SE 3XA3: Requirements Specification
Rogue Reborn

Group #6, Team Rogue++

Ian Prins	prinsij
Mikhail Andrenkov	andrem5
Or Almog	almogo

Due Friday, October 7th, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
09/28/16	1.0	initial setup
10/02/16	1.0	Continued setup
10/07/16	1.0	Fleshing out section 1
10/07/16	1.1	Functional reqs + risks

This document describes the requirements for the Rogue++ project The template for the Software Requirements Specification (SRS) is a subset of the Volere template (Robertson and Robertson, 2012). No further modifications have been made from the blank project template given.

1 Project Drivers

1.1 The Purpose of the Project

The goal of the project is to produce a reimplementation of the original Rogue computer game, originally developed by Michael Toy, Glenn Wichman, and Ken Arnold in 1980. The gameplay of the reimplementation should mimic that of the original whenever possible. The objective of the rewrite is to produce a copy in a modern language, using modern design principles, with superior documentation and a full test suite. The original Rogue is of historical interest as it forms the foundation and is the namesake of the roguelike genre of games, typified by their randomized environments, difficulty, and permadeath features. The motivation for this project is the poor condition of the original source code. The original source was not written with readability in mind, and designed for extremely low-performance systems who required some unusual design patterns. The version of C in which it was written is very old, which hinders compilation or feature extension. The intended audience for this document is the stakeholders of this project, especially Dr Smith and the 3XA3 TAs.

1.2 The Stakeholders

1.2.1 The Client

The client of the project is Dr Spencer Smith. Dr Smith commissioned the project and will be overseeing its production. Dr Smith provides the specifications for this document, as well as other aspects of the project, including the test suite, and all documentation. In addition he will be evaluating the final product.

1.2.2 The Customers

The project customers are the players of the game. It is expected that this will consist primarily of players of the original, as well as players and developers of later roguelike games. The roguelike community has a strong open-source tradition, so a modern, well-documented Rogue could be a valuable starting point or inspiration for projects by other teams.

1.2.3 Other Stakeholders

Other stakeholders include playtesters of the game, as well as the 3XA3 TAs. Playtesters of the game will be recruited to play the game, and therefore have stake in the success of the project. The 3XA3 TAs will be evaluating the success of the project, as well as providing feedback and guiding the project while it is still in development.

1.3 Mandated Constraints

As a constraint imposed by the project client, there are a number of deadlines for the project throughout its development. In particular, the final demonstration of functionality will be on november the 30th, and the final draft of the project documentation must be produced by the 8th of december. The goal of replicating the gameplay of the original without significant change restricts the platforms for which the project can be developed. In particular, the interface for the original is extremely ill-suited to touch-input environments such as phones and tablets.

1.4 Naming Conventions and Terminology

Listed below are a number of video game and/or roguelike specific terms used in this document.

- Rogue: Both the name of the 1980 computer game, and the a reference to the player character (we will always use the term player character).
- Roguelike: A genre of games similar to Rogue. Membership in the roguelike genre is largely determined by the presence or absence of permadeath, but many games feature many more similarities.

- Permadeath: A feature of roguelikes where the game is restarted from the beginning upon character death.
- Hitpoints: A positive integer value that measures the health of a character (more is healthier).
- Item Identification: A common feature of roguelikes where items are scrabbled at the beginning of a game, with the player not knowing which corresponding to which effect. Certain effects or simply using these items can identify items. For example, a blue potions may be potions of healing in one game, but in the next they could be sleeping gas. Item identification also refers to determining whether a given item is cursed.
- Cursed equipment: Equipment that once used reveal itself to be harmful to their user and difficult to remove.
- Dungeon: Consisting of a stack of 30 floors, the dungeon forms the game world in Rogue.
- Gold: Gold coins can be found throughout the dungeon, the number of gold coins collected is the primary basis for the player's score.
- Level: Can refer to a floor of the dungeon, or to the player character's experience level, which determines their hitpoints.
- Experience: Experience is gained by defeating monsters, and sufficient quantities will cause the player character to level up.
- Searching: Certain features of the dungeon, such as traps and hidden doors are not immediately visible to the player. The player character can explicitly search their immediate surroundings for such features.

