**Sigrid Berkebile & Madhura Tendulkar**



**FreeCol**

**Software Quality Assurance Plan**

**Version: 1.3 Date: 05/16/2016**

**Document History and Distribution**

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision # | Revision Date | **Description of Change** | **Author** |
| 0.0 | 05/04/2016 | Added title page | Sigrid |
| 1.0 | 05/07/2016 | Worked on Sections 1 and 2 | Sigrid, Madhura |
| 1.1 | 05/10/2016 | Added to Sections 1 and 2 | Sigrid |
| 1.2 | 05/15/2016 | Added Sections 7, 8 | Sigrid, Madhura |
| 1.3 | 05/16/2016 | Added Section 5, 6 | Sigrid |
| 1.3 | 05/16/2016 | Added Section 3,4 | Madhura |

**Table of Contents**

1. Introduction............................................................................................................................... 4

2. Test Items................................................................................................................................. 5

3. Features To Be Tested............................................................................................................. 6

4. Features Not To Be Tested...................................................................................................... 7

5. Approach.................................................................................................................................. 7

6. Pass / Fail Criteria.................................................................................................................. 11

7. Testing Process.......................................................................................................................12

8. Environmental Requirements................................................................................................. 15

9. Change Management Procedures...........................................................................................17

# **1. Introduction**

**1.1 Objectives**

The FreeCol Java project is an interactive GUI based strategy game similar to the Civilization game (<http://www.civilization.com>) and written in the Java language. With our plan for conducting quality tests for the game and making efforts at refactoring and enhancing the code, we aim to identify and remedy software errors, and by doing that, improve the resilience of the software to failures. Also, our objective is to make the software more maintainable in the future by finding instances of bad coding style in the project and to improve the code in a way to adhere to a better standard.

As the project is large and many possibilities for quality improvement exist, we will select a set of three areas that will be the focus of our testing activities. As we are unfamiliar with the functionalities of the software, the general project structure, and the software architecture, we will work on issues that can be identified by static analysis of the code base. The main areas for code improvement will be

1) Removal of bugs

2) Improvement of coding style

3) Reduction of cyclomatic complexity

4) Improvement of existing unit tests

We will employ a number of software tools to scan and analyze the code, mainly in the form of plugins for Eclipse. Examples (not exhaustive) are Emma, PWD, and FindBug. After analysis, we will pick a number of about 10 Java classes that will be refactored according to the issues detected by the analytics software. The selection criteria for the classes we work on is the number of issues – we will pick classes in which an especially large number of instances of bad style, bugs and a high cyclomatic complexity as compared to other classes are found.

The deliverables after finishing all testing activities will consist of the improved code as well as reports on the number of resolved issues in the form of descriptions stored in the issue tracker of the project on GitHub [(https://github.com/stoiser5/cosc603-berkebile-tendulkar-finalproject/issues](https://github.com/stoiser5/cosc603-berkebile-tendulkar-finalproject/issues)).

We propose to finish the quality assurance plan and all testing and refactoring activities within one week, with a team consisting of two persons.

**1.2 Testing Strategy**

Our testing strategy is based on static analysis of the FreeCol project for possible bugs (such as incorrect variable comparisons and bad variable comparison) and deviations from good coding practices (i.e., if blocks that are not enclosed in braces). Whereas the whole project can be analyzed, only a small number of issues will be actually addressed and refactored. We envision to test and improve ca. ten Java classes. A detailed plan of the approaches to achieve our quality assurance goals, all test activities, and all deliverables can be found in Section 5 of this document.

**1.3 Scope**

Changes to the test plan will be made after requirements for the game have been changed and documented in the requirements document or online documentation. The responsible personnel for requirements documentation will inform the test team of any updates. Also, if additional code and unit tests become available in the course of the game development, they will be analyzed for adherence to the quality standards outlined in this document.

The quality assurance plan shall be kept together with the FreeCol project in the source controlled repository on GitHub. The folder the plan is stored in will be called ‘reports’. If the need to update the test plan arises, changes will be submitted to the FreeCol repository where the latest version of the test plan will always be available to all interested parties.

