SE 3XA3: Test Plan
Rogue Reborn

Group #6, Team Rogue++

Ian Prins	prinsij
Mikhail Andrenkov	andrem5
Or Almog	almogo

Due Wednesday, Dec 7st, 2016

Contents

List of Tables

List of Figures

This document...

1 Functional Requirements Evaluation

Ori

2 Nonfunctional Requirements Evaluation

Mikhail

2.1 Usability

Mikhail

2.2 Performance

Mikhail

2.3 etc.

Mikhail

3 Comparison to Existing Implementation

Ori

4 Unit Testing

Mikhail

Table 1: **Revision History**

Date	**Version**	**Notes**
Dec 6	0.1	Initial draft

5 Changes Due to Testing

Mikhail

6 Automated Testing

6.1 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.

6.2 Specific System Tests

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

Name:	
Initial State:	
Input:	
Expected Output:	

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 Identification
Initial State:	Cursed Armor
Input:	None
Expected Output:	Verification that armor is identified

7 Trace to Requirements

Ori

8 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.

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

9 Code Coverage Metrics

Ori

Table 2: **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**
	Monster	**M9**
	PlayerChar	**M10**
	RipScreen	**M11**
	PlayState	**M12**
	UIState	**M13**
Software Decision Module	Coord	**M14**
	Feature	**M15**
	ItemZone	**M16**
	MasterController	**M17**
	Mob	**M18**
	Random	**M19**
	Terrain	**M20**

Table 3: **Test-Module Trace**

File	Related Module(s)
test.amulet.cpp	Related modules here
test.armor.cpp	Related modules here
test.coord.cpp	Related modules here
test.feature.cpp	Related modules here
test.food.cpp	Related modules here
test.goldpile.cpp	Related modules here
test.item.cpp	Related modules here
test.itemzone.cpp	Related modules here
test.level.cpp	Related modules here
test.levelgen.cpp	Related modules here
test.main.cpp	Related modules here
test.mob.cpp	Related modules here
test.monster.cpp	Related modules here
test.playerchar.cpp	Related modules here
test.potion.cpp	Related modules here
test.ring.cpp	Related modules here
test.room.cpp	Related modules here
test.scroll.cpp	Related modules here
test.stairs.cpp	Related modules here
test.terrain.cpp	Related modules here
test.testable.cpp	Related modules here
test.testable.h	Related modules here
test.trap.cpp	Related modules here
test.tunnel.cpp	Related modules here
test.uistate.cpp	Related modules here
test.wand.cpp	Related modules here
test.weapon.cpp	Related modules here