1.5 Relevant Facts and Assumptions

It is assumed users will be utilizing the product in a 64 bit Linux environment, with a keyboard and monitor of at least [INSERT DIMENSIONS]. Users are assumed to be at least moderately familiar with the original, no extra material describing how to play the game is planned to be produced.

User characteristics should go under assumptions. [DELETE]

2 Functional Requirements

2.1 The Scope of the Work and the Product

Not sure about this section

2.1.1 The Context of the Work

2.1.2 Work Partitioning

2.1.3 Individual Product Use Cases

2.2 Functional Requirements

This section will specify the functional requirements of the Rogue++ project. They are numerous, scattered, and interdependent, therefore an attempt shall be made to organize them into cascading, logical segments.

2.2.1 Basic mechanics

- The player should be able to start a new game
- The player should be able to save the current game by name
- The player should be able to load previous games by name
- The player should be able to quit the game
- The player must always begin with the default level 1 hero
- The player must always see their hero's statistics
- The game must wait until the user takes an action to manipulate the environment

2.2.2 Interaction

- The player should be able to view detailed information about:
  - The hero
  - The surrounding environment
- The player should be able to pass the turn

- The player should be able to walk around
- The player should be able to open and close doors

2.2.3 The Dungeon

- The player must begin at the dungeon's first level
- The game must generate each dungeon level one at a time
- Each level must have a downwards staircase
- Every level must generate rooms, corridors, monsters, treasure, and traps
- The player must be able to see in a 3x3 square centered on the hero
- The player must be able to see the entire room the hero is in, if the hero is in a room
- The player should see the outline of dungeon areas previously explored
- The player should be able to search for hidden doors
- The player should not be able to see hidden doors without explicitly searching for them

2.2.4 Equipment

- The game should maintain an inventory of player items
- The player should be able to view the inventory
- The game should limit the player's inventory based on the weight of its contents
- The player should be able to add, drop, use, hold, and remove objects from the inventory
- Scrolls, rings, and wands should have meaningless names until identified
- The player should be able to identify items
- The player should not be able to remove cursed items

2.2.5 Combat

- Each monster must have its own statistics
- Each monster must calculate a plan of action
- Monsters must only attack the player, not other monsters

3 Non-functional Requirements

3.1 Look and Feel Requirements

3.2 Usability and Humanity Requirements

3.3 Performance Requirements

3.4 Operational and Environmental Requirements

3.5 Maintainability and Support Requirements

3.6 Security Requirements

3.7 Cultural Requirements

3.8 Legal Requirements

3.9 Health and Safety Requirements

This section is not in the original Volere template, but health and safety are issues that should be considered for every engineering project.

4 Project Issues

4.1 Open Issues

4.2 Off-the-Shelf Solutions

4.3 New Problems

4.4 Tasks

4.5 Migration to the New Product

4.6 Risks

- **Computer Usage Risks** - There are several risks associated with computer usage. This is often a subject matter that is discussed thoroughly in an office environment, where computers see frequent, daily usage.
  - When using a computer, there is an ergonomic risk involved. Improper usage of the computer can lead to aches in various parts of the body, including back, neck, hands, and chest.
  - There is also a significant risk of eye aches, along with other vision problems.
  - Repetitive motion is another factor that could cause discomfort when using a computer.
- **Offensive Content** - The game draws heavily from fantasy, involving themes of violence, fear, and witchcraft. While these elements are only displayed in a textual context, certain cultures and societies may find such elements offensive or disturbing.
- **Anger** - The game is not easy. Frustration could easily overcome the player, especially when he/she has progressed far into the game. Anger management issues are widespread, and evidence of anger due to video games is easily found.

References

James Robertson and Suzanne Robertson. *Volere Requirements Specification Template*. Atlantic Systems Guild Limited, 16 edition, 2012.

5 Appendix

This section has been added to the Volere template. This is where you can place additional information.

5.1 Symbolic Parameters

The definition of the requirements will likely call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.