SE 3XA3: Test Plan
Mastermind

Team #204, Trident Inc.
Justin Rosner, rosnej1
Justin Prez, prezj
Harshil Modi, modih1

April 5, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
February 28, 2020	1.0	Initial version of the test plan document
April 5, 2020	2.0	Revision 1 of Test Plan document

1 General Information

1.1 Purpose

The purpose of implementing a test plan for Mastermind is to ensure that all of the functional and non-functional requirements outlined in the software requirement specification document are implemented correctly. Additionally, it serves as a means to provide assurance to the customer on the correctness of the product.

1.2 Scope

This test plan will lay the groundwork to ensure that all specifications mentioned in the functional and non-functional requirements are implemented properly by creating a test plan for the individual modules and functions of the application. The modules will not only be tested individually for correctness, but also as a whole system so that the development team can be confident that cohesion between the modules is implemented correctly.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software Requirement Specification
GUI	Graphical User Interface
PoC	Proof of Concept

Table 3: **Table of Definitions**

Term	Definition
Dart	A programming language for creating multi-platform apps
Flutter	a Dart SDK for creating native interfaces on Android and iOS
Widget	A feature of the Dart language that allows for visual aspects to be implemented

1.4 Overview of Document

This document will ~~go in detail into the~~ provide the steps necessary to create a successful test plan for the game of mastermind as the team re-implements the open source project 'open-mastermind'.

2 Plan

2.1 Software Description

The software is a re-creation of the famous 1970's game Mastermind, in which a user has to try and guess the proper placement and colour of a row of pegs in a limited amount of guesses. The implementation of the game is a mobile application created using Dart and Flutter.

2.2 Test Team

The test team consists of Harshil Modi, Justin Prez, and Justin Rosner. Each member will be responsible for a specific sections of the software to test while also ensuring the software works as a whole.

2.3 Automated Testing Approach

The team will utilize automated testing approaches to test each version of the software implemented. Flutter is rich with testing libraries which will be used to test the implementation. These libraries include suites for unit testing, widget testing and integration testing. Unit testing will be used to

ensure that the general logic of the game and implemented functions are correct. Widget testing will cover each individual module and the correct display onto the application. This is an example of automated testing as each widget test will ensure that elements displayed to the user are in their correct position without any human intervention. Finally, integration testing will be used to test the UI of the application (Flutter Driver) on both IOS & Android and different screens sizes (tablets and mobiles).

2.4 Testing Tools

The testing tools that will be utilized is a combination of package:test and test_coverage from the dart language, and the flutter_test Library. Within these libraries there are specific tools for unit tests, widget tests, and UI integration testing that ~~our~~ the group will make use of. Additionally, ~~our~~ the group will make use of the test_coverage tool to generate code coverage results.

2.5 Testing Schedule

Please refer to the Gantt Chart to see the schedule for the testing plan.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User Input

Peg Selection Commands Please refer to the image of the game board in the appendix for clarity on the game board.

1. test-UI1: Test selection of blue colour peg

   Type: Functional, Dynamic, Manual

   Initial State: ~~The game board has a spot available for a peg to be placed.~~ The User Row Peg Input has pegs in slots 0..i where i <3.

   Input: ~~The blue coloured peg is selected from the peg options.~~ The user taps the blue coloured peg from the list of peg options.

Output: ~~The game board will be updated with a blue peg in the next available peg location.~~ A blue peg is placed in position i + 1

How test will be performed: A visual test will confirm that the game board now contains a blue peg in the next available peg location. Additionally a flutter-test can be performed to ensure that the row gets updated with the correct peg.

2. test-UI2: Test selection of yellow colour peg

   Type: Functional, Dynamic, Manual

   Initial State: ~~The game board has a spot available for a peg to be placed.~~ The User Row Peg Input has pegs in slots 0..i where i <3.

   Input: ~~The yellow coloured peg is selected from the peg options.~~ The user taps the yellow coloured peg from the list of peg options.

   Output: ~~The game board will be updated with a yellow peg in the next available peg location.~~ A yellow peg is placed in position i + 1

   How test will be performed: A visual test will confirm that the game board now contains a yellow peg in the next available peg location. Additionally a flutter-test can be performed to ensure that the row gets updated with the correct peg.

