SE 3XA3: Test Plan Rogue Reborn

Group #6, Team Rogue++

Ian PrinsprinsijMikhail Andrenkovandrem5Or Almogalmogo

Due Wednesday, December 7th, 2016

Contents

1	Introduction	4
	1.1 Overview	4
	1.2 Sections	4
2	Functional Requirements Evaluation	5
3	Non-Functional Requirements Evaluation	7
	3.1 Usability and Aesthetics	7
	3.2 Performance	8
	3.3 Robustness and Maintainability	9
	3.4 Safety	9
4	Comparison to Existing Implementation	10
5	Unit Testing	11
6	Changes Due to Testing	12
7	Automated Testing	13
	7.1 Automated Testing Strategy	13
	7.2 Specific System Tests	13
	7.3 Automated Testing Strategy	13
	7.4 Specific System Tests	13
8	Trace to Requirements	43
9	Trace to Modules	44
10	Code Coverage Metrics	46

List of Tables

1	Revision History
3	Test-Requirement Trace
5	Module Hierarchy
6	Test-Module Trace

List of Figures

Table 1: Revision History

Date	Version	Notes
12/06/16	0.1	Initial Draft
12/06/16	0.2	Automated Tests To PlayerChar
12/06/16	0.3	Functional Requirements Evaluation
12/07/16	0.4	Introduction
12/07/16	0.5	Finished Automated Tests

1 Introduction

1.1 Overview

The primary objective of this document is to provide a comprehensive summary of the verification process with respect to the Rogue Reborn project. Interested parties are welcome to analyze this paper as a means of evaluating the success of the final application regarding the requirements described in the SRS and tests prescribed in the Test Plan. After reviewing the document, the reader should understand the strengths and weaknesses of the Rogue Project as it relates to the expectations of the client.

1.2 Sections

A brief description of each Test Report section is provided below:

- §1 Brief overview of the Test Report
- §2 Functional evaluation of Rogue Reborn
- §3 Non-functional evaluation of Rogue Reborn
- §4 Description of relationship to original Roque with respect to testing
- §5 Explanation of unit testing in Rogue Reborn
- §6 List of changes that were performed as a consequence of testing
- §7 Tabular depiction of automated tests
- §8 Justification of test files with respect to functional requirements
- §9 Decomposition of modules and trace to test files
- §10 Summary of code coverage metrics

2 Functional Requirements Evaluation

Overall, an evaluation of functional requirements reveals near, if not complete coverage. The tests written for the projects turned out to be quite useful, as many caught bugs or business-errors that would have otherwise gone unnoticed. Those will be discussed below. As for the rest of the functional requirements, many were mundane, generic, or crucial enough to have already been satisfied before tests were considered. Those will not be discussed, as their complete satisfaction has already been verified countless times.

The list below refers to each functional requirement by its numerical identifier, as listed in the System Requirements Specification. Please refer to the SRS document if any confusion arises.

FR.16: When performing level tests, a strange anomaly led to one test constantly failing. The test revealed that the player, in fact, did not begin at the first level. Due to an off-by-one error and slight miscommunication between developers, the current level depth the player was on was i in some places and i+1 in others. As soon as the test revealed this, the problem was remedied globally.

FR.19: Whenever the player uncovers a new dungeon level (including the very first level), an algorithm decides on a position in which to place the user initially. This algorithm while appearing flawless, actually had a very slight chance of placing the player in an unreachable location, surrounded by walls, doomed forever. With the automatic tests running thousands upon thousands of simulations, the bug was quickly revealed, and remedied.

FR.35: Some monsters in Rogue Reborn follow a simple AI consisting of steps such as: Look for the player, chase the player, and if you can, hit the player. This is a very simple and easily-implemented AI to both invent and implement, but the enemies in Rogue Reborn go beyond such simple schemes. Some monsters do not seek to kill the player, but rather steal their precious items. One such pest is the Leprechaun, known as the symbol "L". The Leprechaun necessitated the implementation of a variety of methods that were previously unneeded, such as getNearestGold(). The testing of this function revealed some very serious performance issues in the pathfinding algorithm used throughout

the project, in which infinite path traces were possible. It also revealed another bug in which the coordinates of several level features (items mostly) were accidentally set to 0, rendering them impossible to reach by pathfinding and once again causing a pathfinding failure. This was due to the assumption that all items are placed on reachable blocks. Fortunately, thanks to the various tests we implemented, these bugs were caught and fixed.

FR.39: Working with C++ has its benefits, but also its drawbacks. An anomaly in the way C++ handles integers revealed a very serious bug in the code, in which player armor could reach utterly ridiculous values, rendering the player effectively invincible. By simulating every possibility of armor that can be made, this bug was caught and patched. To elaborate, the reason the bug even existed was because an unsigned integer was allowed to be reduced to a negative value, which of course means that it was not reduced to a negative number and instead went to the highest value an integer can be.

3 Non-Functional Requirements Evaluation

The following subsections evaluate the significant non-functional qualities of Rogue Reborn. To simplify notation, NFRT i is used to denote "Non-Functional Requirements Test i" from the Test Plan document. Note that the usability and playtesting surveys described in NFRT 1 , NFRT 2 , NFRT 4 , NFRT 7 , and NFRT 10 were not performed as a direct consequence of the time constraints imposed on the project (the Gantt Chart schedules this survey to be released in early January, 2017) in addition to the sophisticated software environment required for compilation. Distributing a single executable file is also currently infeasible, as Rogue Reborn makes use of a library that exclusively compiles for Linux at the present moment.

