**Software Test Plan (STP) Template**

Items that are intended to stay in as part of your document are in **bold**; explanatory comments are in *italic* text. Plain text is used where you might insert wording about your project.

This document is an annotated outline for a Software Test Plan, adapted from the IEEE Standard for Software Test Documentation (Std 829-1998).

Tailor as appropriate. Where you decide to omit a section, you might keep the header, but insert a comment saying why you omit the element.

**Testing Testing 1,2,3**

**<Insert Game Image/Icon>**

**Free Col Testing**

**Software Quality Assurance Plan**

**Version: (1) Date: (04/16/2018)**

**Document History and Distribution**

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision #** | **Revision Date** | **Description of Change** | **Author** |
| 1 | 04/16/2018 | Unit Test Implemented |  |
| 2 | 04/24/2018 | Refactoring to all of the methods with high CC |  |
| 3 | 5/13/2018 | Implementation of new feature |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**TABLE OF CONTENTS**

**1.** **Introduction 1**

**2.** **Test Items 2**

**3. Features To Be Tested 3**

**4. Features Not To Be Tested 3**

**5. Approach 3**

**6. Pass / Fail Criteria 5**

7. **Testing Process 5**

**8. Environmental Requirements 6**

**9. Change Management Procedures 7**

# INTRODUCTION

According to Foundations of Software Testing, “Software testing deals with a variety of concepts, some mathematical and others not so mathematical. In this project we will be evaluating a game for quality and functionality. We will be using various tools to reach our ideal goal. Foundations of Software Testing states, “ the several measures of software quality are static and dynamic quality attributes. The two attributes include the behavior, structure, maintainability, testable of the code.” The attributes are very important when testing because we can identify faults, bug and failures that happen when the program is running. Imagine everything that runs a program in order to operate. Train, cars, airplanes, phones, satellites and pacemakers all have some form of software they run. However, before they can be implemented, they have to be tested. Otherwise, we would have a lot of danger amongst us.

**1.1 Objectives**

The aim of testing the FreeCol video game is to ensure the quality of the game is fits the criteria of the requirements provided by the user’s. Testing is to ensure bugs, faults, and failures, are identified and corrected by using a number of various software testing tools.

**1.2 Testing Strategy**

We have decided to complete unit, regression, performance, and acceptance testing. The source code contains 27 classes. Although time consuming, each class will be tested by running the junit and attempting to reduce the cyclomatic complexity below 7. If the cyclomatic complexity cannot be reduced below 7, the objects of that class will be considered a failure. This part of the testing is expected to take between 2 to 3 weeks. Afterwards, each member will choose and learn how to use two of the following tools: Codepro, JDepend, Metrics, CodeCity, and XRay. Once this is complete, the tool should develop a report of the developers findings and will be added to the project file. Milestone should be indicated by repository commits after each class is modified.

**1.3 Scope**

Scheduled updates will be implemented via push to the repository. Unscheduled updates will be implemented within integration testing. We will communicate amongst ourselves when unexpected results occur and need to be addressed. Unscheduled updates may come in the form of the team pulling the last successful commit to the repository.

**1.5 Definitions and Acronyms**

* XML (Extensible markup language)
* CodePro - code analyzer which includes code coverage.
* Metrics - contains the stats of the class, objects, and modules associated in the program
* Freecol - a game where the player plays as a explorer and colonizer.
* Requirements (software requirements) - the ideal functionality and behavior the user expects.

# TEST ITEMS

*(Specify the test items included in the plan. Supply references to the following item documentation:*

The items that will be tested are individual packages, classes, objects and methods. The items will be tested using JUNIT testing and code coverage, and metric cyclomatic complexity.

* *Requirements specification, http://www.freecol.org/download.html*
* *Design specification, http://www.freecol.org/screenshots.html*
* *Users guide, http://www.freecol.org/documentation/freecol-user-manual.html*
* *Operations guide, http://www.freecol.org/about.html*
* *Installation guide, http://www.freecol.org/download.html*
* *Features (availability, response time),*
* *Defect removal procedures, and* *Verification and validation plans.) http://www.freecol.org/download.html*

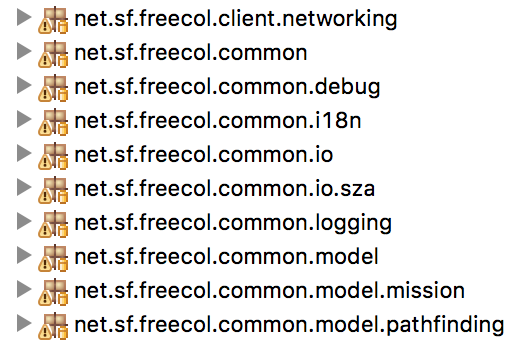
**2.1 Program Modules**

*(Outline testing to be performed by the developer for each module being built.)*