3. test-UI3: Test selection of red colour peg

   Type: Functional, Dynamic, Manual

   Initial State: ~~The game board has a spot available for a peg to be placed.~~ The User Row Peg Input has pegs in slots 0..i where i <3.

   Input: ~~The red coloured peg is selected from the peg options.~~ The user taps the red coloured peg from the list of peg options.

   Output: ~~The game board will be updated with a red peg in the next available peg location.~~ A red peg is placed in position i + 1

   How test will be performed: A visual test will confirm that the game board now contains a red peg in the next available peg location. Additionally a flutter-test can be performed to ensure that the row gets

updated with the correct peg.

4. test-UI4: Test selection of green colour peg

   Type: Functional, Dynamic, Manual

   Initial State: ~~The game board has a spot available for a peg to be placed.~~ The User Row Peg Input has pegs in slots 0..i where i <3.

   Input: ~~The green coloured peg is selected from the peg options.~~ The user taps the green coloured peg from the list of peg options.

   Output: ~~The game board will be updated with a green peg in the next available peg location.~~ A green peg is placed in position i + 1

   How test will be performed: A visual test will confirm that the game board now contains a green peg in the next available peg location. Additionally a flutter-test can be performed to ensure that the row gets updated with the correct peg.

5. test-UI5: Test selection of white colour peg

   Type: Functional, Dynamic, Manual

   Initial State: ~~The game board has a spot available for a peg to be placed.~~ The User Row Peg Input has pegs in slots 0..i where i <3.

   Input: ~~The white coloured peg is selected from the peg options.~~ The user taps the white coloured peg from the list of peg options.

   Output: ~~The game board will be updated with a white peg in the next available peg location.~~ A white peg is placed in position i + 1

   How test will be performed: A visual test will confirm that the game board now contains a white peg in the next available peg location. Additionally a flutter-test can be performed to ensure that the row gets updated with the correct peg.

6. test-UI6: Test selection of purple colour peg

Type: Functional, Dynamic, Manual

Initial State: ~~The game board has a spot available for a peg to be placed.~~ The User Row Peg Input has pegs in slots 0..i where i <3.

Input: ~~The purple coloured peg is selected from the peg options.~~ The user taps the purple coloured peg from the list of peg options.

Output: ~~The game board will be updated with a purple peg in the next available peg location.~~ A purple peg is placed in position i + 1

How test will be performed: A visual test will confirm that the game board now contains a purple peg in the next available peg location. Additionally a flutter-test can be performed to ensure that the row gets updated with the correct peg.

7. test-UI7: Test simultaneous button presses

   Type: Functional, Dynamic, Automated

   Initial State: The Mastermind game is open to the game board.

   Input: The user selects two pegs at the same time.

   Output: The game does not update the board until only one of the buttons is pressed.

   How test will be performed: A tester will play the Mastermind game, and continuously press two buttons/pegs at the same time trying to generate an error.

8. test-UI8: Test undo button

   Type: Functional, Dynamic, Manual

   Initial State: The Mastermind game is open to the game board, and User Row Peg Input has pegs in slots 0..i where i <3.

   Input: The user selects the undo button

   Output: The peg at position i is removed from the game board.

   How test will be performed: A tester will play the Mastermind game, and in doing so will select a peg, and then immediately remove it from the board to ensure that the button works properly.

Menu Navigation Commands

1. test-UI9: Test ~~help screen button~~ game instructions are displayed to the user upon start up of the game

   Type: Functional, Dynamic, Manual

   Initial State: The initial state is that the user ~~is on mobile device about to enter the MasterMind application~~ has just opened up the game on their mobile device.

   Input: The user selects the ~~'help' button from the available buttons at the top of the playing screen~~ MasterMind application and the game loads onto the users mobile device.

   Output: ~~A screen will pop up with the rules of the game and instructions on how to select pegs to make guesses~~ The start screen of MasterMind along with the instructions on how to play the game will be displayed to the user.

   How test will be performed: A visual test will confirm that the ~~help page is~~ instructions are displayed to the user.

2. test-UI10: Test close game button

   Type: Functional, Dynamic, Manual

   Initial State: The initial state is that the user is viewing the game board (of any difficulty) at any point throughout the game.

