Rom Hacking and Legibility

In Tuesday’s workshop, we explored (and hacked) a variety of NES cartridges. We traced Super Mario Bros. from its material allocation on the PRG and CHR banks of its cartridge to its playable, modifiable instantiation on a windows emulator.

The second half of the workshop offered a rich set of code reading and code producing tools to break, tweak and explore our super mario bros. roms. In this short probe, I’m going to think through the affordances of writing with these tools.

We started our workship with this video of code-bending. The asemic substitutions of ram glitching contort the game beyond human legibility, and yet, the game still runs. to watch them slip so casually out of domains of human understanding inspires a sense of both awe and dread (a glimpse of Gumbrecht’s radical perspective of contingency) — but also a desire to understand them.

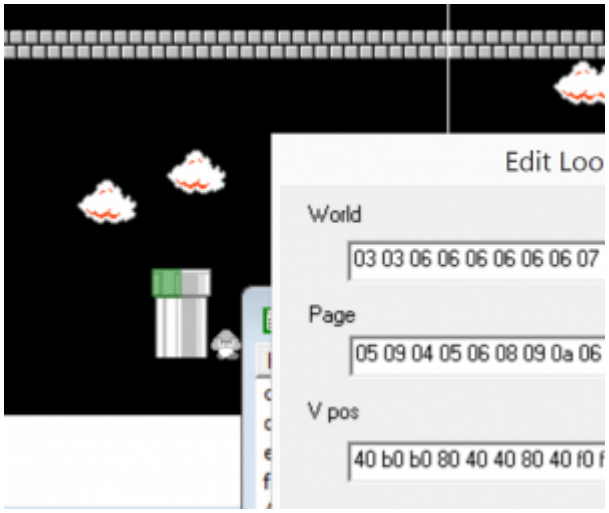
Professor Lemieux offered a couple of approaches unraveling the game’s internal processes. First, he demonstrated ram modification with FCEUX’s hex editor. He listed all the addresses with a particular known value, and then isolated an address by performing an action that would alter that value.

Moreover, he showed us how to freeze an address to make its operation visible, and how to use a breakpoint to find out how the game’s logic is structured. These methods function as a close reading of the game’s processes, and in line with Sterne’s advocacy of a sense of playfulness, an opportunity to break them.

To chart out the technical possibility space of these tools, I began with these small inquiries
— can I let mario jump while he’s still in the air?
— can I remove the floor?

These inquiries focalized my exploration of into two modes:
– (textual) a text editor to modify the pre-assembled game code
– (visual) a map editor to modify the level

The textual approach allowed me to play with a number of physics properties in short order.(they were all grouped together in the pre-assembler). Once mario could jump in midair, I could tweak the movement physics (gravity, jump velocity). This kind of fine tweaking is well suited to an efficient testing loop. “Ah, this feels too floaty”, “The velocity is too spiky”. The ability to iterate quickly over changes was important to me, so I spent some time developing a workflow to do it efficiently (organizing a shortcut, arranging windows)



After playing with the physics for a while, I moved on to the map editor.
This had a different set of challenges:
The map editor’s GUI required a kind of prodding movement through its options and sub-menus.
Finding anything useful was happenstance. There’s something frustrating about the nested structure of dropdown menus that makes it incredibly frustrating to read. Yet, there was still a strong sense of play in exploring the arrays of obscure options.
(and it wasn’t without its good luck – there proved to be a

simple option to remove the floor).

Play was essential to the success of the method, and going forward this week, one of the questions I’m interested in is the following:
If we foreground play as in our methods of inquiry, do we occlude those artifacts that resist play?