3.1 Usability and Aesthetics

Overall, the visual appearance of the application was well-received by the Rogue Reborn stakeholders. This was deduced through the interactions between the Rogue++ team and the *SFWRENG 3XA3* instructor staff, as well as informal conversations with other colleagues. Unfortunately, the usability survey described in NFRT 1 will be carried out in the future, so the impressions of the general public are not yet known.

Since the usability of the original *Rogue* was relatively poor due to its seemingly-arbitrary key bindings, the Rogue Reborn application made goals to improve this area. Specifically, the application featured arrow key bindings for some of player character movements in order to accommodate a more standard and intuitive keyboard layout. However, due to the plethora of other key bindings, the Rogue++ team was *not* successful in alleviating this issue completely. A summary of NFRT 3 is given below.

Non-Functional Requirement Test # 3 Summary

Summary: All strings in the Rogue Reborn source code were

extracted and placed in a text file, where a developer examined all indicated errors that were potentially associated with a GUI output using Microsoft Word. The script that performed the string extraction is located in the src/misc directory under the name stringfinder.py.

Results: The aforementioned script managed to locate

approximately 1400 strings. After manually verifying the grammatical correctness and spelling of each string in Word, it was determined that the GUI output is free of

linguistic errors.

3.2 Performance

In general, the technical performance of the Rogue Reborn client was greatly enhanced by the developers' decision to use C++ as opposed to Python or Java. Consequently, the application averaged around 120 FPS (Frames Per Second) and delivered a smooth experience while on a VM (Virtual Machine). During the final stages of development, the Rogue++ team decided to profile the application using the GDB (GNU Debugger) with respect to peak memory usage and pleasantly discovered that the maximum amount of RAM allocated to Rogue Reborn was 1 MB.

A synopsis of the relevant tests (NFRT 5 and NFRT 6) is portrayed below.

Non-Functional Requirement Test # 5 Summary

Summary: The Rogue Reborn application was compiled with a

special debug parameter; this enabled particular sections of the MasterController module to use a StopWatch implementation to measure the average maximum

execution time between successive frames.

Results: On average, the execution time between successive user

actions appeared to stabilize around 20 ms. Clearly, this is appreciably lower than the maximum allowable delay

of ~ 30 ms.

Non-Functional Requirement Test # 6 Summary

Summary: The following regular expression was applied over the

source code to extract all integer-typed declarations:

(unsigned|int|long)\s+[A-Za-z]+\s*(,|;|=.*)\s*

An associated script to perform the declaration extraction was also created and is located in the src/misc

directory under the name intdeclare.py.

Results: The aforementioned script managed to locate

approximately 170 candidate declarations (there were

several false positives). Among these, there were no

obvious candidates for integer overflow.

3.3 Robustness and Maintainability

Mikhail

3.4 Safety

Mikhail

4 Comparison to Existing Implementation

According to all collected resources of the original implementation, there were no tests done to verify the accuracy of the original product. This is somewhat understandable, all things considered. After all, the original product was released in 1980, almost 40 years ago. Since then, standards of software development were transformed from infanthood to the rigorous forms they take today.

As such, the tests we have written have nothing to be compared to. If there were tests to compare to, we would take a look at the following:

- How does the coverage of the existing test cases compare to the coverage we implemented?
- For cases where the modules/classes are similar in both versions, how do the tests compare? Are there any significant benefits or drawbacks to one or the other?
- Does any one of the implementations completely neglect an aspect of another? Is the remade implementation missing something critical that the original did test for?
- How are random tests approached by the existing implementation? Are random numbers ever used in the testing phase?

5 Unit Testing

Mikhail

6 Changes Due to Testing

Mikhail

7 Automated Testing

7.1 Automated Testing Strategy

For this project we elected not to use a 3rd party testing library. We made this decision to ease configuration/installation problems and reduce our dependencies, as we judged it would not be necessary. Instead a series of files (labeled test.foobar.cpp) in the repository hold tests, which are run by our custom test runner. These automated tests are run on command by executing the produced executable, or by the continuous integration script run whenever changes are pushed to the central repository. The results of these tests are automatically reported, resulting in a failed or successful build.

7.2 Specific System Tests

The following is a list of all automated tests in the project.

7.3 Automated Testing Strategy

For this project we elected not to use a 3rd party testing library. We made this decision to ease configuration/installation problems and reduce our dependencies, as we judged it would not be necessary. Instead a series of files (labeled test.foobar.cpp) in the repository hold tests, which are run by our custom test runner. These automated tests are run on command by executing the produced executable, or by the continuous integration script run whenever changes are pushed to the central repository. The results of these tests are automatically reported, resulting in a failed or successful build.

7.4 Specific System Tests

The following is a list of all system tests in the project.