   Input: The user will select the 'x' button in the top right corner of the game.

   Output: The user will be directed back to the initial start menu for Mastermind.

   How test will be performed: A visual test will be performed to ensure that the user is properly directed back to the start menu of the game.

3. test-UI11: Test new game button

   Type: Functional, Dynamic, Manual

Initial State: The game is initially on the start menu.

Input: The user selects the 'new game' button.

Output: The user is directed to a ~~screen where they will now have the ability to select a difficulty level for the new game~~ a new board game.

How test will be performed: A visual test will be performed to ensure that the user is properly directed to the ~~'select difficulty' screen~~ a new board game.

4. ~~test-UI10: Test 'easy difficulty' button~~
   ~~Type: Functional, Dynamic, Manual Initial State: The game is initially on the select difficulty screen. Input: The user selects the 'easy difficulty' button. Output: The game directs the user to a new board game with 3 peg slots per row, where the user can now begin to play the game. How test will be performed: A visual test will be conducted to ensure that the user is properly directed to a new game board, with the game board having 3 peg slots per row (This is the criteria for the easy difficulty).~~

5. ~~test-UI11: Test 'medium difficulty' button~~
   ~~Type: Functional, Dynamic, Manual Initial State: The game is initially on the select difficulty screen. Input: The user selects the 'medium difficulty' button. Output: The game directs the user to a new board game with 5 peg slots per row, where the user can now begin to play the game. How test will be performed: A visual test will be conducted to ensure that the user is properly directed to a new game board, with the game board having 5 peg slots per row (This is the criteria for the medium difficulty).~~

6. ~~test-UI12: Test 'hard difficulty' button~~
   ~~Type: Functional, Dynamic, Manual Initial State: The game is initially on the select difficulty screen. Input: The user selects the 'hard difficulty' button. Output: The game directs the user to a new board game with 6 peg slots per row, where the user can now begin to play the game. How test will be performed: A visual test will be conducted to ensure that the user is properly directed to a new game board, with the game board having 6 peg slots per row (This is the criteria for the hard difficulty).~~

3.1.2 Game Environment

1. test-GE1: Test load start menu upon startup

   Functional, Dynamic, Manual

   Initial State: A smartphone with Android or iOS is on and running.

   Input: The user selects to open the Mastermind application.

   Output: The phone's GUI loads the game and displays to the user the start menu of Mastermind.

   How test will be performed: A visual test can be done to see if the start menu loaded properly upon opening of the game.

2. test-GE2: Test load ~~help menu~~ instructions upon startup

   Type: Functional, Dynamic, Manual

   Initial State: A smartphone with Android or iOS is on and running ~~, and the Mastermind game is open to the home screen~~.

   Input: The user presses ~~the help icon~~ Mastermind application to open the app.

   Output: The ~~help~~ start screen with the instructions on them are is displayed to the user.

   How test will be performed: A visual test can be done to see if the start menu loaded properly upon selecting the help icon.

3. test-GE3: Test load game board

   Type: Functional, Dynamic, Manual

   Initial State: A smartphone with Android or iOS is on and running, and the Mastermind game is open to the home screen

   Input: The user selects the play icon.

   Output: The game begins and the board is displayed to the user.

   How test will be performed: A visual test can be done to confirm that the game board is loaded to the screen.

3.1.3 Game Logic

1. test-GL1: Test full peg row

   Type: Functional, Dynamic, Manual

   Initial State: The initial state is that the user is viewing the game board ~~(of any difficulty) with~~ and the most current row ~~is~~ full with no room to add additional pegs. Please see the appendix for a labeled description of the game board.

   Input: One of the pegs will be selected to add to the game board.

   Output: The peg will be added to the next row, coming after the row that is already full.

   How test will be performed: A visual test can be done to confirm that the peg was added to the proper row. Additionally a flutter-test can be performed to ensure that the row gets updated with the correct peg.

2. test-GL2: Test incorrect guess (that doesn't finish the game)

   Type: Functional, Dynamic, Manual

   Initial State: The initial state is that the user is viewing the game board ~~(of any difficulty)~~ with the most current row having one availability for a peg to be placed. This most current row will also be in a state such that no matter which peg is placed in the last spot, ~~it will not make the row correct.~~ the row pattern will not match the code pattern.