**1.5 Definitions and Acronyms**

PWD ………………………………………………….. Don’t shoot the messenger plugin for Eclipse

NIST 500-235 ……………………………………….. NIST US National Institute of Standards and Technology (NIST) Special Publication 500-235: Watson, A. H., Mccabe, T. J., & Wallace, D. R. (1996). Special publication 500-235, structured testing: A software testing methodology using the cyclomatic complexity metric. In *US Department of Commerce/National Institute of Standards and Technology*.

# **2. Test Items**

Our testing will focus on component testing. The components (Java classes) selected for analysis and improvement are listed in Section 5.1.1.

*Documentation:*

* Requirements specification: not existing
* Design specification: not existing
* Users guide: <http://www.freecol.org/documentation/freecol-user-manual.html>
* Operations guide: not existing
* Installation guide: <http://www.freecol.org/documentation/freecol-user-manual.html>
* Features (availability, response time): unknown
* Defect removal procedures: unknown
* Verification and validation plans: not existing
* Issue tracker:<https://sourceforge.net/p/freecol/bugs/>
* Releases:<http://www.freecol.org/news/>
* Download: http://www.freecol.org/download.html

**2.1 Program Modules**

The FreeCol game is not composed of a collection of independent software modules that interface with each other, but is a self-contained Java project. Therefore, the packages/classes contained in the project will be the only modules being tested by static analysis (see Sections 5 and 8).

**2.2 User Procedures**

For time limitations, we will not test the user documentation for correctness, completeness and comprehensiveness.

# **3. Features To Be Tested**

Due to the time restriction, we decided to test only selected features. We will be testing the classes in net.sf.freecol.common.model, net.sf.freecol.client.gui.panel and net.sf.freecol.client.control packages.

1. *Null pointer dereference:* There may exist some branch of statement that, if executed, guarantees a null value will be dereferenced, and would generate a NullPointerException.
2. *Suspicious comparison:* There exist instance of comparison of two reference values using the == or !== operator. The operator == or != actually checks if the objects refer to the exact same memory while equals() method compares the contains of the object.
3. *Dead store variables:* A local variable that is assigned a value but is not used or read in the method is referred to as a dead store. Dead stores waste processor time and memory.
4. *High cyclomatic complexity methods:* Cyclomatic complexity indicates complexity of a program or a method and higher the complexity it becomes difficult to test as method has more pathways through the code. Also it is more difficult to understand.
5. *Design smells:* Design smells are structures in the design or code which violates fundamental design principles and negatively impact design quality. It includes long methods, duplicate code.

# **4. Features Not To Be Tested**

Due to time and resource limitation we were unable to test the GUI. We tried to use tool Google WindowTester Pro to test the GUI. Due to some reason the tool was not able to record the test cases.

**5. Approach**

**5.1 Component Testing**

The major effort of our quality assurance plan will be spent on component testing. We will run several static analysis tools to determine incorrect code, style issues and design smells affecting the maintainability of the software in the future. We will check for issues in the following areas:

* *Bugs* (i.e., null-pointer de-references, comparisons that will fail at run-time)
* *Bad coding style* (i.e., non-descriptive variable names, too long class methods, missing comments, missing final keywords for variables, inline conditionals)
* *Code re-organization* (i.e., positioning variables at the top of the class followed by the constructor, and class methods, etc.)
* Design smells (e.g., long class methods, high cyclomatic complexity)

**5.1.1 Items to be tested:**

We analyzed the FreeCol project and ranked the packages by the number of bugs and issues found by the FindBug and PMD analyzers (see Section 8 for a description of these tools):

*FindBugs:*

Packages with the most bugs:

1. net.sf.freecol.server.generator 11 bugs

2. net.sf.freecol.common.option 4 bugs

3. net.sf.freecol.common.model 3 bugs

4. net.sf.freecol.client.gui.panel 6 bugs

*PMD:*

Packages with the most style problems:

1. net.sf.freecol.common.model 2150 violations

2. net.sf.freecol.client.gui 520 violations

3. net.sf.freecol.server.ai 820 violations

4. net.sf.freecol.client.gui.panel 1280 violations

Guided by these numbers, we will primarily chose Java classes from the packages net.sf.freecol.common.model and net.sf.freecol.client.gui.panel for testing and quality improvement which show score very high on issues and bugs and style issues.

