SE 3XA3: Test Plan
Rogue Reborn

Group #6, Team Rogue++

Ian Prins	prinsij
Mikhail Andrenkov	andrem5
Or Almog	almogo

Due Thursday, December 8th, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
12/06/16	0.1	Initial Draft
12/06/16	0.2	Added Automated Tests To PlayerChar
12/06/16	0.3	Added FR Evaluation
12/07/16	0.4	Added Introduction
12/07/16	0.5	Finished Automated Tests
12/07/16	0.6	Added NFR Evaluation
12/07/16	0.7	Added Unit Testing
12/07/16	0.8	Added Changes Due to Testing

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 *Rogue* 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. Unfortunately, the usability and playtesting surveys referenced in NFRT 1, NFRT 2, NFRT 4, NFRT 7, NFRT 9, NFRT 10, NFRT 12, and NFRT 14 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). Another hindering factor is the sophisticated software environment required for compilation: publicly distributing a single executable file is not feasible since Rogue Reborn makes use of a library that exclusively compiles for Linux at the present moment. Despite these inconveniences, an effort has been made to mention several weaker versions of these tests under their corresponding sections.

3.1 Usability and Aesthetics

Overall, the visual appearance of the application was well-received and was universally praised as an improvement over the original *Rogue*. This conclusion was derived from 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 arbitrary key bindings, the Rogue++ team took deliberate actions to improve this area. Specifically, the application featured arrow key bindings for the player character movements in order to accommodate a more standard and intuitive keyboard layout. Additionally, Rogue Reborn featured a convenient help menu inside the game that listed all actionable keys and their corresponding interpretations. However, due to the sheer quantity of key bindings, Rogue Reborn was *not* successful in alleviating this issue completely. With respect to the Test Plan, a report detailing NFRT 3 is given below.

Non-Functional Requirement Test # 3 Report

Execution:	All strings in the Rogue Reborn source code were programmatically extracted and stored in a text file for later inspection with Microsoft Word. The Python script used to populate the text file is located at src/misc/strextract.py.
Results:	The aforementioned script discovered approximately 1400 strings. After manually verifying the grammatical correctness and spelling of each string, 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 exceptional as a consequence of the developers' decision to use C++ as opposed to a less efficient language such as Python or Java. According to *libtcod*, the application managed to average over 120 FPS (Frames Per Second) and delivered a smooth experience even 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 consumed by Rogue Reborn was 1 MB. With respect to the Test Plan, a report detailing NFRT 5 and NFRT 6 is portrayed below.

Non-Functional Requirement Test # 5 Report

Execution:	The Rogue Reborn application was compiled with a special debug parameter that enabled particular sections of the MasterController module to measure the average maximum execution time between successive frames.
Results:	The average execution time between successive user actions appeared to stabilize around 20 ms. Clearly, this is appreciably lower than the maximum allowable delay of MINIMUM_RESPONSE_SPEED (currently $\sim$ 30 ms).

Non-Functional Requirement Test # 6 Report

Execution:	The following regular expression was applied across the source code to extract all integer-typed declarations: `(unsigned|int|long)\s+[A-Za-z]+\s*(,|;|=.*)\s*` The Python script that performed the declaration extraction is located at src/misc/intdeclare.py.
Results:	The aforementioned script identified approximately 170 candidate declarations. Among these, there were no obvious candidates for integer overflow.

3.3 Robustness, Maintainability, and Compatibility

As discussed during the formal presentation, the Rogue Reborn project excelled in the domains of robustness and maintainability. One justification for this claim is that the developers implemented a CI (Continuous Integration) pipeline to immediately flag deficient commits. In addition, compiling the Rogue Reborn source code did not generate any warnings and the system documentation is thorough, expansive, and relevant. That being said, Rogue Reborn has yet to be released in the playtester community. As such, it is likely that several undiscovered bugs still reside within the code base.

Regarding compatibility, the project was significantly less successful: only Linux distributions are supported and SDL (Simple Directmedia Layer) must be installed on the developer's machine to compile the application (SDL). As well, the compilation and execution of Rogue Reborn was only tested on Intel x64 processors (the application is not tested on other brands or architectures). With respect to the Test Plan, a report detailing NFRT 8, NFRT 11, and NFRT 13 is conveyed below.

Non-Functional Requirement Test # 8 Report	
Execution:	The high score file was manually edited by a developer to include more than HIGH_SCORE_CAPACITY records.
Results:	Only the top HIGH_SCORE_CAPACITY records with the highest score were displayed on the RipScreen module; the rest of the records were internally acknowledged but otherwise ignored.

