweak to the BBC Micro, often bought as an expansion package in the 1980s and now offered by emulators as well, which placed 16K of extra RAM into the address space 0x8000 to 0xBFFF, where the BASIC ROM normally lived. As a matter of principle, CRYSTAL CASTLE runs on a stock BBC Micro Model B without such a board: we used Sideways RAM *only* for testing. But oh, we were tempted. Sixteen thousand more bytes to adorn our Castle, after months of living with 10 bytes free?

through was made on 19 December 2024. As guidance for any future speed-runners, we think a good player can complete the Castle in around 25 minutes.

A thousand years have come and gone
But time has passed me by
Stars stopped in the sky
Frozen in an everlasting view

— Rush, 'Xanadu' (1977)