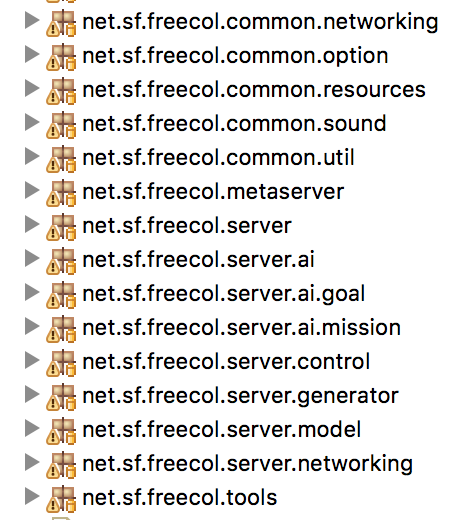
Joshua ran testing on the following packages:



Marcus tested the following packages:



Richie and Tyler tested the following packages:



**2.2 User Procedures**

*(Describe the testing to be performed on all user documentation to ensure that it is correct, complete, and comprehensive.)*

* Online documentation on the Free Col’s team website
* Installation instructions
* Requirements specifications
* User’s guide
* Operations guide

# 3. FEATURES TO BE TESTED

*(Identify all software features and combinations of software features to be tested. Identify the test design specifications associated with each feature and each combination of features.)*

# 4. FEATURES NOT TO BE TESTED

# 5. APPROACH

*(Describe the overall approaches to testing. The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to do each task. Identify the types of testing to be performed along with the methods and criteria to be used in performing test activities. Describe the specific methods and procedures for each type of testing. Define the detailed criteria for evaluating the test results.)*

*(For each level of testing there should be a test plan and the appropriate set of deliverables. Identify the inputs required for each type of test. Specify the source of the input. Also, identify the outputs from each type of testing and specify the purpose and format for each test output. Specify the minimum degree of comprehensiveness desired. Identify the techniques that will be used to judge the comprehensiveness of the testing effort. Specify any additional completion criteria (e.g., error frequency). The techniques to be used to trace requirements should also be specified.)*

In general, our testing method consists of generating tests cases with the Google CodePro tool for classes with a cyclomatic complexity over two depending on how large the package they’re within is. If these classes are within a package with less than 10 classes that have a cyclomatic complexity of 10, all the classes that have a cyclomatic complexity will be tested. In other classes with more than 10 classes that have a cyclomatic complexity over 2, the classes with the top cyclomatic complexities will get JUnit tests. Our JUnit testing isn’t successful unless all of our tests pass.

We will also test the game by playing the game to see if users can access the game on multiple platforms and OS’s. We will test if the game adapts to different platforms well and that the user can interact with the game as they were intended to and not in ways they can compromise the security of the game.

We will maintain group cohesive by communicating the tests and changes we’ve made in our commits and through slack. After every commit, we will download the newest version of the program to see if the program functions properly and that our additions to the program to the program weren’t erased if they weren’t supposed to.

**5.1 Component Testing**

*(Testing conducted to verify the implementation of the design for one software element (e.g., unit, module) or a collection of software elements. Sometimes called unit testing. The purpose of component testing is to ensure that the program logic is complete and correct and ensuring that the component works as designed.)*

We plan to construct JUnit tests for classes with a cyclomatic complexity above 2 in packages with a low number of classes, and in large packages with more than 10 classes, we plan on only making JUnit tests for the classes with the top 10 highest cyclomatic complexities.

We will construct our JUnit tests with the Google CodePro tool. Since much We have so many classes, we will rely on the tool to ensure the quality of our tests. Component testing could take as much as two or three hours long in total because it takes a long time to generate JUnit tests.

**5.2 Integration Testing**

*(Testing conducted in which software elements, hardware elements, or both are combined and tested until the entire system has been integrated. The purpose of integration testing is to ensure that design objectives are met and ensures that the software, as a complete entity, complies with operational requirements. Integration testing is also called System Testing.)*

After every commit, we will notify one another and we will test the game to ensure that it operates correctly. This testing would take maybe an hour: we’d have to play the game to see if it’s operating as intended on our laptops.

**5.3 Interface Testing**

*(Testing done to ensure that the application operates efficiently and effectively outside the application boundary with all interface systems.)*

We will create specific XML files to test whether they are being interpreted correctly. We will also test if user input is being correctly interpreted by the game. For example, we’ll try to see when we name a something we’ll see if that thing is properly named.

**5.4 Security Testing**

*(Testing done to ensure that the application systems control and auditability features of the application are functional.)*

We will test to see if the game can be compromised by certain user input. We make tests around the specification.xml file in the rules folder to see if rules are being accounted for by the program. For instance, we’ll ensure that the value for grain is not 3 instead of 2. We’ll ensure that a user can’t get an insurmountable amount of resources because of a “cheat” or user action.

We will accomplish this testing by actually playing the game and implementing JUnit tests.

**5.5 Performance Testing**

*(Testing done to ensure that that the application performs to customer expectations (response time, availability, portability, and scalability)).*

We will implement performance testing through a number of methods: playing the game, downloading it on our devices (we have access to multiple Windows and Mac laptops with different specifications). Our goal with this testing is to see if we can all play the game on our devices and the gameplay is smooth without any slow downs. This testing will probably only take an hour.

