Chapter 1

TEST PLAN

Version 1.1
May 9, 2014

FOREWORD

FOR

Swords and Sorcery

Prepared for:
Software Engineering: CS 383
Dr. Jeffery

Prepared by:
Keith Drew, Ian Westrope, Jonathan Flake, Colin Clifford, David Klingenberg, Sean Shepherd, Chris, Gabe, John, Wayne, Tao Zhang, Cameron Simon, Matthrew Brown, Tyler Jaszkowiak, Ian Westrope, ChiHsiang Wang

University of Idaho
Moscow, ID 83844-1010

RECORD OF CHANGES (Change History)

Change Number	Date completed	Location of change (e.g., page or figure #)	A M D	Brief description of change	Approved by (initials)	Date approved
1	April, 2014	Original HUD team document.	A	Original HUD team document.	HUD Team	April, 2014
2	May 5, 2014	Entire Document	M	Concatenated all team documents together.	DRK	May 5, 2014

A - ADDED M - MODIFIED D – DELETED

TABLE OF CONTENTS

Section Page

1 TEST PLAN IDENTIFIER

TestPlan 1.5

2 REFERENCES

Use cases and UML diagrams are available on the class website, as well as game rules.
http://www2.cs.uidaho.edu/ jeffery/courses/383/
The code itself can be found at https://github.com/cjeffery/sworsorc/tree/master/src
The automated unit tests are located at https://github.com/cjeffery/sworsorc/tree/master/src/test

3 INTRODUCTION

3.1 HUD View

Our plan is to integrate the working hud/game code and test that it works in tangent, as opposed to only working in discrete locations of the project. We will be using junit tests to evaluate discrete methods and manual tests to test the GUI and game logic. To plan and document manually test for GUI components as they are developed. As new code is integrated the unit tests will be rerun to ensure integration did not break any component. Manual tests will be rerun on a case-by-case basis. All tests will be run the day before the end of a sprint and before deliverable presentation.

3.2 Rules View

The purpose of this test plan is to state the processes used by Game Rules and Play Team in testing the Spells and Characters for the Swords & Sorcery project.(Tao, Cameron)

This test plan covers the creation and implementation of the Moveableunit class and its sub class Armyunits. (Matt)

This test plan is also to provide as much coverage as possible to the methods and data of the Scenario class by verifying the results of execution against expectations through a series of automated unit tests and a manual GUI test. (Tyler)

3.3 T

his test plan for the networking portion of the project. It will outline how we test networking, and how we'll test integration of networking into other code.

4 TEST ITEMS

The army units will be tested for the proper member variable values as well as proper results from the member function of the Moveableunit and Armyunit classes. Variables will tested at creation and members variable will be tested for proper results when applicable. Some example tests will be that the location member variable is properly set during movement. Also another type of test that will be preformed is that conjured units are properly created and and placed in the proper place on the game board.

4.1 Character

- Class Interfaces
- Class Interactions
- Spells Implementation
- Character

4.2 Scenario

The data read by the Scenario test will be read from one of the simple scenario configuration files and then verified against expectations through the class's accessor methods. These data items include

- The scenario's name
- Number of players
- Game length
- The blue sun's initial position
- The names of armies in the scenario
- The controlling players of these armies
- The setup order of these armies
- The movement order of these armies
- Names of nations within these armies
- Names of the neutrals in the scenario
- The provinces controlled by a nation
- The characters within a nation
- The units within a player nation
- The units within a neutral nation
- The races of both neutrals and player nations
- The reinforcement and replacement description strings
- Data related to where a neutral is leaning toward
- Whether or not a neutral accepts human sacrifice

Unfortunately, the most complex functionality of the Scenario class cannot be tested by automated unit tests. The unit pool populator requires a manual check.

4.3 Network

- Client connects and disconnects from the server.
- Server detects new client connections.
- Client and server can exchange messages.
- Client and server are able to identify message type and content.
- Client messages are received by related clients.
- Network events trigger GUI events properly.

