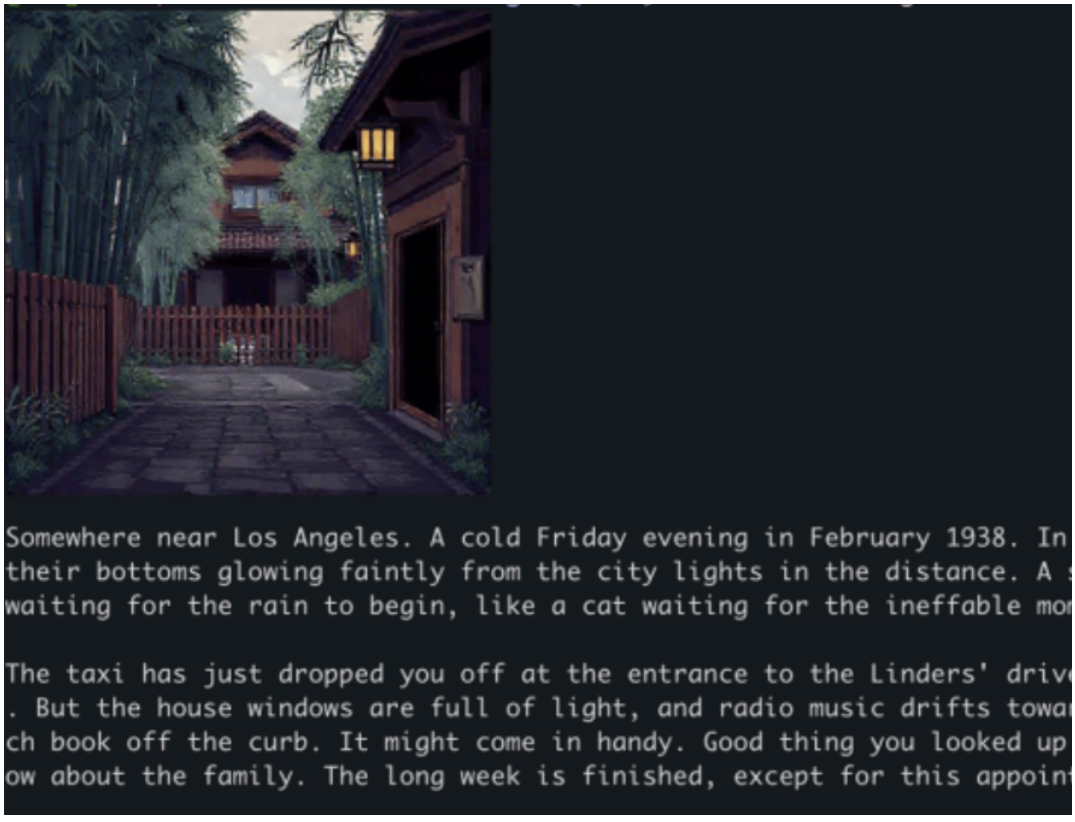
Case Study: TSZM / TSCDN
Engaged: Christopher Shepherd for BE&L (Open Source Community Grant)
Position: Sole Author

Summary:

The first-ever Infocom Z-Machine implementation written from scratch in Typescript. Compatible with z3 games, Quetzal-format savegames, with frontends for console (node) and browser (React). Additionally, an image CDN that makes use of two language models: Qwen, to gather context and write a meaningful JSON image prompt for illustration of the most recent game move, and FLUX-1-Schnell, to execute the image prompt. Illustrations are presented via inline Sixel graphics on console, or a PNG-based React component in the browser.



Process:

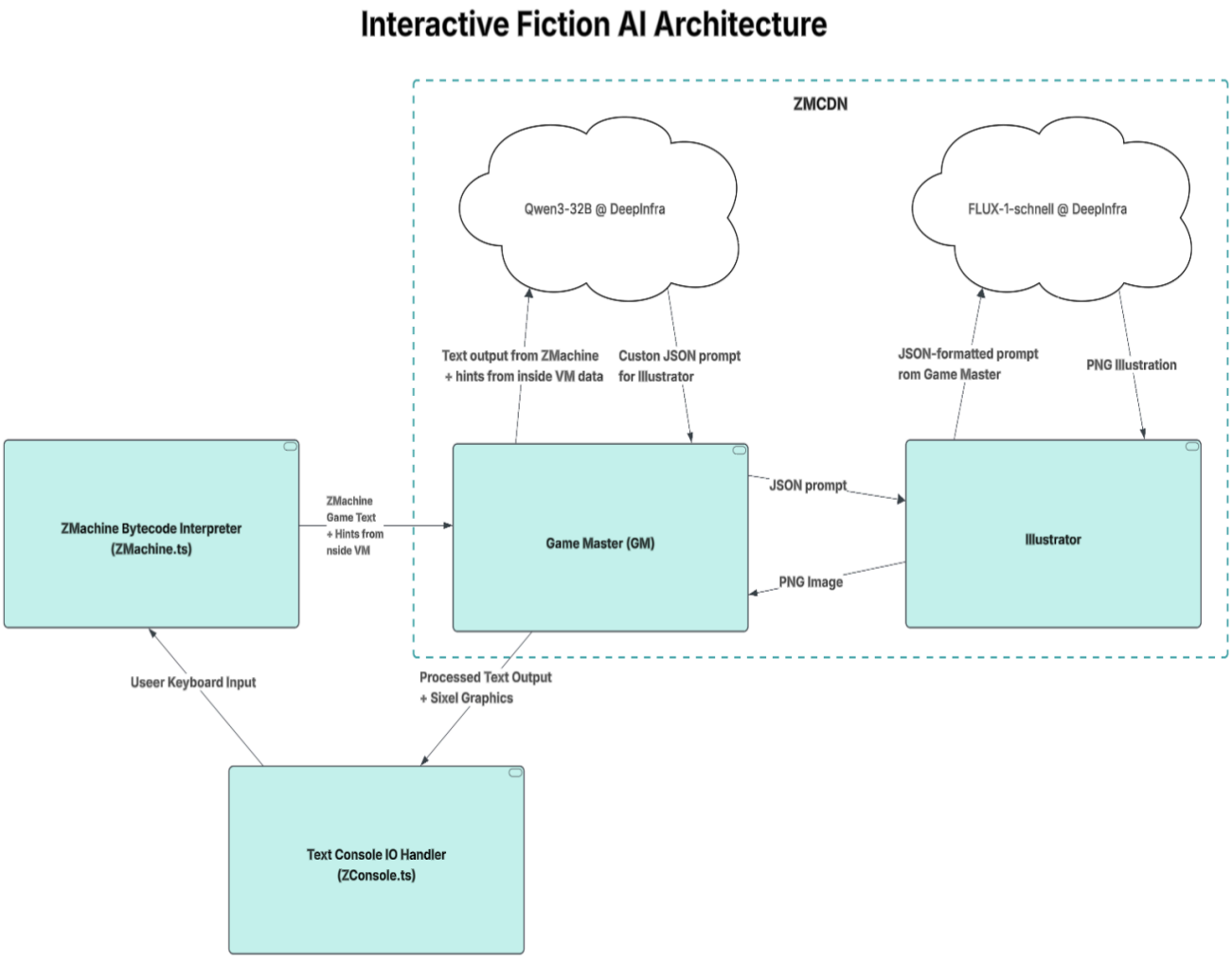
Initial work involved decoding of the Z-Machine game header, followed by opcode decoding logic (which unsurprisingly turned out to be the hot path of the engine). Opcode dispatch initially took the form of a large 'case' statement, as in many other emulators, but was refined into an array of objects indicating the class of instruction, how many additional bytes to ingest from the code path, and so forth.

Very early on (almost at the very start) in the development process, a Typescript Interface class was introduced to enforce a contract for different input/output technologies, as it was foreseen that both console and browser modalities would be supported. This decision was greatly validated, both when the React frontend was added, and when the Z-Machine engine was moved into an npmjs module.

ZMCDN was the real innovation in this project, as someone (hello Jay) had asked what it could look like if someone were to add AI-based on-the-fly illustrations to Infocom games. Another Typescript Interface contract enforces the ZMCDN REST API, which involves sending the past 8 in-game inputs and outputs to Qwen to establish context. At this point, Qwen uses this input, along with its Master Prompt, to return a JSON image prompt suitable for FLUX-1-schnell. FLUX-1-schnell then generates the image, which is cached for the given game and room. As a result, although the language models are hosted offsite at DeepInfra, the games cost pennies per month to serve a few dozen users.

Once the initial logic was validated with jest test cases, opcodes were added, one at a time, until 100% z3 compatibility was achieved. Significant progress has been made towards z4 compatibility, as work continues on this project, although it's no longer at the forefront of Christopher's queue. Although development was not test-driven, tests were a vital part of the opcode and logic development process, both to validate new opcodes and prevent regressions. This project would not be possible without excellent test coverage.

Result:



The above architecture emerged, based on aforementioned interface contracts. This image clearly shows the integration relationship between the various components.

This was really well-received by the community, to the point that BE&L received a community endowment for open source retrogaming-related contributions. It is currently funding the development of a well-loved AAA title that never shipped for the Apple IIGS, despite shipping for the Atari ST, Amiga, and IBM AT. It also involves a bytecode interpreter, although it isn't as complex as Z-Machine.

Live Demo:

https://cshepherd.fr/tszm-react/

Source Code Repositories:

https://github.com/cshepherd/tszmachine (npmjs module containing the Z-Machine engine)
https://github.com/cshepherd/tszm (console-based frontend)
https://github.com/cshepherd/tszm-react (React-based browser frontend)