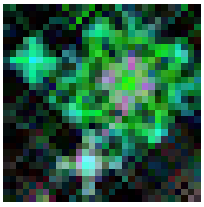

Biogenesis

Software Quality Assurance Plan

Kenway Chung, Sabin Tiwari

TABLE OF CONTENTS

1. INTRODUCTION

(NOTE 1: THE SOFTWARE TEST PLAN GUIDELINES WERE DERIVED AND DEVELOPED FROM IEEE STANDARD FOR SOFTWARE TEST DOCUMENTATION (829-1998)).

(Note 2: The ordering of Software Test Plan (STP) elements is not meant to imply that the sections or subsections must be developed or presented in that order. The order of presentation is intended for ease of use, not as a guide to preparing the various elements of the Software Test Plan. If some or all of the content of a section is in another document, then a reference to that material may be listed in place of the corresponding content.)

The Introduction section of the Software Test Plan (STP) provides an overview of the project and the product test strategy, a list of testing deliverables, the plan for development and evolution of the STP, reference material, and agency definitions and acronyms used in the STP.

The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The plan must identify 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 scope of the testing activities includes the unit testing of the aspects of the code. We will also be testing the GUI of the application to ensure that the GUI acts as it is supposed to. We will be using different tools to run tests on the program. The main goal of the testing process will be to find bugs and issues that may exist in the software.

1.2 Testing Strategy

Testing is the process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item. (*This may appear as a specific document (such as a Test Specification), or it may be part of the organization's standard test approach. For each level of testing, there should be a test plan and an appropriate set of deliverables. The test strategy should be clearly defined and the Software Test Plan acts as the high-level test plan. Specific testing activities will have their own test plan. Refer to section 5 of this document for a detailed list of specific test*

plans.)

Specific test plan components include:

- *Purpose for this level of test,*
- *Items to be tested,*
- *Features to be tested,*
- *Features not to be tested,*
- *Management and technical approach,*
- *Pass / Fail criteria,*
- *Individual roles and responsibilities,*
- *Milestones,*
- *Schedules, and*
- *Risk assumptions and constraints.*

1.3 Scope

We will be updating the Software Test Plan as the testing procedures progress. After the basic test plan is added to the document, changes will be made as needed.

Testing will be performed at several points in the life cycle as the product is constructed. Testing is a very 'dependent' activity. As a result, test planning is a continuing activity performed throughout the system development life cycle. Test plans must be developed for each level of product testing.

2. TEST ITEMS

2.1 Program Modules

- main.java.biogenesis

3. FEATURES TO BE TESTED

- GUI
- World generator
- Organism generator
- Messages

4. FEATURES NOT TO BE TESTED

- Events
- Music
- Scripting

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.)

It was determined that we would test the Organism and World classes because they had the most amount of methods. We used EclEmma Code Coverage to verify this. We're also testing the GUI components of the application since a lot of the classes rely on the GUI to display the right information.

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.)

5.2 Integration Testing

The integration testing will be done as a part of getting the program to run on the host machines to perform the testing.

5.3 Interface Testing

No interface testing planned.

5.4 Security Testing

No plans for security testing.

5.5 Performance Testing

Use the Eclipse Memory analyzer to check memory usage and performance of the program. Make changes to the program to handle memory usage more efficiently.

5.6 Regression Testing

No new functionality is planned so regression testing is not planned either.

5.7 Acceptance Testing

Acceptance testing will be done by running all the JUnit tests including the GUI tests.

5.8 Beta Testing

No beta testing planned.

6. PASS / FAIL CRITERIA

6.1 Suspension Criteria

Any test cases that will cause an OutOfMemory exception will be discarded. Any item that is being tested that cannot be tested with the current tools will be suspended for later.

6.2 Resumption Criteria

The tests that match the suspension criteria will be modified when possible to make sure that the tests will run properly. Any item that was suspended for testing will be resumed for testing whenever the proper tool are obtained.

6.3 Approval Criteria

If the JUnit tests and the GUI tests pass for the elements that are being tested, then the tests will be approved and the item will be deemed approved as well.

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

- Issues will be tracked on the Bug Tracker on GitHub
- Testing reports will be included in a folder on GitHub named Reports

7.2 Testing Tasks

Install the tools being used for Eclipse
Pull the latest code from GitHub
Analyze Code Coverage on Project
Run the JUnit tests on the project (that includes the GUI tests)
Run the Check Style for the code to check any styling issues

7.3 Responsibilities

Kenway: JUnit code, CheckStyle, CodePro, FindBugs
Sabin: JUnit code, Window Tester, CodePro, Reports. Memory Analyzer

7.4 Resources

Testers: Kenway, Sabin
Devs: Kenway, Sabin

7.5 Schedule

JUnit Tests: Finish by May 12, 2015

Reports: Finish by May 17, 2015
Run Tests: Finish by May 12, 2015
Run FindBugs: Finish by May 12, 2015
Window Tester automation: Finish by May 12, 2015
Software Test Plan: Finish by May 17, 2015

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

- RAM: At least 4 GB
- Integrated Graphics Processor
- CPU: At least 2 cores with at least 2.0 GHz clock speed

8.2 Software

- Eclipse IDE
- Windows OS
- Java
- SVN

8.3 Security

- None

8.4 Tools

- JUnit
- Google CodePro
- EclEmma Java Code Coverage
- FindBugs
- CheckStyle
- Eclipse Memory Analyzer
- Google WindowTester Pro

8.5 Risks and Assumptions

- Need to be able to run the Eclipse Memory Analyzer
- Tests may cause out of memory errors because of some tests running multiple instances
- Tests may not be completely accurate of software quality