NTJ

FreeCol

Software Quality Assurance Plan

Version: (5) **Date: (05/01/2016)**

Document History and Distribution

1. Revision History

Revision #	Revision Date	Description of Change	Author
0	5/1/2016	Initialize document	James Rehak
1	5/8/2016	Add Introduction, scope, component testing	newsha
2	5/15/2016	Update Features and bugs	James Rehak
3	5/15/2016	Update tables, ..	newsha
4	5/15/2016	Update tables and descriptions	Tao Qiu
5	5/16/2016	Final Revisions	Newsha, James

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

This Software Quality Assurance Plan (SQAP) sets forth the process, methods, standards, and procedures that will be used to perform the Software Quality Assurance function for the *FreeCol* project. The plan identifies the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.

1.1 Objectives

The primary objective of this test plan is to improve the maintainability of the Freecol source code by cleaning up coding deficiencies. Our secondary goal is to improve test coverage for the packages that are the least tested.

1.2 Testing Strategy

As stated in the primary objective, the goal of this plan was to improve maintainability of the *Freecol* game. This was achieved by using various tools (stated in section 8) to analyze the source code. Based on their results we selected the packages with the worse test code coverage and some packages that contained classes which had average Cyclomatic Complexity greater than 7. In addition to the refactoring of the selected packages, any source code which contained high priority bugs or violations were corrected.

Our testing strategy utilized a combination of testing and refactoring tools, some of which were built into the Eclipse IDE and other with were available via plug in.

1.3 Scope

The use of this plan will help assure the following:

(1) That software evaluation and acceptance standards are developed, documented and followed.
(2) That test results adhere to acceptance standards.
(3) All modified software will be well documented with updated Javadocs.

1.5 Definitions and Acronyms

Here is the definitions of all terms and agency acronyms used in this document:

SQAP = Software Quality Assurance Plan
STP = Software Test Plan
CC = Cyclomatic Complexity
Pkg = package
ACC = Average Cyclomatic Complexity

2. TEST ITEMS

The test items included in this plan are consisted of two packages which were identified to be more critical and important to be tested based on primarily studies that the testing group conducted and these are described in next section in details.

2.1 Program Modules

All modules to be tested will be analyzed for potential bugs, audit violations, and design smells. Modules are listed in section 3.

3. FEATURES TO BE TESTED

The following packages listed in *table 1* were selected to be tested and improved. Their selection was based on static source code analysis using the code pro tools test suite. The following classes listed had very little to no code coverage and in some cases had high cyclomatic complexity.

In addition to correcting the classes listed below, the entire project was checked for bugs using the findbugs tool and codepro audit tool. All severe (scariest) and high danger (scary) bugs identified with high confidence were corrected as well has high priority violations. See *table 2* for identified bugs and violations.

Feature	Package Name	Class Name
James:		
1	net.sf.freecol.server.generator	SimpleMapGenerator.java

2	net.sf.freecol.server.generator	TerrainGenerator.java
3	net.sf.freecol.server.generator	ColonizationMapLoader.java
4	net.sf.freecol.tools	DesktopEntryTest.java
5	net.sf.freecol.tools	ForestMaker.java
6	net.sf.freecol.tools	ColonizationMapReaderTest.java
Newsha:		
7	net.sf.freecol.tools	ColonizationSaveGameReader.java
8	net.sf.freecol.tools	MapConverter.java
9	net.sf.freecol.tools	InstallerTranslations.java
10	net.sf.freecol.tools	RiverMaker.java
11	net.sf.freecol.tools	SaveGameValidator.java
Tao:		
12	net.sf.freecol.server.model	ServerColony.Java
13	net.sf.freecol.server.model	ServerUnit.Java
14	net.sf.freecol.tools	ColonizationMapReader.Java
15	net.sf.freecol.tools	FSGConvert.Java
16	net.sf.freecol.tools	MergeTranslation.Java

Table 1. Tested classes

Bugs to be corrected:

Item #	**Description**	**Affected File**	**Priority**	**Issue Tracker**
1	Project import issue	Project	High	1
2	High Cyclomatic Complexity	TranslationReport.java	High	2

3	High Cyclomatic Complexity	ForestMaker.java	High	3
4	High Cyclomatic Complexity	ColonizationMapReader.java	High	4
5	Bug: Possible null pointer dereference	ServerPlayer.java	Normal	5
6	Bug: Possible null pointer dereference	AIMain.java	Normal	6
7	Bug: Possible null pointer dereference	MergeTranslations.java	Normal	7
8	Bug: Possible null pointer dereference	SaveGameValidator.java	Normal	8
9	Bug: Possible null pointer dereference	TranslationReport.java	Normal	9
10	Duplicate property name	FreeColMessages.properties	High	10
11	Empty Statement	NameCache.java	High	11
12	Empty Statement	PathNode.java	High	11
13	Empty Statement	AIMain.java	High	11
14	Empty Statement	ServerPlayer.java	High	11
15	Environment Variable Access	FreeColDirectories.java	High	12
16	Nullcheck of old location of value previously dereferenced	ServerUnit.java	High (Scary level)	13
17	Nullcheck of old location of value previously dereferenced	InGameController.java	High (Scary level)	13
18	Integer is incompatible with expected argument type	MetaServer.java	Severe (Scariest level)	14

19	Suspicious comparison of Integer references	Limit.java	Severe (Scariest level)	14
20	Suspicious comparison of Integer references	RangeOption.java	Severe (Scariest level)	14
21	Suspicious comparison of Integer references	SelectionOption.java	Severe (Scariest level)	14

Table 2: High priority bugs and violations

4. FEATURES NOT TO BE TESTED

Due to the time restraints, all other packages were ignored for in depth improvement.