Non-Functional Requirement Test # 11 Report	
Execution:	All documented “Bug” issues on the Rogue Reborn GitLab page were examined to ensure they were closed within 30 days of their creation.
Results:	A total of 10 issues were discovered under the “Bug” label; all of these entries were closed within two weeks of their posting.

Non-Functional Requirement Test # 13 Report	
Execution:	The high score file was manually edited by a developer such that several records violated the expected record format.
Results:	All valid records were processed displayed on the RipScreen module; the nonsensical records were internally acknowledged but otherwise ignored.

3.4 Legality and Safety

Every public software project should include a license to govern the software’s terms of use, development, and distribution. Rogue Reborn was no exception and thus contained the licensing agreement from the original *Rogue*.

Regarding health and safety, the Rogue++ team was primarily concerned with the possibility of inducing a seizure by displaying two successive frames with excessively different contrasts. With respect to the Test Plan, a report detailing NFRT 15 is pictured below.

Non-Functional Requirement Test # 15 Report

Execution:	Two consecutive frames with the largest estimated contrast difference were captured and later processed with a Python script to compute their respective monochrome luminosities. All resources pertaining to this test are located in the src/misc/luminosity directory.
Results:	The output of lumtester.py indicated that the frame illustrated in minlum.png was characterized by an average luminosity of 0.03105, while the maxlum.png frame possessed an average luminosity of 0.59774. Since the difference between these two values is 0.56669 and is therefore less than LUMINOSITY_DELTA , the application is relatively safe for epileptic users (although a cautionary notice should be included in the final distribution package).

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

As a means of verifying the implementation of various functions and components, the Rogue++ developers devised a suite of unit tests. Unit testing was employed as a consequence of its superior ability to localize errors and encourage contributors to write code that is highly modular and relatively free of side effects. Broadly speaking, the Rogue Reborn unit tests were designed to gain confidence in the code with respect to the satisfaction of the functional requirements.

To mitigate the dependencies and significant overhead of integrating a professional unit testing framework, the Rogue Reborn team decided to code their own custom test runner. The implemented solution required all tests to extend an abstract class `Testable` with the interface described in Figure 1. This restriction enabled the test driver `test.main.cpp` to invoke all of the concrete unit test classes in a uniform manner. Finally, to ensure that the unit tests remained relevant and visible throughout development, the CI pipeline included a stage to build, run, and evaluate the output of the testing executable.

Figure 1: **Testable Class Interface**

```
class Testable {
    public:
        virtual Testable();
        virtual ~Testable();

        // Test entry point
        virtual void test() = 0;

        void assert(bool condition, std::string comment);
        void comment(std::string comment);
};
```

Regarding coverage, the tests encompassed all item types and abilities, various player actions and monster mechanics, virtually all level generation and processing algorithms, and even several UI functions. As depicted in Figure 1, each test assertion was also complemented with a comment to explain the purpose of the test. For the convenience of the reader, a descriptive summary of every implemented test is included in the Automated Testing section of this report.

On the whole, these tests served to guarantee a minimal level of functionality for each documented feature across the builds. Naturally, the suite did not consider the non-functional requirements of the project since these qualities lend themselves better to manual testing.

6 Changes Due to Testing

Since the Rogue++ developers did not embrace a TDD (Test-Driven Development) methodology, all of the unit tests were designed to catch flaws in existing code rather than guide the development of new code. As such, the unit testing phase of the Rogue Reborn application did not spawn any new features but instead served to minimize the quantity of programming errors.

One area that the greatly benefited from the unit tests were the Item modules, as numerous bugs were caught with respect to the initialization of particular variables (or lack thereof). For example, there was an error where a disenchanted Armor object would claim a completely nonsensical effective armor value since the enchantment variable was never initialized. Another portion of the tests detected several logic flaws (e.g., the negation of a condition was accidentally checked) present in the effects of certain items. For instance, the *Wand of Cold* was reveled to deal critical damage to ice monsters and almost no damage to fire monsters as a consequence of a negated condition.

With respect to the non-functional testing, the most significant changes arose in the modules that interact with the file system. In particular, the RipScreen module was drastically improved by implementing input validation checks to detect anomalies in the high score record file. Additionally, although the actual testing itself did not compel any changes, various GUI elements in Rogue Reborn were enhanced (such as the help screen and splash screen) in preparation for the usability tests to follow.

Aside from catching implementation mistakes, the testing process also influenced the architecture and design of the software system. To facilitate unit testing, various modules such as PlayState were revamped to improve modularization. Other modules introduced blocks of code that were activated by a special `DEBUG` compiler directive; these sections either displayed debug information to standard output or modified conditionals to guarantee a particular trace through a function. Finally, as mentioned in the Unit Testing section of this Test Report, the CI pipeline of the project was also modified to accommodate the execution of unit tests after each commit to the repository.