The preliminary selection of classes for reducing issues are:

|  |  |  |
| --- | --- | --- |
| **Package net.sf.freecol.common.model** | **Package net.sf.freecol.client.gui.panel** | **Package**  **net.sf.freecol.client.control** |
| HighScore.java | BuildQueuePanel.java  Flag.java | InGameController.java |
| Limit.java | FreeColDialog.java |  |
| ColonyWas.java | MapEditorTransformPanel.java |  |
| Market.java | ReportCompactColonyPanel.java |  |
| Player.java | Flag.java |  |
|  |  |  |

We will furthermore work on selected unit tests in order to improve their code coverage and thus their value in finding software errors:

|  |
| --- |
| Unit tests |
| PlayerTest.java |
| MarketTest.java |

The issues addressed and worked on will be documented and traceable on the issue tracker of the FreColproject at <https://github.com/stoiser5/cosc603-berkebile-tendulkar-finalproject/issues>.

**5.1.2 Management and Technical Approach**

All personnel involved in testing activities of the FreeCol project will report to Madhura Tendulkar. The test plan and all subsequent changes to the test plan have to be approved by her in writing. She will oversee all activities and will confirm that all efforts adhere to the test plan and are concluded within the proposed time frame. Also, the final report to be submitted online to all contributors of the FreCol project will have to be signed off by Madhura.

All testing activities will be conducted by qualified testing personnel with higher education in Computer Science. The technical resources for the testing activities (computers, software versions, etc.) will adhere to the standards listed in Section 8.

**5.1.3 Pass/Fail criteria**

The criteria for passing our quality tests are defined as follows:

* The test items do not contain any bugs identified by the FindBugs plugin that is of level “Scariest”, “Scary”, “High Confidence”, or “Troubling”
* The test items do not have any of the following style issues as identified by the PMD and Eclipse Checkstyle plugin:
  + uncommented class variables and methods
  + variables that should be final
  + variable declarations that are not positioned at the top of the class
  + variables that do not follow standard coding conventions as pointed out by PMD (variable names that are too short, all caps for static variables, camel case, etc.)
  + if clauses without braces
  + objects that are assigned to null
  + non-transient, non-static class variables that have no mutators
  + unused imports
* The test items do not show any method with a cyclomatic complexity greater than 10 (after recommendations by NIST 500-235 pg. 25).
* Test items in the form of unit tests have a code coverage greater than 90%.
* The FreeCol project has to compile and execute correctly after all defects have been removed.

**5.1.4 Individual roles and responsibilities**

|  |  |
| --- | --- |
| Madhura | * Test oversight * Testing and defect removal for test items BuildQueuePanel.java, ReportCompactColonyPanel.java, MapEditorTransformPanel.java, InGameController.java, SimpleGenerator.java, PlayerTest.java, MarketTest.java * Documentation of defects in the issue tracker on GitHub * Planning and preparation of test plan |
| Sigrid | * Testing and removal of defects for the test items HighScore.java, Limit.java, Flag.java, BuildQueuePanel.java, MapEditorTransformPanel.java, ColonyWas.java * Documentation of defects in the issue tracker on GitHub * Planning and preparation of test plan |

**5.1.5 Milestones**

**Milestone 1:** Test plan finished

*Deliverable:* test plan signed off and submitted to source control

**Milestone 2:** Test items listed in Section 5.1.1 tested

*Deliverable:* test reports generated by Eclipse Emma, PWD, FindBugs,

JDeodorant (committed to FreeCol repository)