Name: Amulet Construction

Initial State: None

Input: Coordinate, context value

Expected Output: Amulet object in valid initial state

Name: Initial State: Input: Expected Output:	Armor Construction 1 None Coordinate Armor object in valid initial state
Name: Initial State: Input: Expected Output:	Armor Construction 2 None Coordinate, context value, type value Armor object in valid initial state
Name: Initial State: Input: Expected Output:	Armor Identification Cursed Armor None Verification that armor is identified
Name: Initial State: Input: Expected Output:	Armor Curse Cursed Armor None Verification that armor is cursed
Name: Initial State: Input: Expected Output:	Armor Enchantment Cursed Armor Curse level Verification that armor enchantment is correct
Name: Initial State: Input: Expected Output:	Armor Rating Cursed Armor None Verification that armor rating is correct
Name: Initial State: Input: Expected Output:	Coordinate Ordering None $(0,0)$ coordinate and $(1,1)$ coordinate Verification that $(0,0)$; $(1,1)$
Name:	Coordinate Equality

Initial State: Input: Expected Output:	None Two (0,0) coordinates Verification that the two inputs are equal
Name: Initial State: Input: Expected Output:	Coordinate Inequality None (0,0) coordinate and (1,1) coordinate Verification that the two inputs are not equal
Name: Initial State: Input: Expected Output:	Coordinate Addition None (2,3) coordinate and (1,2) coordinate (3,5) coordinate
Name: Initial State: Input: Expected Output:	Coordinate Subtraction None (2,3) coordinate and (1,2) coordinate (1,1) coordinate
Name: Initial State: Input: Expected Output:	Feature Construction None Symbol, coordinate, visibility, color Feature object in valid initial state
Name: Initial State: Input: Expected Output:	Feature Symbol Check Feature with given symbol Symbol Verification that feature's symbol matches given
Name: Initial State: Input: Expected Output:	Feature Invisibility Check Invisible feature None Verification that feature is invisible
Name: Initial State:	Feature Visibility Check Visible feature

None

Input:

Expected Output:	Verification that feature is visible	
Name: Initial State: Input: Expected Output:	Feature Location Check Feature with given location Coordinate Verification that feature's location matches given coordinate	
Name: Initial State: Input: Expected Output:	Food Construction None Coordinate and context value Food object in valid initial state	
Name: Initial State: Input: Expected Output:	Food Eating Food and player objects None Verification that food has increased the player's food life by an appropriate amount	
Name: Initial State: Input: Expected Output:	GoldPile Construction None Coordinate, gold amount value GoldPile object in valid initial state	
Name: Initial State: Input: Expected Output:	GoldPile Quantity Check GoldPile with given amount of gold Amount of gold value Verification that gold's amount matches given amount	
Name: Initial State: Input: Expected Output:	Item Construction 1 None Symbol, coordinate, context value, item class specifier, name value, psuedoname value, item type specifier, item stackability value, item throwability value, weight value Item object in valid initial state	
Name:	Item Construction 2	

Initial State: Input: Expected Output:	None Symbol, coordinate, context value, item class specifier, name value, psuedoname value, item type specifier, item stackability value, item throwability value, weight value Item object in valid initial state
Name: Initial State: Input: Expected Output:	Name Vector Check None Vector of item names Shuffled vector of item names
Name: Initial State: Input: Expected Output:	Item Curse Check Uncursed item None Verification that item is uncursed
Name: Initial State: Input: Expected Output:	Item Curse/Effect Check 1 Uncursed item to which the cursed effect has been applied None Verification that item is cursed
Name: Initial State: Input: Expected Output:	Item Curse/Effect Check 2 Cursed item whose curse effect has been removed None Verification that item is uncursed
Name: Initial State: Input: Expected Output:	Item Unindentified Check Identified item None Verification that item is unidentified
Name: Initial State: Input: Expected Output:	Item Identified Check Unidentified item None Verification that item is identified

Name: Initial State: Input: Expected Output:	Item Display-Name Check 1 Unidentified item Psuedoname Verification that item's display name matches psuedoname		
Name: Initial State: Input: Expected Output:	Item Display-Name Check 2 Identified item True name Verification that item's display name matches true name		
Name: Initial State: Input: Expected Output:	ItemZone Containment Check 1 ItemZone with 2 items None Verification that ItemZone contains the first item		
Name: Initial State: Input: Expected Output:	ItemZone Containment Check 2 ItemZone with 2 items None Verification that ItemZone contains the second item		
Name: Initial State: Input: Expected Output:	ItemZone Empty Check ItemZone with 2 items None Verification that ItemZone is not empty		
Name: Initial State: Input: Expected Output:	ItemZone Size Check ItemZone with 2 items None Verification that ItemZone's size is 2		
Name: Initial State: Input: Expected Output:	ItemZone Keybind Check 1 ItemZone with 2 items None Verification that first item is bound to 'a' key		

Name: Initial State: Input: Expected Output:	ItemZone Keybind Check 2 ItemZone with 2 items None Verification that second item is bound to 'b' key
Name: Initial State: Input: Expected Output:	ItemZone Contents Retrieval 1 ItemZone with 2 items None Item map with exactly 1 copy of first item
Name: Initial State: Input: Expected Output:	ItemZone Contents Retrieval 2 ItemZone with 2 items None Item map with exactly 1 copy of second item
Name: Initial State: Input: Expected Output:	ItemZone Removal ItemZone with 2 items Removal command ItemZone with only second item
Name: Initial State: Input: Expected Output:	ItemZone Keybind Persistence ItemZone with first item removed None Verification that second item is still bound to 'b'
Name: Initial State: Input: Expected Output:	ItemZone Weight Enforcement Empty ItemZone Attempt to add 500 pieces of armor to ItemZone ItemZone with max-weight worth of armor
Name: Initial State: Input: Expected Output:	Level Construction None Depth, player object Level object in valid initial state
Name:	Level Depth Check

