Planet Explorer

Final Presentation
for the
UWW 310 semester project

Project Definition

Planet Explorer

This project aims to create an ASCII-based client application for providing interactive, textual representations of planetary terrain, vegetation, and climate by means of controlling an in-application avatar to simulate traversal of land, sea, or ice.

If successful, the Planetary Explorer code will be integrated into Hexagram30, the author's existing suite of world-generating libraries and tools for distributed-system-style game development and deployments.

Goals

Primary Goals

- Learn the Rust programming language.
- Learn and use a toolkit for developing ASCII-based games in Rust.
- Learn to use a Rust configuration library.
- Learn about a component system for managing state and data exchange.
- Create a text-based, ASCII-character interface based upon classic rogue-like games.
- Modify a standard rogue-like to display above-ground, outdoor areas.

Secondary Goals

- Adapt the game textual interface to display world metadata.
- Select a storage mechanism and library for saving and retrieving game data.
- Successfully store and restore previously saved game data.
- Update the game to read data from external sources.

Stretch Goals

- Display previously-generated terrain data in-game as a user traverses that land.
- Display previously-generated climate and biome data as well.
- Develop in-game time tracking.
- Display in-game time and date.
- Develop tiling mechanism for planetary areas.
- Support whole-planet exploration via tiling mechanisms.
- Provide in-game weather for the user's current location.
- Add Telnet support for connecting to existing servers.

Tools

Programming Language

Rust is a new systems programming language, created in 2010 for applications that need to be run safely in distributed environments, without creating the vulnerabilities that are so common in modern software. Since 2016, Rust has been voted as the #1 “most loved” programming language in the Stack Overflow language survey.

Rust covers areas of programming to which I have had little exposure, and offers the ability to create highly-optimized, fast-running, memory-light applications. All of these are of great personal and professional interest to me.

The primary goal of this project is to gain a beginner's level skill in programming applications with Rust.

Application Framework

The second most important tool to be learned for this project will be a framework for creating scalable, performant, and fully-functional ASCII-based (and even graphics-based) rogue-like games. RLTK, or “rogue-like toolkit,” is just such a framework, and when compared to its peers, offers a winning combination of the best API, the most powerful set of features, and the best documentation.

In-depth knowledge of this framework will be useful in future projects were a creative approach to client applications is desired and/or appropriate for the given problem domain.

Libraries

The last set of tools to be learned are the following libraries; they are of particular interest for since they would be a powerful addition to any Rust project, not just this one:

- Specs - an Entity-Component System written in Rust, used for connecting and supporting communication between different, independent software components (used by RLTK)
- config-rs - a layered configuration system for Rust applications, supporting not only multiple file formats, but also environment variable overrides
- One of the following data storage mechanisms:
    - Rustbreak - a self-contained file-database library
    - sled - an embedded Rust database
    - sqlite - Rust bindings for the famous tiny database

Roadmap

Project Phases

Phase 1 - Proof of Concept

- Identity framework candidates
- Assess available documentation and code examples
- Create an initial rogue-like game

Project Phases

Phase 2 - Develop Key Application Components

- Build more involved world environments (maps) with better land area visualizations.
- Begin layout for textual information / feedback to user.
- Make application fully configurable, allowing non-coders to change worlds.

The above depend upon Phase 1.

Project Phases

Phase 3 - Interactive Client Application

- Diverge from standard rogue-likes to provide non-cavern, above ground areas.
- Read altitude, climate, and biome data from generated worlds.
- Develop mechanisms for generative, procedural vegetative land-cover.
- Support saving generated content to disk ("remembering" visited areas).

The above may be done in any order.

Self-Assessment

Goals

Primary Goals

All of the primary goals of the project were met, though the last two items didn't meet with the full expectations:

- Create a text-based, ASCII-character interface based upon classic rogue-like games.
- Modify a standard rogue-like to display above-ground, outdoor areas.

The full interface as I had envisioned it was only partially realized: there was a text component to the UI, but it didn't have all the data I desired.

I was able to display above-ground areas, but only as a last-minute work-around; again, not entirely as envisioned.

Secondary Goals

This was a mixed bag of success:

- ❌ Adapt the game textual interface to display world metadata.
- ✔️ Select a storage mechanism and library for saving and retrieving game data.
- ✔️ Successfully store and restore previously saved game data.
- ⚠️ Update the game to read data from external sources.

For reasons I will discuss in the “Problems” section, I never got to the stage of displaying world metadata (elevation, biomes, average precipitation in a given area, etc.). The last item was technically accomplished (this is how saved data is restored), but not all the external sources I had hoped for were integrated.