4.4 Additional items to be tested.

- GUI Tests
  - Hexes/Units tile ok, look ok
  - Mouse clicks work
  - Menu navigation
  - Game starting (networked, scenario)
  - General gameplay (to the extent it's implemented)
  - Unit movement
  - Other
- Other manual tests (game compiles and runs on different platforms)
- Auto tests - Junit
- Ensure all files under resources can be loaded (if feasible)
- The user can initiate connection and disconnection from the server.
- The user can... (Network stuff)

5 SOFTWARE RISK ISSUES

Software to be tested includes the following:

1. User's network configuration prevents networking.
2. GUI - Doesn't load working map, can't support unit movement
3. JSON Library - Loads scenario incorrectly, if not at all
4. Incorporating Networking with GUI and game logic may be difficult
5. Undetected Logic Errors (ULEs)
6. Misinterpretation of Rules
7. Connection Failures: For whatever reason, we have no access to a server, or can't connect over user's network

8. Networking code is unable to connect with other code elements.
9. Networking code can be integrated, but is too complicated for anyone to work with.
10. Improper casting of units
11. Unit ID's interpreted incorrectly
12. Depends on Java's JSON reader and the programmer's understanding of it
13. Complex data structures such as a map of maps
14. Complex loops to iterate over data structures
15. Poor documentation surrounding some of these iterators
16. Poor documentation in general
17. 3rd party library's. Including json-simple-1.1.1 and controlsfx-8.0.5

6 FEATURES TO BE TESTED

6.1 Scenario

From the user's perspective, much of this data reading occurs under the hood. In all cases except for populating the unit pool, the Scenario class does not manipulate any pieces of the rest of the project. Other components read the data from the Scenario class. Therefore, while the solar configuration relies on a properly-working Scenario class, this is not apparent to the user because solar configuration is also handled by SolarConfig and HUDInitializer classes.

Therefore, the only visible component being tested by this plan is the placement of units and characters into the correct provinces of the map.

This is a listing of what is NOT to be tested from both the Users viewpoint of what the system does and a configuration management/version control view. This is not a technical description of the software, but a USERS view of the functions.

All data is verified, but testing the Scenario class alone cannot ensure that it reaches the HUD successfully. Integration tests are required outside of this plan for information such as solar configuration, move order, setup order, game length, and diplomacy.

What is not tested is the count of the units on the map that correspond to the scenario loaded. Also the type of unit is not test, this is assumed correct.

6.2 Movement

- conjured unit appear when casted
- units are properly represented on HUD
- movement location is correct on map
- moral status is properly updated after combat

6.3 Character, Spells

- Select character on GUI
- Select Spell
- Effects of Partial Spells
- Manna Costing

6.3.1 Character

Rule Description	Test Description	Expected Result
Create new character object with its information.	createCharacter function is called with specific character name in CharacterMaker.java.	A new character object is created and returned that contains all of the specified characters information.
Potential spells for selected character displayed.	Select character in GUI, then click cast spell button on sidebar panel.	List of spells that can be cast by selected character are displayed.

6.3.2 Spells

Rule Description	Test Description	Expected Result
Character with Power Level should have a spell book.	Generate a spell book for the character.	A frame with a list of spells should be shown on the screen.
Show spell description.	Click on the Spell button.	A frame will be displayed with all information about the spell.
A target need to be selected for most of the spells.	Click on cast button on the frame of displaying information about the spell. Then select a target by right click the target on the map.	A target is selected to cast the spell.

6.4 Additional Features To Be Test

- 1 Complete Turn
- Movement
- Teleporting
- Network
- Loading

- Solar Display
- Diplomacy Display

7 FEATURES NOT TO BE TESTED

All data is verified, but testing the Scenario class alone cannot ensure that it reaches the HUD successfully. Integration tests are required outside of this plan for information such as solar configuration, move order, setup order, game length, and diplomacy.

What is not tested is the count of the units on the map that correspond to the scenario loaded. Also the type of unit is not test, this is assumed correct.

- Full game
- Everything in backlog

7.1 Network

JUnit tests will be used to test both sides of the network (that is, the client and the server networking code), in isolation.

Manual testing will be needed to ensure integration of the networking code with elements of the graphical user interface, as well as testing that the client and server work over physically distinct machines. Because of the limited resources for manual testing, manual testing will test a subset of possible interactions, with the assumption that core network obstacles will block all messages (i.e. no connection can be made over the network), or no messages. The graphic effects of network events will also be tested manually.