Initial State: Input: Expected Output:	Level with given depth Depth value Verification that level's depth matches given value
Name: Initial State: Input: Expected Output:	Level BFSPerp Diagonal Small Empty level object Pair of coordinates diagonally adjacent Path between coordinates with expected length, utilizing taxicab movement
Name: Initial State: Input: Expected Output:	Level BFSPerp Horizontal Empty level object Pair of coordinates with equal y-values Path between coordinates with expected length, utilizing taxicab movemen
Name: Initial State: Input: Expected Output:	Level BFSPerp Vertical Empty level object Pair of coordinates with equal x-values Path between coordinates with expected length, utilizing taxicab movemen
Name: Initial State: Input: Expected Output:	Level BFSDiag Horizontal Empty level object Pair of coordinates with equal y-values Path between coordinates with expected length, utilizing orthogonal movement
Name: Initial State: Input: Expected Output:	Level BFSDiag Vertical Empty level object Pair of coordinates with equal x-values Path between coordinates with expected length, utilizing orthogonal movement
Name: Initial State: Input:	Level BFSPerp Diagonal Empty level object Pair of coordinates on diagonal line

Expected Output:	Path between coordinates with expected length, utilizing taxicab movement
Name: Initial State: Input: Expected Output:	Level Starting Position Empty level object None Valid starting position coordinate
Name: Initial State: Input: Expected Output:	Level getAdjPassable Empty level object Coordinate List of coordinates orthogonally adjacent to given coordinate
Name: Initial State: Input: Expected Output:	Level Path Generation Player object and generated level Series of path requests between random coordinates Valid paths between locations
Name: Initial State: Input: Expected Output:	Level Connectedness Player object and generated level Series of path requests between all rooms in the level Valid paths between each room
Name: Initial State: Input: Expected Output:	Level Staircase Check Player object and generated level None Verification that level contains a staircase
Name: Initial State: Input: Expected Output:	Level GoldPile Check Player object and generated level None Verification that level contains at least one goldpile
Name: Initial State:	Monster Construction None

Input: Expected Output:	level value, maxHP value, name value	
Name: Initial State: Input: Expected Output:	Dice-Math 1 None 1 1-sided die Sum of values of 1	
Name: Initial State: Input: Expected Output:	Dice-Math 2 None 2 1-sided die Sum of values of 2	
Name: Initial State: Input: Expected Output:	Dice-Math 3 None 1 2-sided die 1 ;= Sum of values ;= 2	
Name: Initial State: Input: Expected Output:	Dice-Math 4 None 3 4-sided die $3 \text{ i= Sum of values i= } 12$	
Name: Initial State: Input: Expected Output:	Mob Armor Check Mob object None Verification mob armor is in valid range	
Name: Initial State: Input: Expected Output:	Mob HP Check 1 Mob with given HP value HP value Verification mob has correct HP value	
Name: Initial State: Input:	Mob MaxHP Check Mob with given MaxHP value MaxHP value	

Expected Output:	Verification mob has correct MaxHP value
Name: Initial State: Input: Expected Output:	Mob Level Check Mob with given level value Level value Verification mob has correct level value
Name: Initial State: Input: Expected Output:	Mob Location Check Mob with given location Coordinate Verification mob has correct location
Name: Initial State: Input: Expected Output:	Mob Name Check Mob with given name Name value Verification mob has correct name
Name: Initial State: Input: Expected Output:	Mob setMaxHP Mob with default MaxHP setMaxHP command with MaxHP value mob with given MaxHP value
Name: Initial State: Input: Expected Output:	Mob setcurrentHP Mob with default currentHP setCurrentHP command with currentHP value mob with given currentHP value
Name: Initial State: Input: Expected Output:	Mob Dead Check 1 Living Mob object None Verification mob is alive
Name: Initial State: Input: Expected Output:	Mob HP Check 2 Living Mob object Hit command for ¿¿¿ mob's current HP Verification mob has HP ¡= 0

Name: Initial State: Input: Expected Output:	Mob Dead Check 2 Dead mob object None Verification mob is dead
Name: Initial State: Input: Expected Output:	Monster Construction None Symbol, coordinate Monster object in valid initial state
Name: Initial State: Input: Expected Output:	Monster Flag/Invisibility Visible monster object SetFlag command to make monster invisible Invisible monster object
Name: Initial State: Input: Expected Output:	Monster Aggrevate Idling, sleeping monster object Aggrevate command Awake, chasing monster object
Name: Initial State: Input: Expected Output:	Monster Damage Calculation Monster object calculateDamage command Correct amount of damage
Name: Initial State: Input: Expected Output:	Monster Hit Chance Monster and player objects calculateHitChange command Hit chance in valid range
Name: Initial State: Input: Expected Output:	Monster Armor Check Monster object None Verification that monster armor is in valid range
Name:	Invisible Monster Name Check

Initial State: Input: Expected Output:	Invisible uonster object None Verification monster has hidden name
Name: Initial State: Input: Expected Output:	Visible Monster Name Check Invisible monster object RemoveFlag command to make monster invisible Verification monster has real name
Name: Initial State: Input: Expected Output:	Monster Symbol/Level Association None Depth value Set of symbols for monsters that are valid candidates for given depth
Name: Initial State: Input: Expected Output:	Monster Symbol/Treasure/Level Association None Depth value Set of symbols for monsters that are valid candidates for given depth for a treasure room
Name: Initial State: Input: Expected Output:	PlayerChar Initial Amulet Check Just initialized playerchar object None Verification the game does not believe the player has the amulet
Name: Initial State: Input: Expected Output:	PlayerChar Initial HP Check Just initialized playerchar object None Verification playerchar has full hp
Name: Initial State: Input: Expected Output:	PlayerChar Level-Up Exp Playerchar object at initial level Exp input into playerchar object Playerchar object with increased level

