**Test Plan for Bliss Bazar**

**Overview**

This test plan outlines the testing activities that have been completed for the project. It details the actions taken for unit testing, regression testing, and mutant testing. This plan serves as documentation for current and future team members to ensure that the product remains well-tested and maintainable.

**1. Unit Testing / Code Coverage**

**Objective:**

Achieve comprehensive unit test coverage for the project, ensuring that all major code components are tested, including boundary tests and invalid input handling.

**Actions Taken:**

* **Unit Tests**: We wrote and executed unit tests covering all critical parts of the code. The tests include:
  + **Boundary Tests**: We tested edge cases, such as inputs at the minimum and maximum possible values, to ensure the system handles these correctly.
  + **Invalid Input Tests**: We ensured the system handles invalid inputs, producing appropriate error messages or fallbacks.
  + **Statement and Branch Coverage**: We made sure that all statements and decision branches in the code were covered by at least one unit test.
* **Code Coverage Tools**: We used the Code Coverage Tool to track the coverage metrics. The final results showed:
  + **Statement Coverage**: 100%
  + **Branch Coverage**: 100%.

**Results:**

* **Unit Tests Passed**: All unit tests have passed successfully, confirming that the system behaves as expected.

**Future Considerations:**

* The unit tests are integrated into the Continuous Integration (CI) pipeline to ensure they run automatically with each code change.
* Developers are encouraged to write additional unit tests for new features to maintain high test coverage.

**2. Regression Testing / Build and Test**

**Objective:**

Automate the build and test process to ensure that the software under test (SUT) remains functional after every code change and that tests are executed automatically during each build.

**Actions Taken:**

We created a script named buildTestSuite, which automates the following steps:

1. **Build the Test Code and Software**: The script compiles the software under test and all necessary dependencies.
2. **Run Unit Tests**: After the build is completed, the script automatically runs the entire suite of unit tests.
3. **Email Build and Test Status**: The script sends an email with the status of the build and test results to a designated recipient. The email includes:
   * **Build Status**: True or False (Indicating if the build was successful)
   * **Test Status**: True or False (Indicating if all tests passed)

**Results:**

* The build and tests were successfully completed without errors, confirming that the system was correctly built and all tests passed.

**Future Considerations:**

* We recommend scheduling the buildTestSuite script to run daily or after each code change through a CI pipeline e.g., Jenkins, GitHub Actions.
* The script can be extended to handle more advanced testing scenarios or notify additional team members if needed.

**3. Mutant Testing**

**Objective:**

Test the ability of the existing unit tests to detect faults (mutants) by introducing controlled errors (mutants) into the code.

**Actions Taken:**

We introduced two mutants into the code as part of this testing:

1. **Mutant 1**: Changed the addition operator (+) to subtraction (-) in an arithmetic operation.
2. **Mutant 2**: Changed the multiplication operator (\*) to division (/) in another arithmetic operation.

These mutants were successfully injected into the code, and the tests were executed. Both mutants were detected and killed, as the tests failed in response to the incorrect results introduced by the mutants.

**Results:**

* **Mutant 1**: The test suite detected the change (the subtraction instead of addition) and failed, confirming that the mutant was killed.
* **Mutant 2**: The test suite also detected the incorrect multiplication operation (now division), and the test failed, confirming that the mutant was killed.

**Future Considerations:**

* Mutant testing will be periodically executed to ensure that the test suite remains effective in detecting faults.
* Additional mutants can be introduced as needed to ensure the robustness of the code and the test suite.

**Summary**

This test plan provides a comprehensive overview of the testing activities completed for the project, including unit testing, regression testing, and mutant testing. These efforts ensure that the project remains well-tested, with reliable code that is protected from regressions and subtle bugs.

**Future Steps:**

* **Continuous Integration**: Continue to integrate the test suite into a CI system to ensure that tests are run automatically with every code change.
* **Ongoing Mutant Testing**: Regularly perform mutant testing to verify that the test suite can detect potential faults in the software.