Stretch Goals

For the most part, I didn't have enough time to accomplish these; those I was able to work on were:

- ⚠️ Display previously-generated climate and biome data as well.
- ✔️ Develop tiling mechanism for planetary areas.

One was fully completed, one partially so. This part of the project was interesting, as it was part of a last-minute fall-back plan.

Assessment

All things considered, I feel that the goals were met with success, sometimes even great success. It was a good feeling to have some of the stretch goals checked off as well!

Tool Learning

Assessment: 100% Win

I was able to learn all the new tools I had set out to, most important of which was the new programming language. In addition, I had the opportunity to *create* several new tools using Rust!

The application framework is brilliant, and in discussions with its author, I discovered new aspects of it to explore (e.g., replacing one of the underlying libraries with a higher-performance one).

I learned to use *many more* libraries that what I had set out to, but didn't end up needing one for storage (the SPECS library was sufficient for most of it, and for the other anticipated need, I did something clever instead 🤓).

Phases

Smooth Sailing ... at first

Phase 1 and most of Phase 2 were quite successful. I ran into a spot of bother towards the end of the second exploration due to a fairly arcane usage I was attempting in Rust. Interestingly, I later learned that the author of the tutorial ran into something similar and had asked the community for help in sorting it out -- no one has figured it out yet, so I'm not alone!

In order to get around this, I decided to put my code on hold, and use the completed tutorial created by the author. Updating this to use the infrastructure I had put in place, however, was very time-consuming (easy but very tedious). I was running out of time, so I put that part on hold -- this was the reason I didn't finished the text-rendering of biomes.

Last-mile Work-arounds

Fortunately, due to the design of the project, I was able to work on the remaining part of the project (and some stretch goals!) in parallel, while the other work was on hold. This included creating another new library called 'image-data' for pulling biome information out of map images (like the one at the beginning of this presentation).

This went amazingly well: the whole library was created in a single day, and utilized all the Rust knowledge I'd gained so far, including areas of the code where I noticed significant improvements in idiomatic use of the language.

Then, I created yet a *new* project that utilized this new 'image-data' library. Its focus is simulated ecologies, and will ultimately be what the game uses for things like land cover.

Assessment

In each phase of the roadmap, I stuck to the plan; where obstacles were insurmountable in the given time, a quick pivot allowed for progress to continue. Special thanks to all that life experience!

A final victory worth mentioning is in the new ‘eco’ project, I devised a way of not needing to store content in a database! Any new areas explored in-game where content like land cover/vegetation are created can be uniquely identified by area: this info can be reduced to a number, which can then be used as a random seed. That seed will cause the random generator to *always* create the content exactly the same for that particular area.

These are very fast operations using small bits of data at a time, and thus is actually more performant to generate on-the-fly than to store in a database 😄

Creations

Primary Project

The Game

The code created for this project was originally kept here:

- https://github.com/oubiwann/tech-reflects

However, by the end of phase 2, there was so much good code in there, that I decided to integrate it into the suite of Hexagram30 projects, here:

- https://github.com/hexagram30/client

I am really looking forward to continuing to hack on this project!

Spin-Off Project

The Surprise

The difficulties mentioned previously resulted in an adaptation of work. This was necessarily decoupled from the game (a generally good practice, in software development!), and in just 16 hours, this evolved into a new, dedicated Hexagram30 project!

- https://github.com/hexagram30/eco

New Libraries

Low-Level

To accomplish various tasks, I used a great deal of other Rust libraries -- that saved TONS of time. However, much of what I needed was quite unique, especially in a new programming language. As such, there was some tooling I needed to create for myself:

- Using images as a data store: https://github.com/oxur/image-data
- Customized, easy-to-read debug output logging: https://github.com/oxur/twyg
- In-app signal-handling: https://github.com/oxur/signal-msg

The Future

Deeper Integration with Hexagram30

This semester project spawned two major projects that are already part of a larger effort in which I have been involved for years -- this will undoubtedly continue.

In particular, I've been tracking tasks (and completing them!) here:

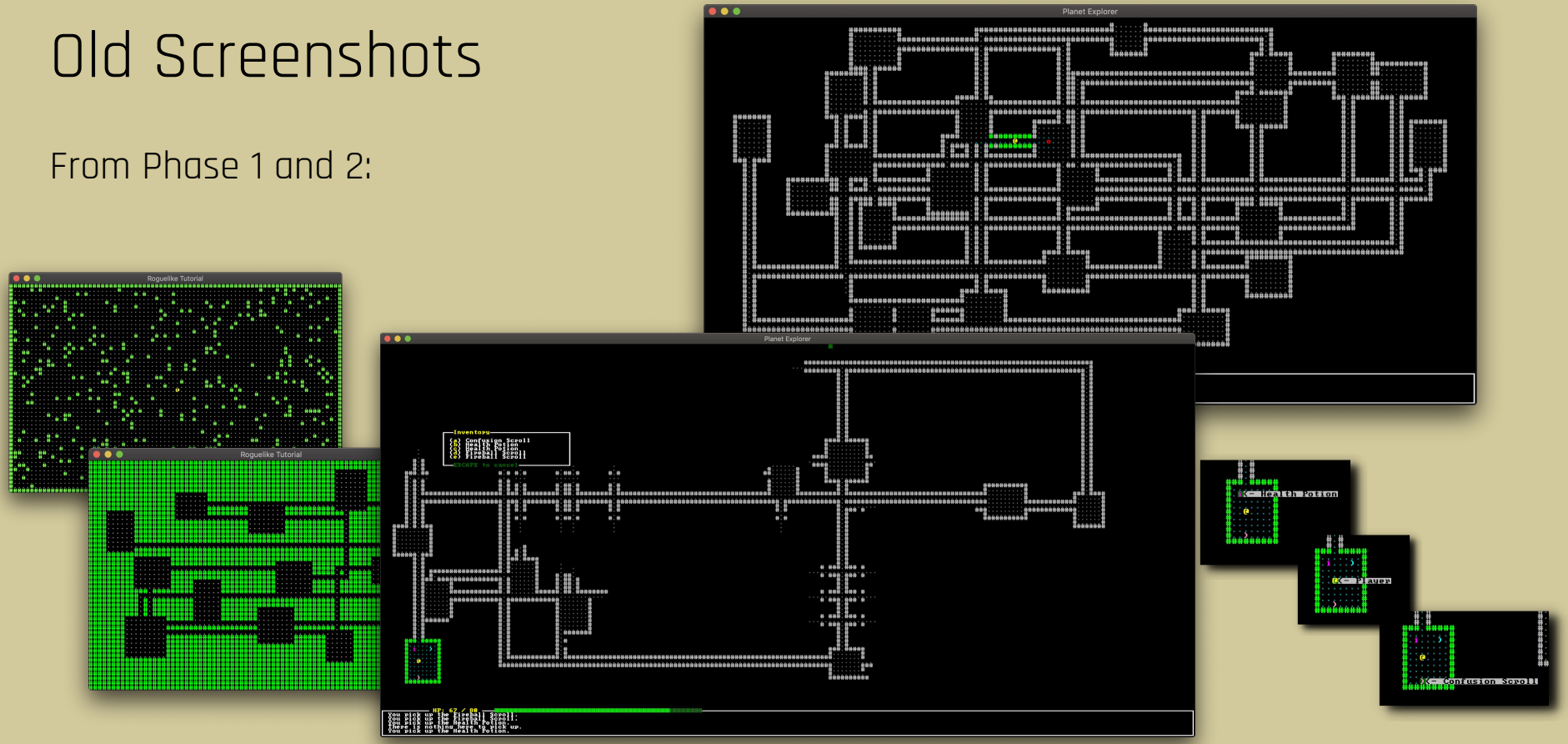
- https://github.com/hexagram30/client/issues
- https://github.com/hexagram30/eco/issues
- https://github.com/oxur/image-data/issues

I am *especially* excited about the work in the eco project!