Name: Initial State: Input: Expected Output:	PlayerChar Level-Up Manual Playerchar object Level-up command Playerchar object with increased level
Name: Initial State: Input: Expected Output:	PlayerChar Damage Playerchar object at full hp Series of damage commands applied to playerchar object Playerchar object with less than full hp
Name: Initial State: Input: Expected Output:	PlayerChar UnArmed 1 Unarmed playerchar object calculateDamage command 0 damage value
Name: Initial State: Input: Expected Output:	PlayerChar Armed Playerchar object armed with weapon calculateDamage command Damage value ¿ 0
Name: Initial State: Input: Expected Output:	PlayerChar Stow Weapon Playerchar object armed with uncursed weapon removeWeapon command PlayerChar object unarmed
Name: Initial State: Input: Expected Output:	PlayerChar UnArmed 2 Armed playerchar object removeWeapon command, then calculateDamage 0 damage value
Name: Initial State: Input: Expected Output:	PlayerChar Remove Non-Armor Playerchar object with no armor removeArmor command Boolean indicating failure to remove armor
Name:	PlayerChar Remove Armor

Initial State: Input: Expected Output:	Playerchar object with uncursed armor removeArmor command Playerchar object without armor
Name: Initial State: Input: Expected Output:	Potion Construction 1 None Coordinate Potion object in valid initial state
Name: Initial State: Input: Expected Output:	Potion Construction 2 None Coordinate, item context value, item type specifier Potion object in valid initial state
Name: Initial State: Input: Expected Output:	Potion of Strength Player object Potion of strength Player with strength increased by 1
Name: Initial State: Input: Expected Output:	Potion of Restore Strength Player object with reduced strength Potion of restore strength Player object with pre-reduction strength
Name: Initial State: Input: Expected Output:	Potion of Healing Player object with full hp Potion of healing Player object with maxHP increased by 1
Name: Initial State: Input: Expected Output:	Potion of Extra Healing Player object with full hp Potion of extra healing Player object with maxHP increased by 2
Name: Initial State:	Potion of Poison Player object with strength ; 0

Potion of poison

Input:

Expected Output:	Player object with reduced strength
Name: Initial State: Input: Expected Output:	Potion of Raise Level Player object with less than max level Potion or raise level Player object with level + 1
Name: Initial State: Input: Expected Output:	Potion of Blindness Player object without the blindness condition Potion of blindness Player object with the blindness condition
Name: Initial State: Input: Expected Output:	Potion of Hallucination Player object without the hallucination condition Potion of hallucination Player object with the hallucination condition
Name: Initial State: Input: Expected Output:	Potion of Detect Monster Player object without the detect-monsters condition Potion of detect monsters Player object with the detect-monsters condition
Name: Initial State: Input: Expected Output:	Potion of Detect Object Player object without the detect-objects condition Potion of detect objects Player object with the detect-objects condition
Name: Initial State: Input: Expected Output:	Potion of Confusion Player object without the confusion condition Potion of confusion Player object with the confusion condition
Name: Initial State: Input: Expected Output:	Potion of Confusion Player object without the confusion condition Potion of confusion Player object with the confusion condition

Name: Initial State: Input: Expected Output:	Potion of Levitation Player object without the levitation condition Potion of levitation Player object with the levitation condition
Name: Initial State: Input: Expected Output:	Potion of Haste Player object without the haste condition Potion of haste Player object with the haste condition
Name: Initial State: Input: Expected Output:	Potion of See-Invisible Player object without the invisible-sight condition Potion of invisible Player object with the invisible-sight condition
Name: Initial State: Input: Expected Output:	Random Range 1 None Upper and lower bounds 0,0 0
Name: Initial State: Input: Expected Output:	Random Range 2 None Upper and lower bounds 5,5 5
Name: Initial State: Input: Expected Output:	Random Range 3 None Upper and lower bounds 0,60, repeated 40 times 0 = result = 60
Name: Initial State: Input: Expected Output:	Random Float None 40 repeats 0 result 1
Name:	Random Boolean

Initial State: Input: Expected Output:	None 10 repeats Both true and false are generated
Name: Initial State: Input: Expected Output:	Random Percent None 40 repeats 0 = result = 100
Name: Initial State: Input: Expected Output:	Random Position None Two coordinates, as top-left and bottom-right of rectangle, 10 repeats Random coordinates within the bounds
Name: Initial State: Input: Expected Output:	Ring Construction 1 None Coordinate Ring object with valid initial state
Name: Initial State: Input: Expected Output:	Ring Construction 2 None Coordinate, item context value, type identifier Ring object with valid initial state
Name: Initial State: Input: Expected Output:	Ring of Stealth Player object without stealth condition Ring of stealth Player object with the stealth condition
Name: Initial State: Input: Expected Output:	Ring of Stealth Deactivate Player object with ring of stealth Remove ring Player object without the stealth condition
Name: Initial State:	Ring of Teleportation Player object without random teleportation condition