7 Automated Testing

7.1 Automated Testing Strategy

With respect to the Unit Testing section of this document, a custom testing framework was developed for the Rogue Reborn project. A series of files following the naming convention `test.<Class Name>.cpp` were created and contained the unit tests specific to the `Class Name` class. Whenever a change was pushed to the Rogue Reborn repository, all of these tests were run and their corresponding output was analyzed by a Python script to determine if any failures occurred.

7.2 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:	Armor Construction 1
Initial State:	None
Input:	Coordinate
Expected Output:	Armor object in valid initial state
Name:	Armor Construction 2
Initial State:	None
Input:	Coordinate, context value, type value
Expected Output:	Armor object in valid initial state
Name:	Armor Identification
Initial State:	Cursed Armor
Input:	None
Expected Output:	Verification that armor is identified
Name:	Armor Curse

Initial State:	Cursed Armor
Input:	None
Expected Output:	Verification that armor is cursed
Name:	Armor Enchantment
Initial State:	Cursed Armor
Input:	Curse level
Expected Output:	Verification that armor enchantment is correct
Name:	Armor Rating
Initial State:	Cursed Armor
Input:	None
Expected Output:	Verification that armor rating is correct
Name:	Coordinate Ordering
Initial State:	None
Input:	(0,0) coordinate and (1,1) coordinate
Expected Output:	Verification that (0,0) ¡ (1,1)
Name:	Coordinate Equality
Initial State:	None
Input:	Two (0,0) coordinates
Expected Output:	Verification that the two inputs are equal
Name:	Coordinate Inequality
Initial State:	None
Input:	(0,0) coordinate and (1,1) coordinate
Expected Output:	Verification that the two inputs are not equal
Name:	Coordinate Addition
Initial State:	None
Input:	(2,3) coordinate and (1,2) coordinate
Expected Output:	(3,5) coordinate
Name:	Coordinate Subtraction
Initial State:	None
Input:	(2,3) coordinate and (1,2) coordinate

Expected Output:	(1,1) coordinate
Name:	Feature Construction
Initial State:	None
Input:	Symbol, coordinate, visibility, color
Expected Output:	Feature object in valid initial state
Name:	Feature Symbol Check
Initial State:	Feature with given symbol
Input:	Symbol
Expected Output:	Verification that feature's symbol matches given
Name:	Feature Invisibility Check
Initial State:	Invisible feature
Input:	None
Expected Output:	Verification that feature is invisible
Name:	Feature Visibility Check
Initial State:	Visible feature
Input:	None
Expected Output:	Verification that feature is visible
Name:	Feature Location Check
Initial State:	Feature with given location
Input:	Coordinate
Expected Output:	Verification that feature's location matches given coordinate
Name:	Food Construction
Initial State:	None
Input:	Coordinate and context value
Expected Output:	Food object in valid initial state
Name:	Food Eating
Initial State:	Food and player objects
Input:	None

Expected Output:	Verification that food has increased the player's food life by an appropriate amount
Name:	GoldPile Construction
Initial State:	None
Input:	Coordinate, gold amount value
Expected Output:	GoldPile object in valid initial state
Name:	GoldPile Quantity Check
Initial State:	GoldPile with given amount of gold
Input:	Amount of gold value
Expected Output:	Verification that gold's amount matches given amount
Name:	Item Construction 1
Initial State:	None
Input:	Symbol, coordinate, context value, item class specifier, name value, psuedoname value, item type specifier, item stackability value, item throwability value, weight value
Expected Output:	Item object in valid initial state
Name:	Item Construction 2
Initial State:	None
Input:	Symbol, coordinate, context value, item class specifier, name value, psuedoname value, item type specifier, item stackability value, item throwability value, weight value
Expected Output:	Item object in valid initial state
Name:	Name Vector Check
Initial State:	None
Input:	Vector of item names
Expected Output:	Shuffled vector of item names
Name:	Item Curse Check
Initial State:	Uncursed item
Input:	None
Expected Output:	Verification that item is uncursed