5. APPROACH

The overall approach to testing for this project is employing unit testing, integration testing, and regression testing. For each level of testing there is a test plan and appropriate deliverables that are introduced in next pages.

5.1 Component Testing

All code will be unit tested to ensure that the individual unit (class) performs the required functions and outputs the proper results and data.

1. Logic Complexity: Use the Cyclomatic complexity matrix index which specifies the number of test cases required to ensure that all code is executed at least once.
2. Boundary Analysis: Specify tests that will execute code using boundaries at n-1, n, n+1. This includes looping instructions, while, and for operators.
3. Mathematical limit checking: Design tests that use out of range values that could cause the mathematical function to calculate erroneous results.
4. Documentation: The documentation must show that the tests have shown that the topics in items 1 through 3 above have been addressed.

5.2 Integration Testing

Critical module which contain critical control operations should be unit tested as soon as possible and then combined and tested as a group.

5.3 Regression Testing

Each refactoring will use existing junit test suite to ensure all new changes do not cause any new failures in the current test suit. In addition to using the existing test suite, occasionally running of the game in debug mode will be used to ensure modifications to map loading and generation tools have not broken basic functionality.

6. PASS / FAIL CRITERIA

In following the criteria to be used to determine whether each item has passed or failed testing is described:

6.1 Suspension Criteria

Here is the criteria used to suspend all or a portion of the testing activity on test items associated with the plan:

1. In the event that new testing prevents currently established tests from running, the current testing activity will be suspended.
    1.1. Tests that prevent other tests from completing will not be pushed to the repository.

6.2 Resumption Criteria

The conditions needed to be met to resume testing activities after suspension are as follows:

1. Correction of the tests that prevent other tests from completing will be allowed to be pushed to the repository. There are two courses of action for suspended tests:
    1.1. Remove suspended test.
    1.2. Alter suspended test.

6.3 Approval Criteria

Here is the conditions that need to be met to approve test results:

Improve coverage results.
Reduce average cyclomatic complexity.
The following criteria allow for approval of test results:

1. The test does not prevent other tests from completing.
2. The test improves the test coverage of the source code it tests.
3. The test does not contain any suspension criteria.
4. The source code average cyclomatic complexity is reduced.

7. TESTING PROCESS

The methods and criteria used in performing test activities with the methods definitions and procedures for each type of test are listed below:

7.1 Test Deliverables

The deliverable documents from the test process, and the reports created are as following:

1. Google Codepro metrics report
    1.1. Initial Report of unmodified code.
    1.2. Final Report of refactored code.
2. Google Codepro audit report
    2.1. Initial Report of unmodified code.
    2.2. Final Report of refactored code.
3. Eclemma test coverage report
    3.1. Initial Report of unmodified code.
    3.2. Final Report of refactored code.

Feature	Package Name	Class Name	Test Coverage	Average Cyclomatic Complexity
James:				
1	net.sf.freecol.server.generator	SimpleMapGenerator.java	51.3%	4.05
2	net.sf.freecol.server.generator	TerrainGenerator.java	95.1%	4.91
3	net.sf.freecol.server.generator	ColonizationMapLoader.java	39.3%	4.00
4	net.sf.freecol.tools	DesktopEntry.java	91.7%	3.20
5	net.sf.freecol.tools	ForestMaker.java	17.7%	3.14
Newsha:				
6	net.sf.freecol.tools	ColonizationSaveGameReader.java	8.9%	1.53
7	net.sf.freecol.tools	MapConverter.java	15.7%	4.00
8	net.sf.freecol.tools	InstallerTranslations.java	92.5%	4.33
9	net.sf.freecol.tools	RiverMaker.java	100.0%	3.50
10	net.sf.freecol.tools	SaveGameValidator.java	23.8%	2.66
Tao:				
11	net.sf.freecol.server.model	ServerColony.Java	51%	1.25
12	net.sf.freecol.server.model	ServerUnit.Java	34.7%	1.88
13	net.sf.freecol.tools	ColonizationMapReader.Java	68.7%	4.7
14	net.sf.freecol.tools	FSGConvert.Java	48%	2.33