Input: Expected Output:	Ring of teleportation Player object with the random teleportation condition
Name: Initial State: Input: Expected Output:	Ring of Teleportation Deactivate Player object with ring of teleportation Remove ring Player object without the random teleportation condition
Name: Initial State: Input: Expected Output:	Ring of Regeneration Player object without regeneration condition Ring of regeneration Player object with the regeneration condition
Name: Initial State: Input: Expected Output:	Ring of Regeneration Deactivate Player object with ring of regeneration Remove ring Player object without the regeneration condition
Name: Initial State: Input: Expected Output:	Ring of Digestion Player object without digestion condition Ring of digestion Player object with the digestion condition
Name: Initial State: Input: Expected Output:	Ring of Digestion Deactivate Player object with ring of digestion Remove ring Player object without the digestion condition
Name: Initial State: Input: Expected Output:	Ring of Dexterity Player object Ring of dexterity Player object with dexterity increased by the appropriate amount
Name: Initial State:	Ring of Dexterity Deactivate Player object with ring of dexterity

Input: Expected Output:	Remove ring Player object with normal dexterity
Name: Initial State: Input: Expected Output:	Ring of Adornment Player object Ring of adornment Identical player object
Name: Initial State: Input: Expected Output:	Ring of Adornment Player object with ring of adornment Remove ring Identical player object
Name: Initial State: Input: Expected Output:	Ring of See-Invisible Player object without the see-invisible condition Ring of see-invisible Player object with the see-invisible condition
Name: Initial State: Input: Expected Output:	Ring of See-Invisible Deactivate Player object with ring of see-invisible Remove ring Player object without the see-invisible condition
Name: Initial State: Input: Expected Output:	Ring of Maintain-Armor Player object without the maintain-armor condition Ring of maintain-armor Player object with the maintain-armor condition
Name: Initial State: Input: Expected Output:	Ring of Maintain-Armor Deactivate Player object with ring of maintain-armor Remove ring Player object without the maintain-armor condition
Name: Initial State: Input:	Ring of Searching Player object without the auto-search condition Ring of searching

Expected Output:	Player object with the auto-search condition
Name: Initial State: Input: Expected Output:	Ring of Searching Deactivate Player object with ring of searching Remove ring Player object without the auto-search condition
Name: Initial State: Input: Expected Output:	Room Construction Check 1 Randomly generated room None Verification that room's size is in valid range
Name: Initial State: Input: Expected Output:	Room Construction Check 2 Randomly generated room None Verification that room edges are within valid bounds
Name: Initial State: Input: Expected Output:	Scroll Construction 1 None Coordinate Scroll object in valid initial state
Name: Initial State: Input: Expected Output:	Scroll Construction 2 None Coordinate, item context value, type identifier Scroll object in valid initial state
Name: Initial State: Input: Expected Output:	Scroll PseudoNames Scrolls are uninitialized initializeScrollNames command Vector of valid scroll psuedonames
Name: Initial State: Input: Expected Output:	Scroll of Protect Armor Player with cursed armor Scroll of protect armor Player with uncursed armor with protect-armor effect

Name: Initial State: Input: Expected Output:	Scroll of Hold Monster Monster without the held flag Scroll of hold monster Monster with the held flag
Name: Initial State: Input: Expected Output:	Scroll of Enchant Weapon Player with weapon Scroll of enchant weapon Player with uncursed weapon with higher enchant level
Name: Initial State: Input: Expected Output:	Scroll of Enchant Armor Player with armor Scroll of enchant armor Player with uncursed armor with higher enchant level
Name: Initial State: Input: Expected Output:	Scroll of Identity None Scroll identity No exceptions
Name: Initial State: Input: Expected Output:	Scroll of Teleportation Player at coordinate $(0,0)$ Scroll of teleportation Player at coordinate $!=(0,0)$
Name: Initial State: Input: Expected Output:	Scroll of Sleep Player without the sleep condition Scroll of sleep Player with the sleep condition
Name: Initial State: Input: Expected Output:	Scroll of Scare Monster None Scroll of scare monster No exceptions
Name:	Scroll of Remove Curse

Initial State: Input: Expected Output:	Player with cursed weapon Scroll of remove curse Player with uncursed weapon	
Name: Initial State: Input: Expected Output:	Scroll of Create Monster Level object Scroll of create monster Level with 1 additional monster	
Name: Initial State: Input: Expected Output:	Scroll of Aggravate Monster Level with sleeping monsters Scroll of aggravate monster Level with no sleeping monsters	
Name: Initial State: Input: Expected Output:	Scroll of Magic Mapping Unrevealed level Scroll of magic mapping Level where all tiles have been revealed	
Name: Initial State: Input: Expected Output:	Scroll of Confuse Monster Player without the confuse-monster condition Scroll of confuse monster Player with the confuse-monster condition	
Name: Initial State: Input: Expected Output:	Stair Construction None Coordinate, direction value Stair object in valid initial state	
Name: Initial State: Input: Expected Output:	Stair Direction Check Stair constructed with direction Direction value Verification stair has given direction value	
Name: Initial State: Input:	Floor Passability Check Floor object None	

Expected Output:	Verification floor is passable
Name: Initial State: Input: Expected Output:	Floor Symbol Check Floor object None Verification floor has correct symbol
Name: Initial State: Input: Expected Output:	Floor Transparency Check Floor object None Verification floor is transparent
Name: Initial State: Input: Expected Output:	Wall Passability Check Wall object None Verification wall is not passable
Name: Initial State: Input: Expected Output:	Wall Symbol Check Wall object None Verification wall has correct symbol
Name: Initial State: Input: Expected Output:	Wall Opacity Check Wall object None Verification wall is transparent
Name: Initial State: Input: Expected Output:	Corridor Passability Check Corridor object None Verification corridor is passable
Name: Initial State: Input: Expected Output:	Corridor Symbol Check Corridor object None Verification corrido has correct symbol

