**Test Plan**

[Fid’Lin]

Version 3.0

[4/2/2019]

CPSC 4910 SP19

Prepared by:

TEAM 2

*Jakob Becklehimer, Dylan Brownell, Tanner Groll, Derek Pendleton, and Phil Snider*

1. **Introduction**

This test plan’s main purpose it to describe the approaches as well as the overall framework that will help us in testing our software. Included in the plan are: a test strategy, a table including a rough estimate of how many test cases we will need, an explanation on how we handle test management, possible criteria for passing/failing the testing phase, and also possible risks associated with testing and how we plan to mitigate said risks.

**1.1 Test Strategy**

The procedures and methods involved with our test process are all for the sake of verifying that our Fid’Lin software functions appropriately, according to our specifications. Our test will consist of several, smaller test cases that will execute and verify that all requirements are met/working accordingly as well as test cases for identifying obscure defects or errors. If errors do arise, problems will be address, fixed, and testing will be executed again.

**1.2 Test Cycles/Duration of Cycles**

There are 4 iterations of our application, however, the initial iteration does not have significant progress to warrant a test cycle. Therefore, there will be only 3 testing cycles, one starting after this first iteration due 2/7 and continuing throughout each initial iteration. This means the first testing cycle will be a month, the second 3 weeks, and the final 2 weeks. The first cycle, being the longest, will focus on testing all critical features of our application (as they are implemented). Priority will be based on first come first serve as we do not yet have enough features implemented to warrant sorting by priority. Towards the later end of the cycle is when the critical features will take priority over finishing old tests (ie: a new critical feature is released yet the previous 4 features have not been tested, then those 4 will go on hold). The first few test cases are likely to be finished in the first cycle as they can be done without the software being complete. Things like testing the menu navigation and user account creation will be in the first cycle as well as critical tests related to the metronome, playing scales and tempo. The tests that are deemed low priority will likely be handled in the second test cycle while the third test cycle will be allowed to fix any new bugs that arise towards the end of development. It is important to have ample time allocated in case critical defects late in production as those can completely derail a project.

**1.3 Test Cases**

*Since this is still early in development, these are more like a broad set of test case examples. Within the next few weeks, a much more detailed list of test cases can be written once we have a better understanding of the layout for our application.*

*All Test Cases are laid out in detail at the end of this Test Plan*

|  |  |
| --- | --- |
| **Test Case** | **Associated Feature or Requirement** |
| **1** | **Main Menu GUI tabs** |
| **2** | **Tuning GUI buttons** |
| **3** | **Practice GUI buttons** |
| **4** | **Metronome GUI buttons** |
| **5** | **Options for choosing Scales** |
| **6** | **Appropriate Options implemented for Tuning** |
| **7** | **Appropriate Options implemented for Metronome** |
| **8** | **Appropriate Options implemented for Scale Practice** |
| **9** | **Tuning notes button provides corresponding note output** |

**1.4 Test Management**

Our team is utilizing a message board known as “Trello” in order to communicate our progress, what tasks still need to be tackled, and also the importance of said tasks. We also have a section in there specifically for testing. This way, it can be managed easily as everyone can check the progress of test anytime, and the high priority tests can be marked accordingly. SImilarly, if a certain test has been eating up too much of our resources, but is not high priority, a call can be made to put that on hold and flag it as not important so that we can better allocated our time to the essential testing first.

*Note*: In addition to trello, we are utilizing a github repo so that we may all be up to date on any additions to the code and easily grab the most up to date renditions. We also use both discord and groupme for group messaging so that if any new cards are added to trello or the github is updated, we can all be informed quickly through these messaging platforms.

In this trello message board, this is where defects will be reported by the “tester” so that a “developer” can fix said defects in order to have them tested again and then finally approved. Chain of events: testing reports defects to trello, developer sees defects and fixes it and then posts to trello to notify tester, tester sees notification and then checks the github to then re test the newly fixed code, if the defect is gone then the tester marks it as approved and marks the trello card appropriately. High priority tests will be marked as such and will taken care of first.