8 APPROACH

The plan is to use Junit tests and manual tests to ensure that our code follows the rules of the game and works itself. Junit tests are done according to the individuals who have developed the methods being tested, and those are not listed here. However, they should be able to be run together and work. The manual tests will work as though a mock player (tester) is running the game. They should be able to select a unit, move a unit, teleport a unit, advance and view the solar display, send chat messages, and view the diplomacy display. Functionalities of each display should also be tested. For example, a unit with a move allowance should only be viewed moving at or under their limit and a chat message should be sent from one user and be seen by all users.

8.1 Network

JUnit tests will be used to test both sides of the network (that is, the client and the server networking code), in isolation.

Manual testing will be needed to ensure integration of the networking code with elements of the graphical user interface, as well as testing that the client and server work over physically distinct machines. Because of the limited resources for manual testing, manual testing will test a subset of possible interactions, with the assumption that core network obstacles will block all messages (i.e. no connection can be made over the network), or no messages. The graphic effects of network events will also be tested manually.

9 ITEM PASS/FAIL CRITERIA

All tests should meet the specifications of the Swords and Sorcery board game rules manual, as well as follow logical design patterns and language rules/conventions. The manual tests of movement should fall within 20% of total movement possibilities the movement rules indicate.

10 SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

- If the github project is broken or otherwise compromised, productive development and testing would be suspended for a time. To resume, fix whatever has been broken on github.
- If any component test fails, the larger dependent pieces are unable to be tested until the component is fixed. In such a case, fix the component and continue testing.

11 TEST DELIVERABLES

- Test plan document
- Test cases
- Relevant error logs or problem reports
- Possible solutions

12 REMAINING TEST TASKS

There are many remaining test tasks, as we have not achieved a working game yet. Remaining tests include full combat, spell casting, scenario loading, full gameplay, etc. We also anticipate unexpected integration requirements to be added to the test plan.

13 ENVIRONMENTAL NEEDS

In the final test, we'll need a remote machine to host the server code. Along with two computers connected over the internet too the server. Otherwise, we cannot test full networking capabilities and full gameplay as intended.

14 RESPONSIBILITIES

The hierarchy of "inchargeness" is as follows:

1. Dr. J
2. John Goettsche
3. Everyone else

15 SCHEDULE

Junit tests should be implemented and run as functionality is added to classes and packages. Manual testing will be done as the GUI is developed and functionality added, as well as when more aspects of other group's code are added. Test to be rerun before demonstration day.

16 PLANNING RISKS AND CONTINGENCIES

As the end of the semester is a fixed date, there are no contingencies for failure. Public beatings will be carried out as needed.

17 APPROVALS

The only person capable of fully approving any fixes/modifications is Dr. J.

18 GLOSSARY

Junit test is a discrete code test designed to be ran as part of an automated test sequence that tests all Junit tests.

19 APPENDIX A. Example JUnit Test

```
@Test
public void testStackWarning() {

        boolean test;
        pool.addUnit(1, new Bow(), "0101");
        pool.addUnit(1, new Bow(), "0101");

        test = HexStack.overStackWaring(pool.getUnitsInHex("0101"),false);

        assertFalse(test);

        pool.addUnit(1, new Bow(), "0101");

        test = HexStack.overStackWaring(pool.getUnitsInHex("0101"),false);
        assertTrue(test);
}
```

20 APPENDIX B. Example Manual Test

This test will ensure that the graphical components of the over stack warning in the stack class is behaving as designed.

1. Start with any hex that has one unit in it.
2. Add one unit to the stack. No warning should be triggered.
3. Add one unit to the stack a warning should be triggered.
4. Close the over stack warning box.
5. Remove one unit from the stack. No warning should be triggered.
6. Remove one unit from the stack. No warning should be triggered.
7. Add one unit to the stack. No warning should be triggered.
8. Add one unit to the stack a warning should be triggered.
9. add one unit to the staff a warning should be triggered indicating there are 2 too many units in the stack.

The test pass if all steps are successfully completed. Otherwise, test fails.

21 APPENDIX C. Individual Team Test Plans

All individually developed tests to be found at the following link:
https://github.com/cjeffery/sworsorc/tree/master/doc/TestPlans