   Input: The user will select one colour of peg to place into the open position in the row.

   Output: The game will display how many colours/positions are correct, and then proceed to allow for the user to start adding pegs to the next row.

   How test will be performed: A flutter-test suite can be written to ensure that the proper output is produced upon an incorrect guess that does not end the game. Additionally, a visual test will allow for confirmation that the user is able to start placing pegs in the next row.

3. test-GL3: Test win game

Type: Functional, Dynamic, Manual

Initial State: The initial state is that the user is viewing the game board (of any difficulty) with the most current row having one availability for a peg to be placed. This most current row will also be in a state such that all pegs currently in the row are correct.

Input: The user selects the correct peg to add to the row.

Output: The user will get a message saying that they have won the game.

How test will be performed: A flutter-test suite can be written to ensure that the row is in fact the correct combination of pegs. A visual test will also be performed to ensure that the proper winning message is shown to the user.

4. test-GL4: Test lose game

   Type: Functional, Dynamic, Manual

   Initial State: The initial state is that the user is viewing the game board (of any difficulty) with the most current row having one availability for a peg to be placed. This most current row will also be in a state such that no matter which peg is placed in the last spot, it will not make the row correct. Additionally, this is the last available row on the game board.

   Input: The user selects any peg to fill the last position on the board.

   Output: The user will get a message saying that they have lost the game.

   How test will be performed: A flutter-test suite can be written to ensure that the row is an incorrect combination of pegs. A visual test will also be performed to ensure that the proper losing message is shown to the user.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. test-LF1: Test that the game board resembles the classic game board

from the 70's

Type: Functional, Dynamic, Manual

Initial State: A smartphone with Android or iOS is on and running Mastermind.

Input/Condition: The game resembles the classic Mastermind game with the same rules.

Output/Result: The game resembles Mastermind.

How test will be performed: Twenty or more people will individually load the game and play long enough to conclude that the game resembles Mastermind.

3.2.2 Usability and Humanity Requirements

1. test-UHR1: Test that the game shall be playable by users MIN_AGE and up

   Type: Functional, Dynamic, Manual

   Initial State: The game is loaded and given to a child and elder to play.

   Input/Condition: Both the child and elder will play on the same difficulty and should be able to understand how to play.

   Output/Result: The child and elder are able to play Mastermind

   How test will be performed: Team members will have relatives (old and young) to play the game and record their opinion on the difficulty of playing and understanding the game. If no difficulties are reported, the test passes.

2. test-UHR2: Test that users will remember how to play the game.

   Type: Functional, Dynamic, Manual

   Initial State: The game is loaded and given to a user who has previously played Mastermind.

   Input/Condition: The user will be capable of playing without any instructions.

Output/Result: The user will be able to play the game based on their previous knowledge.

How test will be performed: Mastermind will be given to a user who has previously played mastermind before. The user will not be given any instructions on how to play. The test passes if they can complete a game without any prior knowledge on how to play.

3.2.3 Performance

1. test-P1: Speed and Latency

   Type: Functional, Dynamic, Manual

   Initial State: The Mastermind game is open to the home screen.

   Input: The game ~~interface~~ system shall respond to each user input in under MAX_LATENCY.

   Output: The game ~~should~~ responds to ~~all~~ each of the user's inputs within MAX_LATENCY.

   How test will be performed: A tester will launch the game and select a variety options and attempt to carry out a game. The tester will visually observe if there is any significant latency during this interactive process.

2. test-P2: Precision and Accuracy

   Type: Functional, Dynamic, Manual

   Initial State: The Mastermind game is open to the home screen.

   Input: User input should translate directly to the intended output within the context of the game.

   Output: The user will experience no unintended in-game glitches during their experience playing.

   How test will be performed: A tester will play the mastermind game several times, ~~changing difficulties and options,~~ starting and closing the game to ensure that all actions perform as intended.

3. test-p3: System Stress Test

   Type: Functional, Dynamic, Automated

   Initial State: The Mastermind game is open to the game board.

   Input: The user continuously selects pegs, with one coming immediately after another.

   Output: The game handles the speed of the inputs, and properly displays the pegs on the game board.

