Test Plan
Freedom in the Galaxy, Python Team

Version 1.0

Prepared for:
CS 383 Course Project

Prepared by:
CS 383 Python Team
University of Idaho
Moscow, ID 83844-1010

November 18, 2013

Record of Changes

Change Number	**Date Completed**	**Location of Change (e.g. page # or figure #)**	**Brief Description of Change**	**Approved by (initials)**	**Date Approved**
1	Nov 9	ALL	INITIAL PREP	-	-
2	Nov 11	Sec5.1 pg6 ; Sec6 pg6 ; Sec7 pg6 ; Sec8.5 pg6 ; Sec9 pg8-9 ; Sec9.2.1 pg9-13	1st Draft Review and Approval	EThom JHall	Nov 11

Contents

1 TEST PLAN IDENTIFIER

FREEDOM IN THE GALAXY MASTER TEST PLAN VERSION 1.0, PYTHON GROUP

2 REFERENCES

Use Cases and State diagrams are available at `https://github.com/Freedom-Galaxy`.

3 INTRODUCTION

The purpose of this master test plan is to state the processes used by the Python Team in testing the Freedom in the Galaxy software project. This master plan covers the entire testing plan for the project. The Python Team is divided into three sub teams. Each sub team each has different testing requirements and should prepare documents accordingly.

4 TEST ITEMS

- Class Interfaces
- Class Interactions
- User Interface Functions
- Network Layer Functions
- API Level Functions
- Requirements stated in System Software Requirement Specification
- Requirements stated in System Software Design Documentation

5 SOFTWARE RISK ISSUES

Each piece of code has different levels of risk to the party. Standard library modules are usually designed to be backwards compatible, do not change for each version of Python, are heavily tested before incorporation into the standard library, and have a high level of use. Therefore, the standard library modules have a very low level of risk to the project. The tests will therefore be focused on the correct use of those modules.

Third party software has a higher level of risk than standard library and a much larger variance of quality. Some of the software has been heavily tested and some is only in a prerelease state. The level of quality of each third party module will have to be determined and testing scaled according. The level of risk can be reduced if sufficient tests are included with the library.

Testing will mainly focus on code written by Python Group members. This plan largely focuses on providing a testing plan that adequately covers both rules and implementation.

As the code base is not yet complete, the complexity of the code cannot be yet determined. Also, since the documentation is often incomplete, risk also arises because of poor documentation. Complete and improved documentation of all areas of code with reduce this risk.

Finally, because the length of this project is short and is not dependent on any third party schedules, there is no to little risk from new versions of software or failure of any third party.

5.1 Critical Risk Areas

1. RPyc
2. PyGame
3. SQLAcademy

6 FEATURES TO BE TESTED

Feature Description	**Risk Level**
Start a game	High
Smart AI	Low
Network communication	High
Play game with graphical interface	Med
Play game with text interface?	Low
User Experience	-

7 FEATURES NOT TO BE TESTED

Feature Description	**Details**
Python Standard Library.	
Probably we can put third party modules here.	
Speed of software.	

8 APPROACH

8.1 Testing Tools

The testing tools for use in this project include the standard library modules doctests, unittest, and the third party tool Nose. Each tool has different levels of complexity and strengths, so each test will be written in one of these tools depending on the tester and condition.

Training will be available from other members of the team for any questions. Tutorials are also available on the Python website for doctest and unittest. Training for Nose is unknown.

8.2 Metric

The number of game rules implemented will be the metric.

8.3 Configurations

The AI and UI will be tested using both the Rebel and Imperial player, and all User type tests for both player types will be executed.

8.4 Software

The software will be developed with Python 2.7.5 with doctest and unittest of the same version. The current version of Nose is 1.3.0, the current version of Pygame is 1.2.1.

8.5 Hardware

The harware specifications used for testing will be at or higher than the minimum hardware specifications for Python 2.7.5.

8.6 Automated Testing

The automated test process will be divided into three phases, Unit testing, Integration testing and System testing.

8.6.1 Unit Testing

Unit testing will be used to validate individual classes objects. Some objects will not be required to be validated by unit testing, such as the user interface and the supporting classes and any classes or code that is exempted according to section 5.

Each team should develop TCSs as needed for their code. Each TCS should include tests for all classes as required and for each test, validate as many inputs as needed to test that the class operates as required by the rules covered in the SSRS, and also according to an requirements and design documents applicable to that class.

8.6.2 Required Class Unit Tests

- What classes do we have? Each class should have tests?