**5.6 Regression Testing**

*(Testing done to ensure that that applied changes to the application have not adversely affected previously tested functionality.)*

We will test our program with new JUnit tests that test our new features in combination with old JUnit tests we believe should operate the same in order to test if the program functions as intended after changes have been applied.

**5.7 Acceptance Testing**

*(Testing conducted to determine whether or not a system satisfies the acceptance criteria and to enable the customer to determine whether or not to accept the system. Acceptance testing ensures that customer requirements' objectives are met and that all components are correctly included in a customer package.)*

We will use a combination of JUnit tests and playing the game to assess whether a potential customer can play the game smoothly and all features are available to them in terms of player options, in game options, and that that. This testing will probably take around 6 hours to verify that all features are available to the customer.

**5.8 Beta Testing**

*(Testing, done by the customer, using a pre-release version of the product to verify and validate that the system meets business functional requirements. The purpose of beta testing is to detect application faults, failures, and defects.)*

To do this we will do a total playthrough of the game. This will probably take 10 hours.

# 6. PASS / FAIL CRITERIA

*(Specify the criteria to be used to determine whether each item has passed or failed testing.)*

* User can interact with program as intended
* If xml files can be read correctly
* If the user’s input is interpreted correctly
* If the program still runs
* If the program is adapting to the user’s preferred platform as intended
* Game elements are visible
* The GUI elements of the game are functioning
* Variables are in the state and have the values they are supposed to have

**6.1 Suspension Criteria**

(*Specify the criteria used to suspend all or a portion of the testing activity on test items associated with the plan.)*

* If a user can’t interact with the program as intended
* If xml files can’t be read
* If a user’s input isn’t or cannot be interpreted correctly
* The program can’t run
* If the program is not adapting to the user’s preferred platform as intended
* Game elements are not visible
* If the GUI elements of the program aren’t functioning properly
* If variable is null or full of data when it’s not supposed to be

**6.2 Resumption Criteria**

*(Specify the conditions that need to be met to resume testing activities after suspension. Specify the test items that must be repeated when testing is resumed.)*

* The specific bug must be fixed and tested to totally ensure that it is fixed

**6.3 Approval Criteria**

*(Specify the conditions that need to be met to approve test results. Define the formal testing approval process.)*

* All tests must pass
* 30% of line coverage must be achieved

# 7. TESTING PROCESS

*(Identify the methods and criteria used in performing test activities. Define the specific methods and procedures for each type of test. Define the detailed criteria for evaluating test results.)*

**7.1 Test Deliverables**

*(Identify the deliverable documents from the test process. Test input and output data should be identified as deliverables. Testing report logs, test incident reports, test summary reports, and metrics' reports must be considered testing deliverables.)*

**7.2 Testing Tasks**

*(Identify the set of tasks necessary to prepare for and perform testing activities. Identify all intertask dependencies and any specific skills required.)*

**7.3 Responsibilities**

*(Identify the groups responsible for managing, designing, preparing, executing, witnessing, checking, and resolving test activities. These groups may include the developers, testers, operations staff, technical support staff, data administration staff, and the user staff.)*

**7.4 Resources**

*(Identify the resources allocated for the performance of testing tasks. Identify the organizational elements or individuals responsible for performing testing activities. Assign specific responsibilities. Specify resources by category. If automated tools are to be used in testing, specify the source of the tools, availability, and the usage requirements.)*

**7.5 Schedule**

*(Identify the high level schedule for each testing task. Establish specific milestones for initiating and completing each type of test activity, for the development of a comprehensive plan, for the receipt of each test input, and for the delivery of test output. Estimate the time required to do each test activity.)*

*(When planning and scheduling testing activities, it must be recognized that the testing process is iterative based on the testing task dependencies.)*

# 8. ENVIRONMENTAL REQUIREMENTS

(Specify both the necessary and desired properties of the test environment including the physical characteristics, communications, mode of usage, and testing supplies. Also provide the levels of security required to perform test activities. Identify special test tools needed and other testing needs (space, machine time, and stationary supplies. Identify the source of all needs that is not currently available to the test group.)

**8.1 Hardware**

*(Identify the computer hardware and network requirements needed to complete test activities.)*

**8.2 Software**

*(Identify the software requirements needed to complete testing activities.)*

**8.3 Security**

*(Identify the testing environment security and asset protection requirements.)*

**8.4 Tools**

*(Identify the special software tools, techniques, and methodologies employed in the testing efforts. The purpose and use of each tool shall be described. Plans for the acquisition, training, support, and qualification for each tool or technique.)*

**8.5 Risks and Assumptions**

*(Identify significant constraints on testing such as test item availability, test resource availability, and time constraints. Identify the risks and assumptions associated with testing tasks including schedule, resources, approach and documentation. Specify a contingency plan for each risk factor.)*

# 9. CHANGE MANAGEMENT PROCEDURES

*(Identify the software test plan change management process. Define the change initiation, change review, and change authorization process.)*