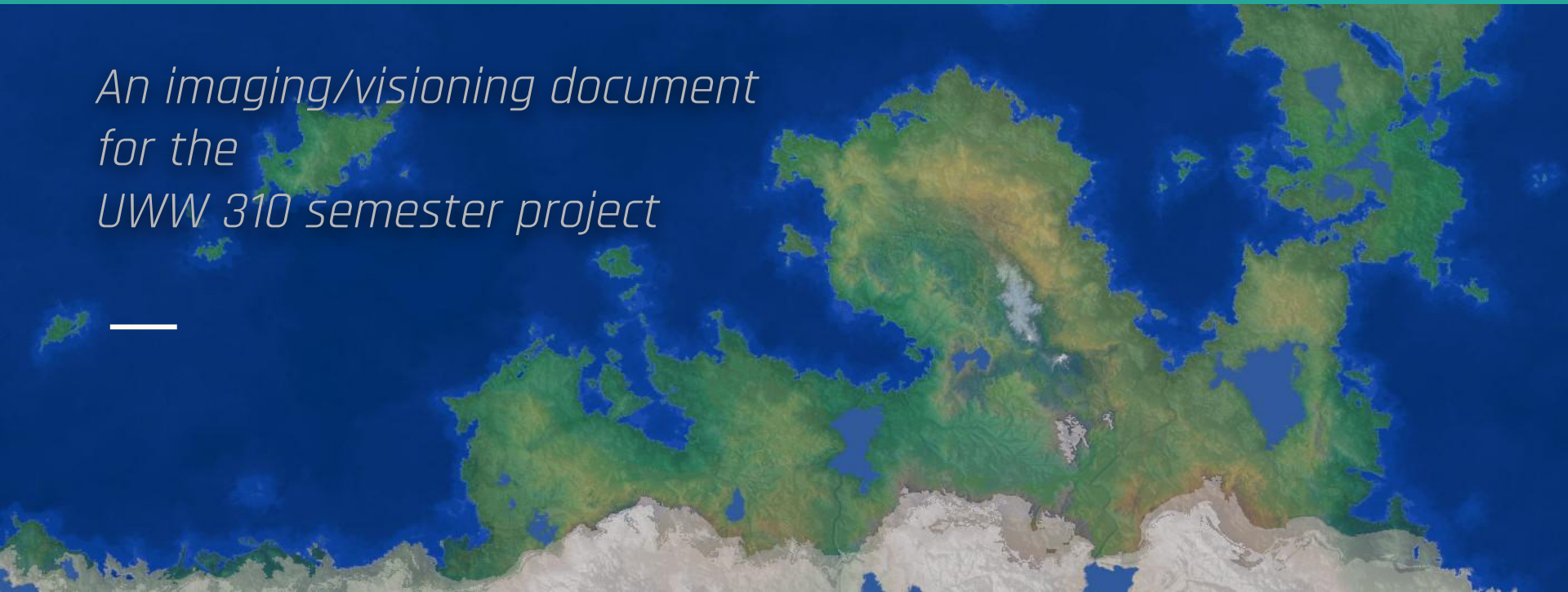
Planet Explorer



An imaging/visioning document
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.

Metrics for Success

Project Assessment

Most of the primary, secondary, and stretch goals for this project are explicit and easily measured via the software features and/or source code commits in git/Github. Perhaps the most difficult to assess is the first of the primary goals: learning the language itself.

Regardless of actual progress in each sub-goal, a key indicator of success will be that by the end of the project, I have the ability to create a full, complete application with the desired features *at some point in the future.* Significant progress toward both establishing that such a thing is possible, and implementing features that help attain that larger goal are metrics for success.