Unit testing for each of these classes will consist of validating the class public interface. Public variables and methods are required to pass the following tests:

- validation using expected data input
- validation using erroneous data input, including None or empty values
- class instantiation

8.6.3 Integration Testing

Integration testing will validate the defined associations between classes. This will be an incremental process performed as new classes are created and after classes have successfully passed their unit testing phase.

Integration Tests to be Performed

- Class sets under test exhibit the relationships as defined in the class diagram
- Methods and functions are called using the correct parameters

- Methods and functions return the expected data types or structures.

Integration should be done at the group level. This should include mock objects or mock data to simulate any objects outside of the team as required. Integration testing should cover all reasonable combinations of objects in the game. These tests should be documented in TCSs.

8.6.4 System Testing

System testing will exercise the overall software product to verify conformance to defined game rules. System testing will use predefined scenarios of initial states representing possible game states that a user will encounter. Each scenario will be tested for acceptable results for each possible user action that may occur. Acceptability will be defined by the game rules.

8.7 Manual Testing

Manual testing will be required for the portions of the program that can not undergo automated testing. This section applies to the testing of the user interface to simulate user orientated testing to verify conformance to the documented use cases.

9 ITEM PASS/FAIL CRITERIA

9.1 Reporting a failure

If a failure happens during the execution of any test, a failure report should be submitted to provide information of the failure. Failure should be reported in the issue page of the repository.

The name of the issue should be the name of the name of the code file and the name of the test if applicable.

9.1.1 What Constitutes a "failure"?

Any deviation from a specification, e.g. SRS, UML diagrams.

9.1.2 What Does Not Constitute a "failure"?

Any unit or action that does not have any requirements documentation, cannot cause a 'failure'.

9.1.3 What to do when a "failure" is discovered"?

Produce a SCR to document each "failure" that needs to be corrected.

9.1.4 What if a specification document is incorrect (e.g. outdated, misstated)?

This also constitutes a "failure" and an SCR should be created.

9.1.5 What sections to include in an SCR

Failure Identified
Expected Outcome
Actual Behavior
Steps to reproduce

Each section should be brief and to the point, but yet convey enough information for the coder.

9.2 Unit Testing Pass/Fail Criteria

Each of these tables might need to be expanded to cover priority and type of test (i.e. unittest, integration, system). Also, additions of conditionals might also be good.

9.3 Integration Testing Pass/Fail Criteria

9.4 System Testing Pass/Fail Criteria

TBD

9.5 User Interface Manual Testing Pass/Fail Criteria

This is up to the client team?

10 SUSPENSION CRITERIA

The automated testing procedure shall be completed in the following order:

1. Unit Testing
2. Integration Testing
3. System Testing

Each item to be tested is required to pass each unittest of a classification of 1 with 100% success before it can be included in a higher level of testing. This complete success can be waived for the following reasons:

- Feature under test will not be included in the upcoming deliverable.
- Feature under test will not be included in the final product.
- Feature under test is complex and meeting the test requirements will delay the deliverable.

Decision to allow a feature to proceed to a higher level of testing shall be determined by team leader.

11 TEST DELIVERABLES

- Test Cases
- Test Logs
- Incident Reports
- Outputs
- Corrective Actions

11.1 Test Cases

All unittests, integration tests, and system tests should be documented in TCSs. The name of a TCS should match the module name (e.g. Character Class TCS) or functions (e.g. Main Menu TCS). TCS should be developed from any requirements document and should tests every requirement of those documents.

12 REMAINING TEST TASKS

- Divsion of Labor by each group leader.
- Implementation of testing.
- Complication of testing results.

13 ENVIRONMENTAL NEEDS

None

14 STAFFING AND TRAINING NEEDS

Training on portions of the project shall be carried out by the authors of the code and the documented design and the responsibility of said authors. Help is available from other members of the team.

15 RESPONSIBILITIES

The responsibility of this document is the entire Python Group

16 SCHEDULE

Deliverable	**Description**	**Due Date**
Test Plan version 1.0	Completion of first draft of the complete Test Plan documentation.	11/12/2013
Test Assigment	Assign tests to team members	11/14/2013
Delivery of Unit Tests	Completion of assigned Unit Tests	11/25/2013
Delivery of Integration Tests	Completion of assigned Integration Tests	12/5/2013
Delivery of System Tests and Manual Tests	Completion of assigned System Tests	12/10/2013

17 PLANNING RISKS AND CONTINENCIES

Because the end of the semester is fixed, there are no contingencies if the product does not meet requirements by that date. Public beatings will be carried out as needed.

18 APPROVALS

Approval is everyones responsibility since all students will be evaluated by this document.

19 GLOSSARY

SCR - Software Change Request TCS - Test Specifications Document