**Milestone 3:** Test items listed in Section 5.1.1 corrected and refactored

*Deliverable:* test items submitted to source control, final report submitted to QA

manager

**5.1.6 Schedule**

|  |  |  |
| --- | --- | --- |
| **Week** | **Activities** | **Assigned personnel** |
| 5/2/2016 - 5/6/2016 | Analysis of project for improvement opportunities  Preparation of test plan  Start of testing and refactoring activities | Madhura, Sigrid |
| 5/9/2016 - 5/13/2016 | Updating of test plan  Testing and refactoring of the majority of test items  Finishing of the test plan | Madhura, Sigrid |

**5.1.7 Risk assumptions and constraints**

**Constraints:**

* *Time:* all testing and improvement activities have to be completed within two weeks and before 5/16/2016
* *Resources:* all testing activities have to be completed by only two QA specialists; all analytics software used for testing must be open-source; all analytics software shall be compatible with Eclipse Mars

**Risks:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Impact** | **Probability** | **Countermeasure** |
| De-scoping of testing activities because refactoring affects other classes in the project significantly | high | medium | The remaining test items with the least number of issues will not be tested to stay on time |
| Code not functional after testing activities completed | high | low | Code changes have to be rolled back to last working code base |
| Static analyzers which we plan on using have bugs and do not work, or are incompatible with our copy of Eclipse | high | low | We try to install a different version of Eclipse and use our backup analyzers -> pass/fail criteria have to be changed in the test plan |

**5.2 Integration Testing**

Due to time limitations, no integration testing will be performed.

**5.3 Interface Testing**

Due to time limitations, no interface testing will be performed.

**5.4 Security Testing**

Due to time limitations, no security testing will be performed.

**5.5 Performance Testing**

Due to time limitations, no performance testing will be performed.

**5.6 Regression Testing**

Whenever the code is modified or bug is resolved all the test cases are re-runto ensure that changes made have no adversary effect on the project.

**5.7 Acceptance Testing**

Acceptance testing is not in the scope of this quality assurance plan.

**5.8 Beta Testing**

Beta testing can be started after all test items have been worked on and adhere to the quality criteria outlined in Section 5.1.3. The FreeCol executable built with the improved code will be made available (marked as beta) online <http://www.freecol.org/download.html>. The beta version will be released in the form of a Windows installer and a platform independent jar file. After acceptance testing, a regular release will be scheduled.

# **6. Pass / Fail Criteria**

The pass/fail criteria for our software test items are listed in Section 5.1.3.

**6.1 Suspension Criteria**

Every two days, the FreeCol project with all new code will be built and executed. Continuation of test activities depends on the successful execution of these builds. If the project is not in working condition, all testing activities will be stopped and the problems tracked down and eliminated.

**6.2 Resumption Criteria**

If testing has been suspended, it can only continue if the FreeCol project has been shown to execute again and has been approved by the manager overseeing the testing activities.

**6.3 Approval Criteria**

*Testing approval process:*

1. Test activities are finished when all issues marked as “Milestone 3” closed on the FreeCol issue tracker on GitHub
2. Manager executes improved FreeCol project
3. After successful execution, manager marks issues as “Tested successfully” by commenting on all issue.
4. Manager approves release of the beta version of FreeCol.

# **7**. **Testing Process**

This section explains the testing process conducted to improve the quality of code and includes the deliverables and responsibilities among the team members.

**7.1 Test Deliverables**

1. Completed quality assurance plan
2. Code coverage report generated using EclEmma Plugin.
3. Metrics’ report generated using Google Code Pro.
4. Refactored code without bugs submitted to FreeCol repository
5. Final report

**7.2 Testing Tasks**

1. To begin with testing or refactoring, first we need to study the available FreeCol documentation and Javadoc available at <http://www.freecol.org/documentation/>.
2. Once we have identified classes and features to be tested we should analyze and select the testing tool to be used for improving code quality.
3. Learn the working of the testing tools either by reading documentation or watching tutorials in Youtube.
4. Understand the interdependencies of the classes to be tested, i.e., the class may use some methods defined in other classes.
5. Perform the static analysis of the selected class to find bugs and bad practices.
6. Fix the issues and bugs identified in step 5. Improve the code quality by adding comments and run the test cases, i.e., perform regression testing to ensure that code is working fine and changes do not break the code.
7. Calculate metrics and perform refactoring to reduce complexity of the methods by extracting some methods, removing the unused variables. Re-run the test cases and check that code can be built successfully.
8. Add test cases to increase the code coverage. Run the test cases and resolve if any issue arises.