15	net.sf.freecol.tools	MergeTranslation.Java	38%	4.0

Table 3: Test deliverables

7.2 Testing Tasks

1. For all Tests:
   1.1. Ensure Eclipse Luna or Mars is installed and utilizing JDK 8
   1.2. Clone the repository into your local workspace.
   1.3. In the event that the project does **not** register that there are test cases, do the following steps:
      1.3.1. Using the package explore, In the project directory: Navigate to test → src. Right click on src and select "Build path" → "Use as source folder".
      1.3.2. The project should be able to run as a JUnit test now.
   1.4. To run all tests, right click on project and select run as JUnit test.
2. For testing features 1 through 3:
   2.1. Using package explorer navigate to test/src→ net.sf.freecol.server.generator package.
   2.2. Right click on and select run as JUnit test.
   2.3. To test features individually, open package and select desired test class to run as JUnit test.
3. For testing features 4 and 5:
   3.1. Using package explorer navigate to test/src→ net.sf.freecol.tools package.
   3.2. Right click on and select run as JUnit test.
   3.3. To test features individually, open package and select desired test class to run as JUnit test.
4. For testing features 11 and 12:
   4.1. Using package explorer navigate to test/src→ net.sf.freecol.server.model package.
   4.2. Right click on and select run as CodePro->Audit Code or Compute Metrics.
   4.3. To test features individually, open package and select desired test class to run as JUnit test or Eclemma Coverage .
5. For testing features 13 through 15:
   5.1. Using package explorer navigate to test/src→ net.sf.freecol.tools package.
   5.2. Right click on and select run as JUnit or Coverage Test.
   5.3. To test features individually, open package and select desired test

class to run as JUnit test or EClemma Coverage.

7.3 Responsibilities

The members of the group responsible for managing, designing, preparing, executing, checking, and resolving test activities for this project are listed below:

- *Newsha Amirhormozaki* - software tester. Responsible for refactoring poor maintenance and test coverage attributes in the net.sf.freecol.tools package
- *Tao Qiu* - software tester. Responsible for refactoring poor maintenance and test coverage attributres in net.sf.freecol.tools and server.model package.
- *James Rehak* - software tester, scribe (records the issues of the reviews and audits, and note problems and action items.) Responsible for refactoring poor maintenance and test coverage attributres in net.sf.freecol.tools and server.generator package.

7.4 Schedule

Table 4 summarizes are the schedule for testing activities of NTJ FreeCol project:

Time	Task	Output	Resource
Week 1	Initialize Development Environment	N/A	All team members
Week 2	Planning	Draft Plan	All team members
Week 2	Run code analysis tools (Google code pro metrics, audit, and Eclemma Coverage) to identify deficiencies	Metric, audit, and coverage reports	All team members
Week 3	Refactor freecol.tools package & Inspect annotations problems using StyleChecker	Corrected Source Code	Newsha Amirhormozaki
Week 3	Remove high severity errors and bugs from code identified by codepro audit, FindBugs, PMD	Corrected Source Code	James Rehak

Week 3	Refactor freecol.tools & server.generator package to reduce CC and improve test coverage	Corrected Source Code	James Rehak
Week 3	Refactor freecol.server.model package Correct the violations in ServerUnit.Java and ServerColony.Java after Audit Refactor some freecol.server.tools test classes	Corrected Source Code	Tao Qiu

Table 4: Test activities schedule

8. ENVIRONMENTAL REQUIREMENTS

Here is a list of hardware, software, and tools used during the project completion:

8.1 Hardware

Personal Computer with either of the following operating systems: Windows 7 or greater, OS 10.6 or greater, or Linux. The hardware must be capable of running Eclipse luna or greater. Suggested minimum specifications are:

1. RAM : 2 GB or greater
2. Disk space: 1GB or greater
3. 64-Bit Processor
4. Screen Resolution of 1024x768 or greater

8.2 Software

1. Eclipse IDE luna or greater. 64-bit version.
2. Java Runtime Environment and Java Developer Kit 8
3. Google Code pro
4. Eclemma Coverage
5. Junit 4 library

8.3 Tools

The software tools employed in the testing activities are as follows:

1. Google Codepro - used to audit and calculate source code metrics.
    1.1. Metrics
    1.2. Audit
    1.3. Test case generator
2. Eclemma Coverage - used to check unit test coverage of the source code.
3. FindBugs - Identify potential bugs in code base.
4. PMD - Identify potential design smells and Improve maintainability.
5. CheckStyle - Inspect annotation problems.
6. JAutoDoc - Create javadocs for all new methods.
7. JDepend - Identify dependencies between class objects.

8.4 Risks and Assumptions

1. There is the restriction of time constraints for each tester.
2. Refactoring code can sometimes introduce new bugs. Regression tests are mandatory to prevent bad code from entering the repository.

9. CHANGE MANAGEMENT PROCEDURES

Initially we checked through the whole application and planned to work on two other packages. In the practice, we realized that CodePro could not handle the complex of FreeCol. Either our computer ran out of memory when generating test codes or the generated test cases cannot run normally. Finally we changed our plan and mainly focused on one package, net.sf.freecol.tools.