**Testing Strategy:**

**Introduction:**

This test plan is to test our software- “Music Generator”. This application is intended to generate melodies from the specifications a user provides. These specifications are in the form of themes and lead instruments that the user chooses from a list. Based on the specifications and a music theory based pattern implementation of the combination of elements as the input, the application generates a melody that is pleasant to hear and is acceptable as music in theory. The output is a song that can be generated, played, paused and saved as a midi file.

**Scope:**

This document covers all aspects of our team’s testing plan. It details out the testing strategy, approach followed, environment required for the test implementations, a comprehensive list of test cases, and a plan how the test results will be mapped to the intended and documented test plan, at the time of the final release.

**Test Approach:**

We understand that the test cases and their results are quantifiable entities. At the same time, the music generated as output is subjected to the user’s discretion of how efficiently the application produces it.

Thus, this document covers test cases that can pass or fail UI functionalities and algorithmic functionalities that the software MUST abide by. The test cases make ensure testing functionalities of the application like- does it save a song as a file when the user clicks on “save”, or- does it show an “unpause button” when a pause button is clicked.

Also, the algorithm and its implementation into methods for developing song patterns and playing them is also being tested by the automated testing tool of JUnit.

**Environment requirements:**

The GUI will be tested manually. Test cases in this document explain what is the description of the functionality that the test is performed on, what are the steps to test the functionalities and what is the expected test result when the steps are performed.

On the other hand, as a suggested best practice, we have decided to test some methods in the functionalities of the developed code of our application, through JUnit testing framework. Since our project is built in Java, JUnit is the most preferred testing method for it.

The environment required to test through JUnit consists of:  
**JDK:** 1.5 or above is required. (We used JDK version 8)

**Operating System:**

**Java:** Version 8

**JUnit Jar file:**

**Testing team:**

**Test Plan:**

We began with the testing process after a code demo and code review was done in the interim release. We tried developing the software by detailed planning and a number of meetings conducted to sketch out what the desired system would look and work like. A per the planning, we tried to test the User Interface and the functionalities of the application while we were in the Sprint 4 stage of the development process, at our own individual levels. But, none of it is compared to the rigorous test plan that categorizes and quantifies each aspect of the application to be tested and reported as pass or fail.

For carrying out the process of testing our application, we jotted down a test plan as a time boxed schedule right from Interim release to the final release phase, as shown below:

**Test Schedule:**

Table 1: Test Schedule

|  |  |
| --- | --- |
| **Dates** | **Testing activities** |
| Nov 4- Nov 7 | * Meeting conducted for test plan. * Planning the test strategies. * Division of tasks for each part of the test-plan- JUnit and Manual tests. |
| Nov 7- Nov 11 | * Setting up of framework for JUnit testing * Begin writing the source code in JUnit. |
| Nov 11- Nov 15 | * Meeting conducted to check on status of the testing plan and manual test documents. * Discussion of GUI test cases. |
| Nov 15- Nov 21 | * Implementing test cases as source code. * Finishing up manual test document. |
| Nov 21- Nov 28 | * Conducting and recording test results mapped against the test cases from the already planned test document. |