Name:	Item Curse/Effect Check 1
Initial State:	Uncursed item to which the cursed effect has been applied
Input:	None
Expected Output:	Verification that item is cursed
Name:	Item Curse/Effect Check 2
Initial State:	Cursed item whose curse effect has been removed
Input:	None
Expected Output:	Verification that item is uncursed
Name:	Item Unindentified Check
Initial State:	Identified item
Input:	None
Expected Output:	Verification that item is unidentified
Name:	Item Identified Check
Initial State:	Unidentified item
Input:	None
Expected Output:	Verification that item is identified
Name:	Item Display-Name Check 1
Initial State:	Unidentified item
Input:	Psuedoname
Expected Output:	Verification that item's display name matches psuedoname
Name:	Item Display-Name Check 2
Initial State:	Identified item
Input:	True name
Expected Output:	Verification that item's display name matches true name
Name:	ItemZone Containment Check 1
Initial State:	ItemZone with 2 items
Input:	None

Expected Output:	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:	ItemZone Removal
Initial State:	ItemZone with 2 items
Input:	Removal command
Expected Output:	ItemZone with only second item
Name:	ItemZone Keybind Persistence
Initial State:	ItemZone with first item removed
Input:	None
Expected Output:	Verification that second item is still bound to ’b’
Name:	ItemZone Weight Enforcement
Initial State:	Empty ItemZone
Input:	Attempt to add 500 pieces of armor to ItemZone
Expected Output:	ItemZone with max-weight worth of armor
Name:	Level Construction
Initial State:	None
Input:	Depth, player object
Expected Output:	Level object in valid initial state
Name:	Level Depth Check
Initial State:	Level with given depth
Input:	Depth value
Expected Output:	Verification that level’s depth matches given value
Name:	Level BFSPerp Diagonal Small
Initial State:	Empty level object
Input:	Pair of coordinates diagonally adjacent
Expected Output:	Path between coordinates with expected length, utilizing taxicab movement
Name:	Level BFSPerp Horizontal
Initial State:	Empty level object
Input:	Pair of coordinates with equal y-values

Expected Output:	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:** **Expected Output:**	Level BFSPerp Diagonal Empty level object Pair of coordinates on diagonal line 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:	Level Path Generation
Initial State:	Player object and generated level
Input:	Series of path requests between random coordinates
Expected Output:	Valid paths between locations
Name:	Level Connectedness
Initial State:	Player object and generated level
Input:	Series of path requests between all rooms in the level
Expected Output:	Valid paths between each room
Name:	Level Staircase Check
Initial State:	Player object and generated level
Input:	None
Expected Output:	Verification that level contains a staircase
Name:	Level GoldPile Check
Initial State:	Player object and generated level
Input:	None
Expected Output:	Verification that level contains at least one goldpile
Name:	Monster Construction
Initial State:	None
Input:	Symbol, coordinate, armor value, HP value, exp value, level value, maxHP value, name value
Expected Output:	Monster object in valid initial state
Name:	Dice-Math 1
Initial State:	None
Input:	1 1-sided die
Expected Output:	Sum of values of 1
Name:	Dice-Math 2
Initial State:	None
Input:	2 1-sided die
Expected Output:	Sum of values of 2

Name:	Dice-Math 3
Initial State:	None
Input:	1 2-sided die
Expected Output:	1 ¡= Sum of values ¡= 2
Name:	Dice-Math 4
Initial State:	None
Input:	3 4-sided die
Expected Output:	3 ¡= Sum of values ¡= 12
Name:	Mob Armor Check
Initial State:	Mob object
Input:	None
Expected Output:	Verification mob armor is in valid range
Name:	Mob HP Check 1
Initial State:	Mob with given HP value
Input:	HP value
Expected Output:	Verification mob has correct HP value
Name:	Mob MaxHP Check
Initial State:	Mob with given MaxHP value
Input:	MaxHP value
Expected Output:	Verification mob has correct MaxHP value
Name:	Mob Level Check
Initial State:	Mob with given level value
Input:	Level value
Expected Output:	Verification mob has correct level value
Name:	Mob Location Check
Initial State:	Mob with given location
Input:	Coordinate
Expected Output:	Verification mob has correct location
Name:	Mob Name Check

Initial State:	Mob with given name
Input:	Name value
Expected Output:	Verification mob has correct name
Name:	Mob setMaxHP
Initial State:	Mob with default MaxHP
Input:	setMaxHP command with MaxHP value
Expected Output:	mob with given MaxHP value
Name:	Mob setcurrentHP
Initial State:	Mob with default currentHP
Input:	setCurrentHP command with currentHP value
Expected Output:	mob with given currentHP value
Name:	Mob Dead Check 1
Initial State:	Living Mob object
Input:	None
Expected Output:	Verification mob is alive
Name:	Mob HP Check 2
Initial State:	Living Mob object
Input:	Hit command for ¿¿¿ mob's current HP
Expected Output:	Verification mob has HP ¡= 0
Name:	Mob Dead Check 2
Initial State:	Dead mob object
Input:	None
Expected Output:	Verification mob is dead
Name:	Monster Construction
Initial State:	None
Input:	Symbol, coordinate
Expected Output:	Monster object in valid initial state
Name:	Monster Flag/Invisibility
Initial State:	Visible monster object
Input:	SetFlag command to make monster invisible