**7.3 Responsibilities**

|  |  |
| --- | --- |
| Task | Responsible |
| Identify the classes/ packages for testing | Sigrid, Madhura |
| Preparing test plan | Sigrid, Madhura |
| Improve code quality by static analysis | Sigrid, Madhura |
| Improve the code coverage by adding/modifying test cases | Sigrid,Madhura |

Specific responsibilities pertaining to component testing are outlined in Section 5.1.4.

**7.4 Resources**

* **QA Team:** Sigrid and Madhura

*Responsibilities:*

1.Selecting and prioritizing the classes/ packages to improve and test.

2. Preparing test plan

3. Identify the tools to be used for testing.

4. Resolving bugs in the selected classes.

5. Refactoring the selected classes by eliminating wrong or bad coding style, renaming variable names if they are not according to Java coding standard and adding comments and JavaDocs.

6. Add and modify the test cases.

* **Tools:** 
  + *FindBugs:* FindBugs is free software distributed under the terms of the Lesser GNU Public License. FindBugs perform static analysis to identify possible errors or bugs in the Java code. FindBugs Plugin is available in Eclipse IDE from Eclipse Marketplace under the category of source code analyzer.To use the FindBugs Plugin for Eclipse IDE, Eclipse version 3.3 or higher and JRE/JDK 1.5 or higher is required.
  + *PMD:* PMD is free plugin for Eclipse IDE which scans the Java code and looks for duplicate code, dead code and bad coding practice. PMD is available in Eclipse Marketplace under the category of source code analyzer.
  + *Google Code Pro:* Google Code Pro is free software testing tool for Eclipse IDE developed by Google. It performs code analysis, generates test cases and compute metrics. It is available in Eclipse Marketplace under source code analyzer and testing category.
  + *EclEmma:* EclEmma is a free Java code coverage tool for Eclipse IDE which shows how efficient test cases are. EclEmma can be installed in Eclipse IDE from Eclipse Marketplace under testing category.
  + *Eclipse Checkstyle Plugin:* Eclipse Checkstyle Plugin ensures that Java code adheres to set of coding standards. It can be installed in Eclipse IDE from Eclipse Marketplace under source code analyzer category.
  + *JDeodrant:* JDeodorant is an Eclipse plugin that identifies design problems in software, known as bad smells, and resolves them by applying appropriate refactorings. It can be installed in Eclipse IDE from Eclipse Marketplace under source code analyzer category.

**7.5 Schedule**

|  |  |
| --- | --- |
| Task | Hours |
| Create GitHub repository and commit the code | < 1 hour |
| Setup the environment in local machine | < 1 hour |
| Read the FreeCol documentation and Javadocs | 2 |
| Select and prioritize the classes/packages to improve and test | 3 hours |
| Perform static analysis of selected class | 4 to 5 hours |
| Identify the dependencies for the methods to be tested | 2 hours |
| Create unit test cases | 4 to 5 hours |
| Create coverage and metrics reports | < 1 hour |

# **8. Environmental Requirements**

**8.1 Hardware**

To compile and run FreeCol the computer hardware requirements are:

1. Recent version of Windows, Linux or Mac OS X.
2. FreeCol requires at least 256 MB memory. As FreeCol is large software, system may slow down and hence 512 MB memory is recommended.
3. Required screen resolution is at least 1024x 768 pixels.

**8.2 Software**

The software requirements are:

1. As FreeCol is written in Java programming language, it requires a Java Virtual Machine.
2. FreeCol is compatible with Sun Java 8.
3. Ant Build System is needed along with Java to compile and run the tests.
4. Eclipse IDE for Java Developer
5. Jar files : junit- 4.12.jar and hamcrest-core-1.3.jar must be added in project to test the software.