**1.5 Test Risks**

|  |  |
| --- | --- |
| **Possible Risks** | **Mitigation Strategy** |
| **Schedule**  Chance the schedule is too tight if we get held up on design tasks. | Having good preparation for testing will make it so that it can be done much more quickly. Knowing to pull back on design tasks and simplifying a feature that is not essential will allow for appropriate testing time as well. |
| **Defects**  Finding a major defect in a late stage of our testing cycle could be disastrous. Very time consuming to solve and can delay the other cycles or can cause a deadline to be missed. | Prioritizing the identification of high priority defects, mostly by testing vital pieces to software first, will decrease the likelihood of a surprise defect arising in the final stages of testing. It would be be better to discover these early and then run out of time for the more minor testing parts. |
| **Scope**  Scope should be defined already and there should not likely be any changes or additions to the requirements, it is more if we have time to add additional features that are not necessary as a functional or non-functional requirement. | Having our first cycle focus on only the test that can be performed now and doing these tests extensively will allow us enough time in the second cycle to focus on the new issues that may arise due to the addition of new features that we may have been unsure if we had time to implement. |

**Actualized Risks:**

Our schedule and scope of our project changed. We got held up on design tasks, which caused us to shift our scope away from implementing certain features and our schedule was pushed back for testing, hence the ?/?/2019 date for many test cases. We plan to finish all these test cases between sprint 4 and the final presentation as we should have our application completed by sprint 4 (at least according to our new scope) and will be able to test all the key features in depth. Non-imperative test cases have been omitted since we will not have time for testing that is not absolutely necessary.

**Test Cases**

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 1: Main Menu GUI tabs |
| Priority | High |
| Designed by | Dylan Brownell |
| Date of Design | 2/24/2019 |
| Executed by | Tanner Groll |
| Date Executed | 2/25/2018 |
| Summary of Test | Tests the functionality of tab on the main menu to ensure that each feature can be accessed easily by the user |
| Pre-requirements | We need to implement our application code to setup a main menu framework |
| Steps | 1. Launch App 2. Select Tuning Tab 3. Select Practice 4. Select Tuning Tab 5. Select Metronome 6. Select Practice 7. Select Metronome 8. Select Tuning Tab 9. Now test all different combinations of the above steps at varying speeds |
| Expected Results | Transition between any of the tabs of the application should be smooth and quick. No errors should arise, nor should there be an extended loading time |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (2/25/2019)  - menu can change in future (retest?) |
| Automation | No |

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 2: Tuning GUI buttons |
| Priority | High |
| Designed by | Dylan Brownell |
| Date of Design | 2/28/2019 |
| Executed by | Dylan Brownell |
| Date Executed | 2/28/2019 and 3/1/2019 |
| Summary of Test | Tests the functionality of tuning tab and the interface buttons it contains |
| Pre-requirements | We need to implement our application code to setup a main menu framework as well as implement the buttons required in the tuning tab |
| Steps | 1. Launch App 2. Select Tuning 3. Test all possible variations of button inputs (not necessarily fast though) |
| Expected Results | Ensure that each button performs the appropriate action and that one session can experience all the different actions without crashing or causing errors. Does not necessarily need support for multiple button inputs or inputs in quick succession, just needs to avoid errors or crashing. |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (3/1/2019)   * tuning may change in future (retest?) |
| Automation | No |

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 3: Practice GUI buttons |
| Priority | High |
| Designed by | Dylan Brownell |
| Date of Design | 2/28/2019 |
| Executed by | Dylan Brownell |
| Date Executed | 2/28/2019 and 3/1/2019 |
| Summary of Test | Tests the functionality of practice tab and the interface buttons it contains |
| Pre-requirements | We need to implement our application code to setup a main menu framework as well as implement the buttons required in the practice tab |
| Steps | 1. Launch App 2. Select Scale Practice 3. Test all possible variations of button inputs (not necessarily fast though) |
| Expected Results | Ensure that each button performs the appropriate action and that one session can experience all the different actions without crashing or causing errors. Does not necessarily need support for multiple button inputs or inputs in quick succession, just needs to avoid errors or crashing. |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (3/1/2019)   * Scale Practice may change in future (retest?) |
| Automation | No |

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 4: Metronome GUI buttons |
| Priority | High |
| Designed by | Dylan Brownell |
| Date of Design | 2/28/2019 |
| Executed by | Dylan Brownell |
| Date Executed | ?/?/2019 and ?/?/2019 |
| Summary of Test | Tests the functionality of videos tab and the interface buttons it contains |
| Pre-requirements | We need to implement our application code to setup a main menu framework as well as implement the buttons required in the tuning tab |
| Steps | 1. Launch App 2. Select Metronome 3. Test all possible variations of bpm selections |
| Expected Results | Ensure that our metronome can handle all the varying speeds, from 30 – 210 bpm all while maintaining the accurate bpm and constistency. |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (?/?/2019)   * Video tab currently lacking buttons to test |
| Automation | No |