Expected Output:	Invisible monster object
Name:	Monster Aggrevate
Initial State:	Idling, sleeping monster object
Input:	Aggrevate command
Expected Output:	Awake, chasing monster object
Name:	Monster Damage Calculation
Initial State:	Monster object
Input:	calculateDamage command
Expected Output:	Correct amount of damage
Name:	Monster Hit Chance
Initial State:	Monster and player objects
Input:	calculateHitChange command
Expected Output:	Hit chance in valid range
Name:	Monster Armor Check
Initial State:	Monster object
Input:	None
Expected Output:	Verification that monster armor is in valid range
Name:	Invisible Monster Name Check
Initial State:	Invisible uonster object
Input:	None
Expected Output:	Verification monster has hidden name
Name:	Visible Monster Name Check
Initial State:	Invisible monster object
Input:	RemoveFlag command to make monster invisible
Expected Output:	Verification monster has real name
Name:	Monster Symbol/Level Association
Initial State:	None
Input:	Depth value

Expected Output:	Set of symbols for monsters that are valid candidates for given depth
Name:	Monster Symbol/Treasure/Level Association
Initial State:	None
Input:	Depth value
Expected Output:	Set of symbols for monsters that are valid candidates for given depth for a treasure room
Name:	PlayerChar Initial Amulet Check
Initial State:	Just initialized playerchar object
Input:	None
Expected Output:	Verification the game does not believe the player has the amulet
Name:	PlayerChar Initial HP Check
Initial State:	Just initialized playerchar object
Input:	None
Expected Output:	Verification playerchar has full hp
Name:	PlayerChar Level-Up Exp
Initial State:	Playerchar object at initial level
Input:	Exp input into playerchar object
Expected Output:	Playerchar object with increased level
Name:	PlayerChar Level-Up Manual
Initial State:	Playerchar object
Input:	Level-up command
Expected Output:	Playerchar object with increased level
Name:	PlayerChar Damage
Initial State:	Playerchar object at full hp
Input:	Series of damage commands applied to playerchar object
Expected Output:	Playerchar object with less than full hp
Name:	PlayerChar UnArmed 1
Initial State:	Unarmed playerchar object

Input:	calculateDamage command
Expected Output:	0 damage value
Name:	PlayerChar Armed
Initial State:	Playerchar object armed with weapon
Input:	calculateDamage command
Expected Output:	Damage value ¿ 0
Name:	PlayerChar Stow Weapon
Initial State:	Playerchar object armed with uncursed weapon
Input:	removeWeapon command
Expected Output:	PlayerChar object unarmed
Name:	PlayerChar UnArmed 2
Initial State:	Armed playerchar object
Input:	removeWeapon command, then calculateDamage
Expected Output:	0 damage value
Name:	PlayerChar Remove Non-Armor
Initial State:	Playerchar object with no armor
Input:	removeArmor command
Expected Output:	Boolean indicating failure to remove armor
Name:	PlayerChar Remove Armor
Initial State:	Playerchar object with uncursed armor
Input:	removeArmor command
Expected Output:	Playerchar object without armor
Name:	Potion Construction 1
Initial State:	None
Input:	Coordinate
Expected Output:	Potion object in valid initial state
Name:	Potion Construction 2
Initial State:	None
Input:	Coordinate, item context value, item type specifier

Expected Output:	Potion object in valid initial state
Name:	Potion of Strength
Initial State:	Player object
Input:	Potion of strength
Expected Output:	Player with strength increased by 1
Name:	Potion of Restore Strength
Initial State:	Player object with reduced strength
Input:	Potion of restore strength
Expected Output:	Player object with pre-reduction strength
Name:	Potion of Healing
Initial State:	Player object with full hp
Input:	Potion of healing
Expected Output:	Player object with maxHP increased by 1
Name:	Potion of Extra Healing
Initial State:	Player object with full hp
Input:	Potion of extra healing
Expected Output:	Player object with maxHP increased by 2
Name:	Potion of Poison
Initial State:	Player object with strength ¿ 0
Input:	Potion of poison
Expected Output:	Player object with reduced strength
Name:	Potion of Raise Level
Initial State:	Player object with less than max level
Input:	Potion or raise level
Expected Output:	Player object with level + 1
Name:	Potion of Blindness
Initial State:	Player object without the blindness condition
Input:	Potion of blindness
Expected Output:	Player object with the blindness condition