Eye Candy

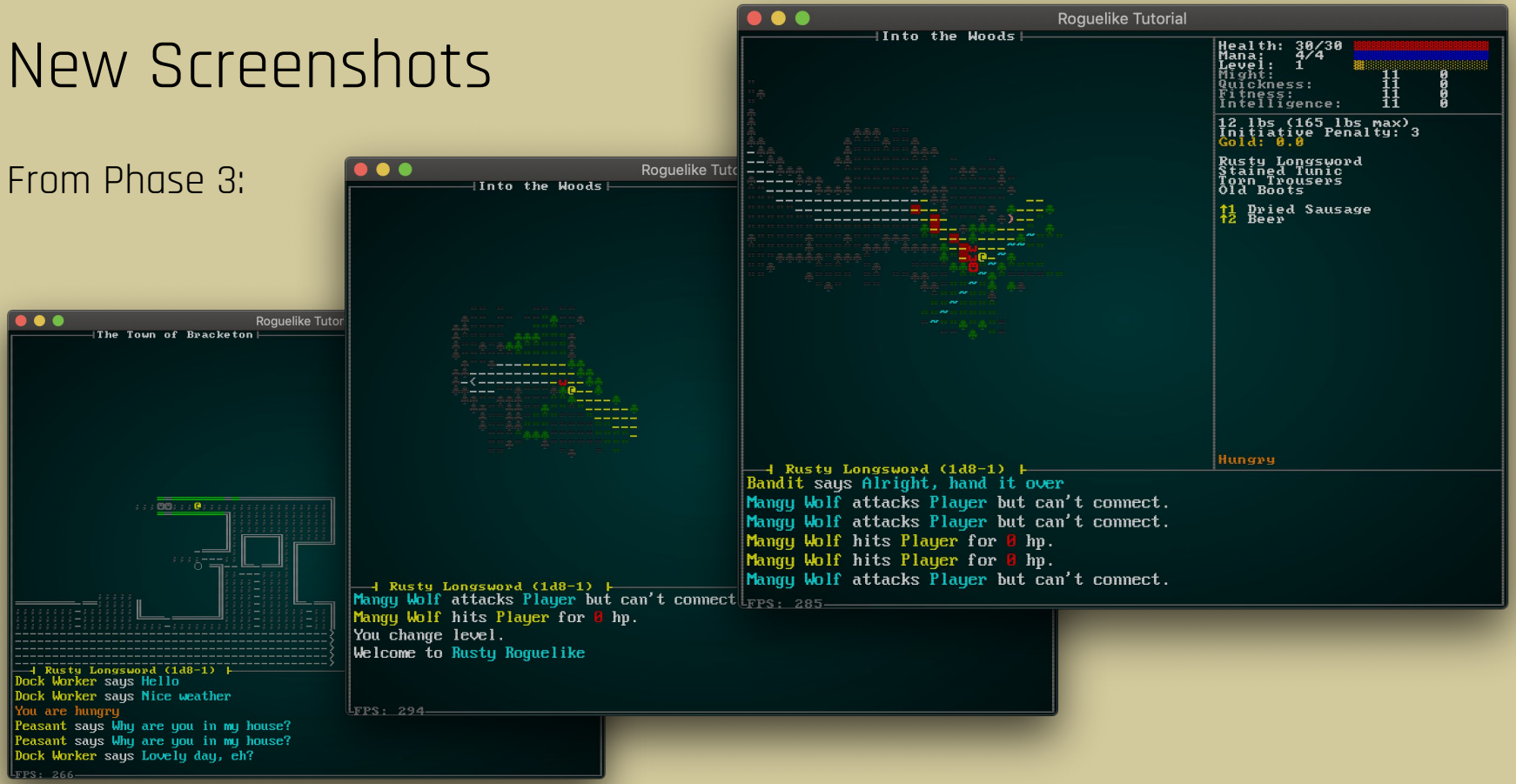
Old Screenshots

From Phase 1 and 2:



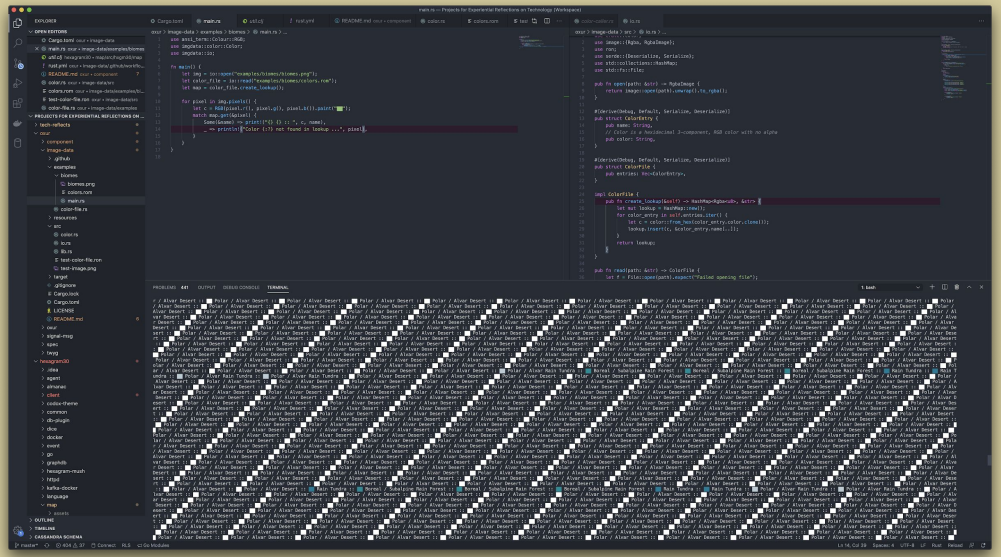
New Screenshots

From Phase 3:



Library Screenshots - image-data/eco

Examining the color-coded biomes in millions of pixels (from the south polar region):



Library Screenshots - image-data/eco

Examining the color-coded biomes in millions of pixels (from a temperate region):