Name: Initial State: Input: Expected Output:	Corridor Transparency Check Corridor object None Verification corridor has special corridor transparency	
Name: Initial State: Input: Expected Output:	Door Passability Check Door object None Verification door is not passable	
Name: Initial State: Input: Expected Output:	Door Symbol Check Door object None Verification corridor has correct symbol	
Name: Initial State: Input: Expected Output:	Door Transparency Check Door object None Verification Door has special corridor transparency	
Name: Initial State: Input: Expected Output:	Door Trap Player and level Door trap Player at a level with depth + 1	
Name: Initial State: Input: Expected Output:	Rust Trap Player with enchanted weapon Rust trap Player with unenchanted weapon	
Name: Initial State: Input: Expected Output:	Sleep Trap Player without the sleep condition Sleep trap Player with the sleep condition	
Name:	Bear Trap	

Initial State: Input: Expected Output:	Player without the immobilized condition Bear trap Player with the immobilized condition	
Name: Initial State: Input: Expected Output:	Teleport Trap Player Teleport trap Player at a different location	
Name: Initial State: Input: Expected Output:	Dart Trap Player Dart trap Player with less hp	
Name: Initial State: Input: Expected Output:	Tunnel Digging Level and pair of unconnected rooms Dig command Valid path between the two rooms	
Name: Initial State: Input: Expected Output:	Open Inventory Screen Playstate, player, empty level Inventory key Inventory screen	
Name: Initial State: Input: Expected Output:	Close Inventory Screen Inventory screen, player, empty level Exit key Playstate	
Name: Initial State: Input: Expected Output:	Movement Playstate, player, empty level Movement key Player should be in expected location in the level	
Name: Initial State:	Open Status Screen Playstate, player, empty level	

Status key

Input:

Expected Output:	Status screen
Name: Initial State: Input: Expected Output:	Exit Status Screen Status Screen, player, empty level Exit key Playstate
Name: Initial State: Input: Expected Output:	No Wand Zap Playstate, player with no wand Zap key Unchanged playstate
Name: Initial State: Input: Expected Output:	Zap Wand Select Playstate, player with wand, empty level Zap key, then direction key Inventory Screen
Name: Initial State: Input: Expected Output:	Zap Wand Fire 1 Inventory wand select Item select hotkey Playstate
Name: Initial State: Input: Expected Output:	Zap Wand Fire 2 Inventory wand select Item select hotkey wand with charges - 1
Name: Initial State: Input: Expected Output:	Game Quit Playstate Quit key and confirmation key RIPScreen
Name: Initial State: Input: Expected Output:	Wand Construction 1 None Coordinate Wand in valid initial state

Name: Initial State: Input: Expected Output:	Wand Construction 2 None Coordinate, item context value, type specifier Wand in valid initial state	
Name: Initial State: Input: Expected Output:	Wand of Teleport Away Player, nearby monster Wand of teleport away Monster has distance to player $i = 20$	
Name: Initial State: Input: Expected Output:	Wand of Slow Monster Player, monster without slowed flag Wand of slow monster Monster has slowed flag	
Name: Initial State: Input: Expected Output:	Wand of Invisibility Player, monster without invisible flag Wand of invisibility Monster with invisible flag	
Name: Initial State: Input: Expected Output:	Wand of Polymorph Player, monster Wand of polymorph Different monster at previous monster's locations	
Name: Initial State: Input: Expected Output:	Wand of Haste Monster Player, monster without haste flag Wand of haste monster Monster with haste flag	
Name: Initial State: Input: Expected Output:	Wand of Magic Missile Player, monster Wand of magic missile Monster with reduced hp	
Name:	Wand of Cancellation	

Initial State: Input: Expected Output:	Player, monster without cancelled flag Wand of cancellation Monster with cancelled flag	
Name: Initial State: Input: Expected Output:	Wand of Do Nothing Player, monster Wand of do nothing No exceptions	
Name: Initial State: Input: Expected Output:	Wand of Drain Life Player with reduced health, monster Wand of drain life Player with increased health, monster with reduced health	
Name: Initial State: Input: Expected Output:	Wand of Cold Player, monster Wand of cold No exceptions	
Name: Initial State: Input: Expected Output:	Wand of Fire Player, monster Wand of fire No exceptions	
Name: Initial State: Input: Expected Output:	Weapon Construction 1 None Coordinate Weapon in valid initial state	
Name: Initial State: Input: Expected Output:	Weapon Construction 2 None Coordinate, item context value, type specifier Weapon in valid initial state	
Name: Initial State:	Weapon Identification Check Identified weapon	

Input: Expected Output:	None Verification that weapon is identified
Name: Initial State: Input: Expected Output:	Weapon Curse Check Cursed weapon None Verification that weapon is cursed
Name: Initial State: Input: Expected Output:	Weapon Name Check Weapon None Verification that weapon has valid name
Name: Initial State: Input: Expected Output:	Weapon Enchantment Check Cursed weapon None Verification that weapon has expected enchantment values

8 Trace to Requirements

The following table maps each implemented test file to a set of functional and non-functional requirements