**Test Cases:**

Table 2: Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Description** | **Steps to test** | **Expected Results** | **Priority** |
| TC01 | Check if the application installs | When a user is handed an exe file to install “Music Generator”, it installs as an application on the user’s system. | 1. Double-click on the .exe file to install the application. 2. Check if the application is available to be launched in the list of Applications located in the C Drive. | An application with the name “Music Generator” is available to be launched. | Very High |
| TC02 | The application launches. | After the application is installed, it must launch displaying a user interface of specification options that the user can supply to generate a melody. | 1. Double-click on the installed application and | A screen with Grey-Background should be displayed as the application gets launched. The screen contains the title “Red Giraffe Music Generator”, a box labeled section A, and the five buttons: the “Generate”, “Load”, “Pause”, “Save” and “Exit” | Very High |
| TC03 | Verify that The user interface shows a centered heading: “Red Giraffe Music generator “ | The display screen must show the title of our application as a heading in Black font in grey background. | 1. The tester needs to look for the heading and check it’s alignment. | When the screen appears, a heading in Black fonts must be at the top center of the screen with the title of the application. | Medium |
| TC04 | Verify that the minimize/maximize/close application buttons work accordingly | Basic functionality of the application and its state is being tested through this case. | 1. The tester needs to click on each of the buttons to check if the application minimizes, maximizes and closes when the respective buttons are clicked. | It is expected that the screen works according to the buttons of minimize, maximize and close application | High |
| TC05 | Verify that the “Generate”, “Load”, “Pause”, “Save” and “Exit” buttons are aligned. | Checking the alignment of buttons shown in the user interface of the application. | 1. The tester must look for proper alignment of the five buttons. | These buttons appear at the bottom of the screen and they are similar functionalities being performed on the song being generated. They must be aligned in the interface. | Medium |
| TC06 | Check that the size of each the five button at the bottom of the screen is exactly the same and they are equally spaced. | Checking the consistency in design of the buttons. | 1. The tester must check for any inconsistencies in the design of the buttons | There must be consistency in the buttons and overall design of the screen. | Low |
| TC07 | Verify that the “add new section” button creates a box “Section B”, exactly like the box “Section A”. | There must be different sections (up to 3) to generate melodies with specifications supplied. | 1. Tester must click on the “add new section” to check that it creates new sections. 2. Tester must check if the button allows more than 3 sections, or not. | A new section up to three gets created when the “add new section” button is clicked. | High |
| TC08 | Verify that when sections B, C, and so on, are generated, the button “Add Section” disappears. | The button is applicable to create only three sections, once it reaches this threshold; the “Add Section” button must disappear. | 1. The tester must click on the button until it disappears. 2. It must disappear after the second click. | The button must disappear when three sections are created. | High |
| TC09 | Verify that the Sections Generated are parallel to the already generated boxes, equally spaced from each other and of the same size. | The design consistency of Sections is being tested by the test case. | 1. The tester must check the alignment of the sections | As part of the design consistency, the boxes of sections must be aligned and be identical. | Medium |
| TC10 | Check that the “Theme” and “Lead Instrument” labels are consistent as a list and as identical boxes in all the sections generated. | The design consistency of boxes with song specifications is being tested by the test case. | 1. The tester must check for any inconsistencies in the boxes with labels “Theme” and “Lead Instrument” | As part of the design consistency, the boxes of song specifications must be aligned and be identical | Medium |
| TC11 | Verify that when a radio button in the options of “Theme” and “Lead Instrument”, is clicked, it gets selected, as the radio button turns shaded. | The GUI display of the functionality of choosing specifications is being tested in this test case. | 1. The tester must click on the radio buttons to check if the buttons show selection by turning grey. | The radio buttons must show the user which options he/she has selected from the menu to generate a song. | Medium |
| TC12 | When the user selects a combination of specification that he/she wants to generate and clicks on the “Generate” button, a pleasant melody is played. | Functionality test of the song generation method. |  |  | Mission Critical |
| TC13 | The melody of a song needs to be checked to follow the rules | Every pattern needs to follow specific rules to display the expected melody | 1. Access the Song object 2. Access its voices 3. Access its phrases 4. Check on the combination of notes | The Pattern consists of notes should be in the specific range and follow the pattern rules | High |
| TC14 | The instruments of a song need to be checked | Different instruments have different functionalities in displaying emotions. Right instrument is vital. | 1. Access the Song object 2. Its voice 3. Check on the instruments | The instrument should be in a specific set of instruments displaying the target emotion | High |
| TC15 | Check the tempo of a song | Tempo is one of the most vital elements to lead the song which is needed to be checked | 1. Access the Song object 2. Check on its tempo | The tempo should be in the specific range | High |
| TC16 | Check the rhythm of a song is proper | Rhythm of a song is also necessary to be checked in order to make sure the song is pleasant | 1. Check the Pattern consists of notes and rhythm | The rhythm needs to be in a correct range | High |

**Test result template for final release:**

The test results will be recorded by mapping the results to each test case and documenting the results. This process will be carried out to display the efficiency of the application developed, in the final release.

As a template, following will be the columns of “Actual Results” added to the above table of test cases to find out and report the efficiency of each functionality and GUI test in a “pass/fail” criteria.  
  
**Test Results:**

Table 3: Test Results Template

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **Expected Results** | **Actual Results** | **Result of the test case** |
| TC01 | An application with the name “Music Generator” is available to be launched. | The application is gets installed once the tester double-clicks on the installation file. | Pass |