Name:	Potion of Hallucination
Initial State:	Player object without the hallucination condition
Input:	Potion of hallucination
Expected Output:	Player object with the hallucination condition
Name:	Potion of Detect Monster
Initial State:	Player object without the detect-monsters condition
Input:	Potion of detect monsters
Expected Output:	Player object with the detect-monsters condition
Name:	Potion of Detect Object
Initial State:	Player object without the detect-objects condition
Input:	Potion of detect objects
Expected Output:	Player object with the detect-objects condition
Name:	Potion of Confusion
Initial State:	Player object without the confusion condition
Input:	Potion of confusion
Expected Output:	Player object with the confusion condition
Name:	Potion of Confusion
Initial State:	Player object without the confusion condition
Input:	Potion of confusion
Expected Output:	Player object with the confusion condition
Name:	Potion of Levitation
Initial State:	Player object without the levitation condition
Input:	Potion of levitation
Expected Output:	Player object with the levitation condition
Name:	Potion of Haste
Initial State:	Player object without the haste condition
Input:	Potion of haste
Expected Output:	Player object with the haste condition
Name:	Potion of See-Invisible

Initial State:	Player object without the invisible-sight condition
Input:	Potion of invisible
Expected Output:	Player object with the invisible-sight condition
Name:	Random Range 1
Initial State:	None
Input:	Upper and lower bounds 0,0
Expected Output:	0
Name:	Random Range 2
Initial State:	None
Input:	Upper and lower bounds 5,5
Expected Output:	5
Name:	Random Range 3
Initial State:	None
Input:	Upper and lower bounds 0,60, repeated 40 times
Expected Output:	0 ¡= result ¡= 60
Name:	Random Float
Initial State:	None
Input:	40 repeats
Expected Output:	0 ¡= result ¡= 1
Name:	Random Boolean
Initial State:	None
Input:	10 repeats
Expected Output:	Both true and false are generated
Name:	Random Percent
Initial State:	None
Input:	40 repeats
Expected Output:	0 ¡= result ¡= 100
Name:	Random Position
Initial State:	None

Input:	Two coordinates, as top-left and bottom-right of rectangle, 10 repeats
Expected Output:	Random coordinates within the bounds
Name:	Ring Construction 1
Initial State:	None
Input:	Coordinate
Expected Output:	Ring object with valid initial state
Name:	Ring Construction 2
Initial State:	None
Input:	Coordinate, item context value, type identifier
Expected Output:	Ring object with valid initial state
Name:	Ring of Stealth
Initial State:	Player object without stealth condition
Input:	Ring of stealth
Expected Output:	Player object with the stealth condition
Name:	Ring of Stealth Deactivate
Initial State:	Player object with ring of stealth
Input:	Remove ring
Expected Output:	Player object without the stealth condition
Name:	Ring of Teleportation
Initial State:	Player object without random teleportation condition
Input:	Ring of teleportation
Expected Output:	Player object with the random teleportation condition
Name:	Ring of Teleportation Deactivate
Initial State:	Player object with ring of teleportation
Input:	Remove ring
Expected Output:	Player object without the random teleportation condition
Name:	Ring of Regeneration
Initial State:	Player object without regeneration condition

Input:	Ring of regeneration
Expected Output:	Player object with the regeneration condition
Name:	Ring of Regeneration Deactivate
Initial State:	Player object with ring of regeneration
Input:	Remove ring
Expected Output:	Player object without the regeneration condition
Name:	Ring of Digestion
Initial State:	Player object without digestion condition
Input:	Ring of digestion
Expected Output:	Player object with the digestion condition
Name:	Ring of Digestion Deactivate
Initial State:	Player object with ring of digestion
Input:	Remove ring
Expected Output:	Player object without the digestion condition
Name:	Ring of Dexterity
Initial State:	Player object
Input:	Ring of dexterity
Expected Output:	Player object with dexterity increased by the appropriate amount
Name:	Ring of Dexterity Deactivate
Initial State:	Player object with ring of dexterity
Input:	Remove ring
Expected Output:	Player object with normal dexterity
Name:	Ring of Adornment
Initial State:	Player object
Input:	Ring of adornment
Expected Output:	Identical player object
Name:	Ring of Adornment
Initial State:	Player object with ring of adornment
Input:	Remove ring