Table 3: Test-Requirement Trace

File	Related Requirement(s)	
test.amulet.cpp	FR.25	
test.armor.cpp	FR.29, FR.34, FR.39,	
test.coord.cpp	FR.17	
test.feature.cpp	FR.5, FR.13, FR.14, FR.15, FR.25, FR.31	
test.food.cpp	FR.5, FR.31	
test.goldpile.cpp	FR.5	
test.item.cpp	FR.5, FR.13, FR.14, FR.15, FR.25, FR.30 FR.31	
test.itemzone.cpp	FR.5, FR.9, FR.26	
test.level.cpp	FR.16-19	
test.levelgen.cpp	FR.16-19	
test.main.cpp	Put everything together	
test.mob.cpp	FR.37, FR.38, FR.39	
test.monster.cpp	FR.35-39	
test.playerchar.cpp	FR.9-15, FR.26-34, NFR.5	
test.potion.cpp	FR.5, FR.13, FR.14, FR.15, FR.25, FR.31	
test.ring.cpp	FR.5, FR.13, FR.14, FR.15, FR.25, FR.31	
test.room.cpp	FR.17, FR.18, FR.19, FR.21	
test.scroll.cpp	FR.5, FR.13, FR.14, FR.15, FR.25, FR.31	
test.stairs.cpp	FR.18, FR.19	
test.terrain.cpp	FR.13, FR.15, FR.18, FR.19, FR.23, FR.24	
test.testable.cpp	Defines test-suite	
test.testable.h	Defines test-suite	
test.trap.cpp	FR.12, FR.15, FR.19, FR.20, FR.23, FR.24, FR.34	
test.tunnel.cpp	FR.17, FR.19	
test.uistate.cpp	FR.1-4, FR.6-10, NFR.1, NFR.3, NFR.5	
test.wand.cpp	FR.5, FR.13, FR.14, FR.15, FR.25, FR.31	
test.weapon.cpp	FR.5, FR.13, FR.14, FR.15, FR.25, FR.31	

9 Trace to Modules

The following table re-iterates the modules of the project, along with their respective domain and module ID. The module IDs are used to refer to modules in the trace. More about the modules can be found in the Module Guide.

Table 5: Module Hierarchy

Level 1	Level 2	
Hardware-Hiding	BasicIO	M1
Module	Doryen	M2
	Input Format	M3
Behaviour-Hiding	External	M4
Module	Item	M5
	Level	M6
	LevelGen	M7
	MainMenu	M8
	Mob	M9
	Monster	M10
	PlayerChar	M11
	RipScreen	M12
	PlayState	M13
	SaveScreen	M14
	UIState	M15
Software Decision	Coord	M16
Module	Feature	M17
	ItemZone	M18
	MasterController	M19
	Random	M20
	Terrain	M21

The following table maps test files, which implement tests, to specific modules, given by their IDs.

Table 6: Test-Module Trace

File	Related Module(s)
test.amulet.cpp	M7, M13, M15
test.armor.cpp	M5, M9, M11
test.coord.cpp	M2, M5, M6, M7, M16, M20
test.feature.cpp	M5, M11, M17, M18
test.food.cpp	M5, M6, M7, M11, M13
test.goldpile.cpp	M5, M6, M7, M10, M11, M17, M18
test.item.cpp	M5, M17
test.itemzone.cpp	M5, M6, M16, M17, M18
test.level.cpp	M5, M6, M10, M11, M16, M17, M20
test.levelgen.cpp	M5, M6, M10, M16, M17, M20, M21
test.main.cpp	None (Puts everything together)
test.mob.cpp	M9, M10, M11, M13, M15, M16
test.monster.cpp	M9, M10, M16
test.playerchar.cpp	M5, M6, M9, M11, M12, M13, M15, M16, M17, M18, M19
test.potion.cpp	M5, M6, M7, M10, M11, M17, M18
test.ring.cpp	M5, M6, M7, M10, M11, M17, M18
test.room.cpp	M6, M7, M16, M20
test.scroll.cpp	M5, M6, M7, M10, M11, M17, M18
test.stairs.cpp	M7, M17, M19, M21
test.terrain.cpp	M6, M7, M20, M21
test.testable.cpp	Defines test-suite
test.testable.h	Defines test-suite
test.trap.cpp	M6, M7, M11, M15, M17
test.tunnel.cpp	M5, M6, M16
test.uistate.cpp	M4, M8, M12, M13, M15, M19
test.wand.cpp	M5, M6, M7, M10, M11, M17, M18
test.weapon.cpp	M5, M6, M7, M10, M11, M17, M18

10 Code Coverage Metrics

By looking at the test-requirements matrix, and also cross-referencing the test-module trace above with the module-requirements trace given in the Module Guide, it is possible to determine exactly which functional and nonfunctional requirements were satisfied with the test cases we created.

As can be expected, near **complete coverage** of both functional and non-functional requirements is achieved. Except for a few non-functional requirements, the modules and direct requirements reflected in the test cases offer a complete coverage of the requirements. Some (in particular, non-functional) requirements are nigh impossible to test using code. An example includes NFR.2: "The Rogue Reborn game shall be fun and entertaining." Whatever software exists that can determine such a thing would never pass the Turing test, and thus can be deemed an impossibility as of current technology. But while it is impossible to test with code, such a thing is easily testable with human playtesters.

Along with NFR.2, several non-functional requirements were not feasible to assert with software, but all were correctly proven by other means, most of which involved manual human labor.

To expand on the previous statements, we encountered some requirements where the achievable target was difficult to materialize, but still algorithmic and computational in nature. A prime example of this is the luminosity constraint, which ruled that no two consecutive frames may have a change in brightness greater than some defined delta. In order to properly measure this, we had to go outside of the program, and write a separate script to do the hard work. We used python to calculate the pixel-accurate luminosity of some key screenshots, and using the calculation proposed by the non-functional requirement, arrived at correct results. The results were deemed close enough to the predefined delta, which itself was based more or less on our intuition.