**8.3 Security**

FreeCol does not require any testing of environment security.

**8.4 Tools**

Utilized tools are:

1. *Google Code Pro:* Google Code Pro is free software testing tool for Eclipse IDE developed by Google.

Features: 1. Code Analysis (audit, dependency,similarity)

2. JUnit test generation

3. Compute Metrics (Calculate cyclomatic complexity of methods)

We utilized Code Pro to compute metrics to identify classes which needs to be refactored. The packages and classes with high complexity as compared to others classes were selected for testing and improvement.

2. *EclEmma:* EclEmma is a free Java code coverage tool for Eclipse.

Feature: Generates code coverage reports based on:

Method coverage: number of methods called by test cases.

Statement coverage:number of statements tested by test cases.

Branch coverage: decision points (if statements) evaluated both true and false

Condition coverage: boolean expressions evaluated both true and false

To examine the coverage of existing test cases, we employed the EclEmma tool. It also assisted us in selecting classes for testing and refactoring.

3. *FindBugs:* FindBugs is free software distributed under the terms of the Lesser GNU Public License. Eclipse 3.3 or later and JRE/JDK 1.5 or later is required to use FindBugs

Plugin.However it can analyze the programs compiled for any version of Java from 1.0 to 8.0.

Feature: Static analysis to look for bugs in Java code. FindBugs categorizes bugs into bad practice, correctness, performance, security and malicious code vulnerability.

To perform static analysis of the selected classes, we employed FindBugs plugin .

4. *PMD:* PMD is acronym for Programming Mistake Detector. It is a free source code

analysis tool to find the problems in Java code and improve the code quality.

Features: 1. PMD scans Java source code and reports problems like:

Dead code: unused local variables, parameters and private methods.

Possible bugs: empty try, catch, finally and switch statements.

Duplicate code: Redundant code, which may be copied and pasted twice.

2. Additionally PMD also provides explanation about the issues and how to resolve errors by showing the related example.

We resolved the coding violations in the classes using PMD.

5. *Eclipse Checkstyle Plugin:* It integrates the well-known source code analyzer Checkstyle into Eclipse IDE.

Features: Ensures that Java code adheres to set of coding standards. It points outs the items that deviate from a defined set of coding rules.

We resolved the coding violations in the classes using Eclipse Checkstyle plugin. It also   
 provides ways to improve readability of code by

6. *JDeodrant:* JDeodorant is an Eclipse plugin that identifies design problems in software,

known as bad smells, and resolves them by suggesting appropriate refactorings.

Feature: JDeodrant tool identifies bad smells including

Long methods problems can be resolved by appropriate extract method   
 refactoring.

God class problems are solved by appropriate extract class refactoring.

Duplicate code problems are resolved by appropriate extract clone refactoring.

We used JDeodrant tool to identify long methods and duplicate code problems and   
 resolved by extracting methods suggested by the tool.

7. *Github:* Github is online repository to save the projects. It is version control system, which manages and saves the versions of the projects. It also helps to keep the track of issues that arise in the project.

**8.5 Risks and Assumptions**

The most significant constraint on testing was time constraint. FreeCol is large software consisting of thousands of methods and classes. It is literally impossible to test all aspects of code. Hence we decided to select some classes to improve and test them. To select the classes, we considered various factors such as high complexity, number and type of bugs in classes and bugs and tried to resolve the most of the bugs in the selected classes.

Specific risks for component testing can be found in Section 5.1.7.

# **9. Change Management Procedures**

*Change management process:*

1. Change in software requirements and implementation reported to QA manager
2. QA manager appoints QA specialist to conduct impact analysis for the test plan
3. QA manager approves appropriate changes for the test plan
4. QA manager assigns changes to QA specialist for implementation
5. Changes must be documented with date, reason for the change and responsible personnel that implemented the changes

# 