Expected Output:	Identical player object
Name:	Ring of See-Invisible
Initial State:	Player object without the see-invisible condition
Input:	Ring of see-invisible
Expected Output:	Player object with the see-invisible condition
Name:	Ring of See-Invisible Deactivate
Initial State:	Player object with ring of see-invisible
Input:	Remove ring
Expected Output:	Player object without the see-invisible condition
Name:	Ring of Maintain-Armor
Initial State:	Player object without the maintain-armor condition
Input:	Ring of maintain-armor
Expected Output:	Player object with the maintain-armor condition
Name:	Ring of Maintain-Armor Deactivate
Initial State:	Player object with ring of maintain-armor
Input:	Remove ring
Expected Output:	Player object without the maintain-armor condition
Name:	Ring of Searching
Initial State:	Player object without the auto-search condition
Input:	Ring of searching
Expected Output:	Player object with the auto-search condition
Name:	Ring of Searching Deactivate
Initial State:	Player object with ring of searching
Input:	Remove ring
Expected Output:	Player object without the auto-search condition
Name:	Room Construction Check 1
Initial State:	Randomly generated room
Input:	None
Expected Output:	Verification that room's size is in valid range

Name:	Room Construction Check 2
Initial State:	Randomly generated room
Input:	None
Expected Output:	Verification that room edges are within valid bounds
Name:	Scroll Construction 1
Initial State:	None
Input:	Coordinate
Expected Output:	Scroll object in valid initial state
Name:	Scroll Construction 2
Initial State:	None
Input:	Coordinate, item context value, type identifier
Expected Output:	Scroll object in valid initial state
Name:	Scroll PseudoNames
Initial State:	Scrolls are uninitialized
Input:	initializeScrollNames command
Expected Output:	Vector of valid scroll psuedonames
Name:	Scroll of Protect Armor
Initial State:	Player with cursed armor
Input:	Scroll of protect armor
Expected Output:	Player with uncursed armor with protect-armor effect
Name:	Scroll of Hold Monster
Initial State:	Monster without the held flag
Input:	Scroll of hold monster
Expected Output:	Monster with the held flag
Name:	Scroll of Enchant Weapon
Initial State:	Player with weapon
Input:	Scroll of enchant weapon
Expected Output:	Player with uncursed weapon with higher enchant level
Name:	Scroll of Enchant Armor

Initial State:	Player with armor
Input:	Scroll of enchant armor
Expected Output:	Player with uncursed armor with higher enchant level
Name:	Scroll of Identity
Initial State:	None
Input:	Scroll identity
Expected Output:	No exceptions
Name:	Scroll of Teleportation
Initial State:	Player at coordinate (0,0)
Input:	Scroll of teleportation
Expected Output:	Player at coordinate != (0,0)
Name:	Scroll of Sleep
Initial State:	Player without the sleep condition
Input:	Scroll of sleep
Expected Output:	Player with the sleep condition
Name:	Scroll of Scare Monster
Initial State:	None
Input:	Scroll of scare monster
Expected Output:	No exceptions
Name:	Scroll of Remove Curse
Initial State:	Player with cursed weapon
Input:	Scroll of remove curse
Expected Output:	Player with uncursed weapon
Name:	Scroll of Create Monster
Initial State:	Level object
Input:	Scroll of create monster
Expected Output:	Level with 1 additional monster
Name:	Scroll of Aggravate Monster
Initial State:	Level with sleeping monsters
Input:	Scroll of aggravate monster

Expected Output:	Level with no sleeping monsters
Name:	Scroll of Magic Mapping
Initial State:	Unrevealed level
Input:	Scroll of magic mapping
Expected Output:	Level where all tiles have been revealed
Name:	Scroll of Confuse Monster
Initial State:	Player without the confuse-monster condition
Input:	Scroll of confuse monster
Expected Output:	Player with the confuse-monster condition
Name:	Stair Construction
Initial State:	None
Input:	Coordinate, direction value
Expected Output:	Stair object in valid initial state
Name:	Stair Direction Check
Initial State:	Stair constructed with direction
Input:	Direction value
Expected Output:	Verification stair has given direction value
Name:	Floor Passability Check
Initial State:	Floor object
Input:	None
Expected Output:	Verification floor is passable
Name:	Floor Symbol Check
Initial State:	Floor object
Input:	None
Expected Output:	Verification floor has correct symbol
Name:	Floor Transparency Check
Initial State:	Floor object
Input:	None
Expected Output:	Verification floor is transparent