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 5: Options for choosing Scales |
| Priority | High |
| Designed by | Dylan Brownell |
| Date of Design | 3/3/2019 |
| Executed by | Dylan Brownell |
| Date Executed | 3/3/2019 |
| Summary of Test | Tests each scale option to ensure they work when selected as well as that all scales are present: major/minor/chromatic |
| Pre-requirements | We need to implement our application code to setup a scales practice tab as well as the GUI to input options |
| Steps | 1. Launch App 2. Select Scales Practice 3. Select options 4. Check each option with a list of all major/minor/chromatic scales to ensure we have options for each one 5. Click each option to ensure each works when selected 6. Make sure that only one scale can be selected at a time |
| Expected Results | Ensure that each button performs the appropriate action and there is not an error that arises with multiple scales being selected while also checking that we have an option to select any scale in major/minor/chromatic |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (3/3/2019)   * Need to retest whenever we link our interfaces together with the actual backend framework |
| Automation | No |

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 6: Appropriate Options implemented for Tuning |
| Priority | Medium |
| Designed by | Dylan Brownell |
| Date of Design | 3/3/2019 |
| Executed by | Dylan Brownell |
| Date Executed | ?/?/2019 |
| Summary of Test | Tests each tuning option to ensure that all buttons work when selected and we are not missing any vital tuning settings |
| Pre-requirements | We need to implement our application code to setup a tuning tab as well as the GUI to input options for said tuning tab |
| Steps | 1. Launch App 2. Select Tuning 3. Select options 4. Check all different options available to ensure each button works when selected 5. Ensure that no conflicting options can be selected simultaneously 6. Ensure they save upon switching tabs |
| Expected Results | Ensure that each button performs the appropriate action and there is not an error that arises with multiple scales being selected while also checking that we have an option to select any scale in major/minor/chromatic |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (?/?/2019)   * No time to test this sprint, instead test start of next cycle |
| Automation | No |

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 7: Appropriate Options implemented for Videos |
| Priority | Medium |
| Designed by | Dylan Brownell |
| Date of Design | 3/3/2019 |
| Executed by | Dylan Brownell |
| Date Executed | To Be Determined |
| Summary of Test | Tests each option in videos to ensure they are functional |
| Pre-requirements | We need to implement our application code to setup a videos tab as well as the GUI to input options |
| Steps | 1. Launch App 2. Select Videos 3. Select options 4. Check all different options available to ensure each button works when selected 5. Ensure that no conflicting options can be selected simultaneously 6. Ensure they save upon switching tabs |
| Expected Results | Ensure that each button performs the appropriate action and there is not an error that arises with different conflicting options being selected at once |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (?/?/2019)   * No time to test this sprint, instead test start of next cycle |
| Automation | No |

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 8: Appropriate Options implemented for Tuning |
| Priority | Medium |
| Designed by | Dylan Brownell |
| Date of Design | 3/3/2019 |
| Executed by | Dylan Brownell |
| Date Executed | ?/?/2019 |
| Summary of Test | Similar to test 4 except we are testing the other options in the tuning tab (not the scales selection) |
| Pre-requirements | We need to implement our application code to setup a tuning tab as well as the GUI to input options |
| Steps | 1. Launch App 2. Select Tuning 3. Select options 4. Check all different options available to ensure each button works when selected 5. Ensure that no conflicting options can be selected simultaneously 6. Ensure they save upon switching tabs |
| Expected Results | Ensure that each button performs the appropriate action and there is not an error that arises with different conflicting options being selected at once |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (?/?/2019)   * No time to test this sprint, instead test start of next cycle |
| Automation | No |

|  |  |
| --- | --- |
| **Test Case Field** | **Description** |
| Test Case # | 9: Tuning notes button provides corresponding note output |
| Priority | Medium |
| Designed by | Dylan Brownell |
| Date of Design | 3/3/2019 |
| Executed by | Dylan Brownell |
| Date Executed | ?/?/2019 |
| Summary of Test | Tests functionality of the notes that must be played for tuning. Compare them to other tuning functions or applications to gauge accuracy of our tuning buttons |
| Pre-requirements | We need to implement our application code to setup a tuning tab along with it’s appropriate interface and implement the methods that can make up the tuning class |
| Steps | 1. Launch App 2. Select Tuning 3. Select note for tuning 4. Compare output to other applications notes 5. Ensure that note output is consistent upon multiple selections and differing inputs |
| Expected Results | Ensure that each button plays the appropriate note and is consistent across all notes |
| Pass or Fail | Complete Pass (?/?/2019)  Partial Pass (?/?/2019)   * These methods have not yet been implemented into our application |
| Automation | No |