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

Initial State:	Door object
Input:	None
Expected Output:	Verification corridor has correct symbol
Name:	Door Transparency Check
Initial State:	Door object
Input:	None
Expected Output:	Verification Door has special corridor transparency
Name:	Door Trap
Initial State:	Player and level
Input:	Door trap
Expected Output:	Player at a level with depth + 1
Name:	Rust Trap
Initial State:	Player with enchanted weapon
Input:	Rust trap
Expected Output:	Player with unenchanted weapon
Name:	Sleep Trap
Initial State:	Player without the sleep condition
Input:	Sleep trap
Expected Output:	Player with the sleep condition
Name:	Bear Trap
Initial State:	Player without the immobilized condition
Input:	Bear trap
Expected Output:	Player with the immobilized condition
Name:	Teleport Trap
Initial State:	Player
Input:	Teleport trap
Expected Output:	Player at a different location
Name:	Dart Trap
Initial State:	Player
Input:	Dart trap

Expected Output:	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:** **Input:** **Expected Output:**	Open Status Screen Playstate, player, empty level Status key 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:	Zap Wand Select
Initial State:	Playstate, player with wand, empty level
Input:	Zap key, then direction key
Expected Output:	Inventory Screen
Name:	Zap Wand Fire 1
Initial State:	Inventory wand select
Input:	Item select hotkey
Expected Output:	Playstate
Name:	Zap Wand Fire 2
Initial State:	Inventory wand select
Input:	Item select hotkey
Expected Output:	wand with charges - 1
Name:	Game Quit
Initial State:	Playstate
Input:	Quit key and confirmation key
Expected Output:	RIPScreen
Name:	Wand Construction 1
Initial State:	None
Input:	Coordinate
Expected Output:	Wand in valid initial state
Name:	Wand Construction 2
Initial State:	None
Input:	Coordinate, item context value, type specifier
Expected Output:	Wand in valid initial state
Name:	Wand of Teleport Away
Initial State:	Player, nearby monster
Input:	Wand of teleport away
Expected Output:	Monster has distance to player ¿= 20
Name:	Wand of Slow Monster

Initial State:	Player, monster without slowed flag
Input:	Wand of slow monster
Expected Output:	Monster has slowed flag
Name:	Wand of Invisibility
Initial State:	Player, monster without invisible flag
Input:	Wand of invisibility
Expected Output:	Monster with invisible flag
Name:	Wand of Polymorph
Initial State:	Player, monster
Input:	Wand of polymorph
Expected Output:	Different monster at previous monster's locations
Name:	Wand of Haste Monster
Initial State:	Player, monster without haste flag
Input:	Wand of haste monster
Expected Output:	Monster with haste flag
Name:	Wand of Magic Missile
Initial State:	Player, monster
Input:	Wand of magic missile
Expected Output:	Monster with reduced hp
Name:	Wand of Cancellation
Initial State:	Player, monster without cancelled flag
Input:	Wand of cancellation
Expected Output:	Monster with cancelled flag
Name:	Wand of Do Nothing
Initial State:	Player, monster
Input:	Wand of do nothing
Expected Output:	No exceptions
Name:	Wand of Drain Life
Initial State:	Player with reduced health, monster
Input:	Wand of drain life

Expected Output:	Player with increased health, monster with reduced health
Name:	Wand of Cold
Initial State:	Player, monster
Input:	Wand of cold
Expected Output:	No exceptions
Name:	Wand of Fire
Initial State:	Player, monster
Input:	Wand of fire
Expected Output:	No exceptions
Name:	Weapon Construction 1
Initial State:	None
Input:	Coordinate
Expected Output:	Weapon in valid initial state
Name:	Weapon Construction 2
Initial State:	None
Input:	Coordinate, item context value, type specifier
Expected Output:	Weapon in valid initial state
Name:	Weapon Identification Check
Initial State:	Identified weapon
Input:	None
Expected Output:	Verification that weapon is identified
Name:	Weapon Curse Check
Initial State:	Cursed weapon
Input:	None
Expected Output:	Verification that weapon is cursed
Name:	Weapon Name Check
Initial State:	Weapon
Input:	None

Expected Output:	Verification that weapon has valid name
Name:	Weapon Enchantment Check
Initial State:	Cursed weapon
Input:	None
Expected Output:	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 Module	BasicIO	**M1**
	Doryen	**M2**
	Input Format	**M3**
Behaviour-Hiding Module	External	**M4**
	Item	**M5**
	Level	**M6**
	LevelGen	**M7**
	MainMenu	**M8**
	Mob	**M9**
	Monster	**M10**
	PlayerChar	**M11**
	RipScreen	**M12**
	PlayState	**M13**
	SaveScreen	**M14**
	UIState	**M15**
Software Decision Module	Coord	**M16**
	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 non-functional 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.

References

About SDL. https://www.libsdl.org/index.php. Accessed: December 7, 2016.