   How test will be performed: A tester will play the Mastermind game, and continuously spam/click a button until they run out of guesses to make.

3.2.4 Operational and Environmental

1. test-OE1: The game can be loaded on an Android system

   Type: Dynamic, Manual

   Initial State: An Android emulator or Android smartphone loads the game.

   Input: Attempt to load Mastermind on the Android smartphone/emulator and perform the tests described above.

   Output: The game should run as expected.

   How test will be performed: The tester will download and run the game on their Android emulator or smartphone. If they can successfully load and play the game, the test passes.

3.3 Traceability Between Test Cases and Requirements

The following table is a traceability matrix that show how the above test cases relate to the functional requirements.

Test Case	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
UI1		X						X						
UI2			X					X						
UI3				X				X						
UI4					X			X						
UI5						X		X						
UI6							X	X						
UI7								X						
UI8														X
UI9									X					
UI10									X					
UI11									X					
GE1									X					
GE2									X					
GE3									X					
GL1								X			X	X		
GL2								X			X	X		
GL3								X		X	X	X		
GL4								X		X	X	X	X	
OE1	X													

Table 4: **Mapping of functional requirements to test cases**

Test cases that test the non-functional requirements are labelled in section 3.2. (e.g. test-LF1 tests the non-functional requirement LF1 from the SRS)

Please note that any non-functional requirements not included in the traceability matrices are not testable by running just the source code.

4 Tests for Proof of Concept

The purpose of the proof of concept (PoC) was to demonstrate that a simple UI for the application could be displayed on an Android emulator that would respond to user input. The goal of the PoC was to ensure that the final

product is viable to build. Testing the PoC is similar to the testing of the game environment described in test-GE1, and the peg selection commands described in test-UI1 - test-UI6.

4.1 Game Launch Testing

1. test-GL1: Testing the ability to launch the application within an Android emulator.

   Type: Dynamic, Manual

   Initial State: An Android emulator is running locally on the tester's computer.

   Input: The user runs the source code to launch the Mastermind PoC application.

   Output: The Android emulator loads the the PoC application.

   How test will be performed: The tester will visually test that the game successfully appeared within the emulator window.

4.2 UI Response Testing

1. test-UIR1: Testing that the UI can recognize input.

   Type: Functional, Dynamic, Manual

   Initial State: The PoC application is loaded into the emulator.

   Input: The user interacts with a button on the application.

   Output: A counter on the screen increments to indicate that the user's button press has been registered.

   How test will be performed: A visual test will be performed to ensure that that upon pressing the button the counter increments accordingly.

5 Unit Testing Plan

The Flutter test library will be used for testing throughout the development process.

5.1 Unit testing of internal functions

Each and every function within the implementation will be tested using Flutter test framework and a combination of both structural and functional testing. The function under testing will either return a value or throw an exception. The returned value or exception will be compared to the intended result determined by us prior to testing. Hence, with the help of Unit testing, ~~we~~ the team will be able to ensure that each function works as intended to. It is vital to know that the individual functions are working as intended because after unit testing, ~~we~~ the team will move on to integration testing. In this phase of testing, ~~we~~ the team are checking to make sure that the modules/functions are communicating properly between themselves. Additionally ~~we~~ the team will be using the results from the code coverage tests to help identify redundant code, and to point out any issues with branch/statement/path coverage in ~~our~~ the existing test cases.

5.2 Unit testing of output Files

Given that this implementation of Mastermind is an interactive GUI, there are not output files that are generated, and thus none that can be tested. Instead, the development team will rely heavily on manual visual tests as well as unit tests to ensure the game board operates with correctness. Visual tests will confirm that when a peg or button is pressed that the proper response is generated. Unit tests will verify that the proper data is stored in the board upon button presses, as well as to verify winning and losing positions.

5.3 Unit testing of User interface

To test the UI, ~~we~~ the team will be using the Flutter Driver (part of flutter_test) to ensure that the buttons work properly. To do this, the driver will be selected to click a section of the screen and check for change in widget. If the intended widget closes or pops open, the test is successful. Next, Flutter driver will be used to check locations of texts with the help of the function 'find.byValueKey('text')' to get the position of the text on the screen relative to other widgets based on our UI specification.

6 Appendix

6.1 Figures

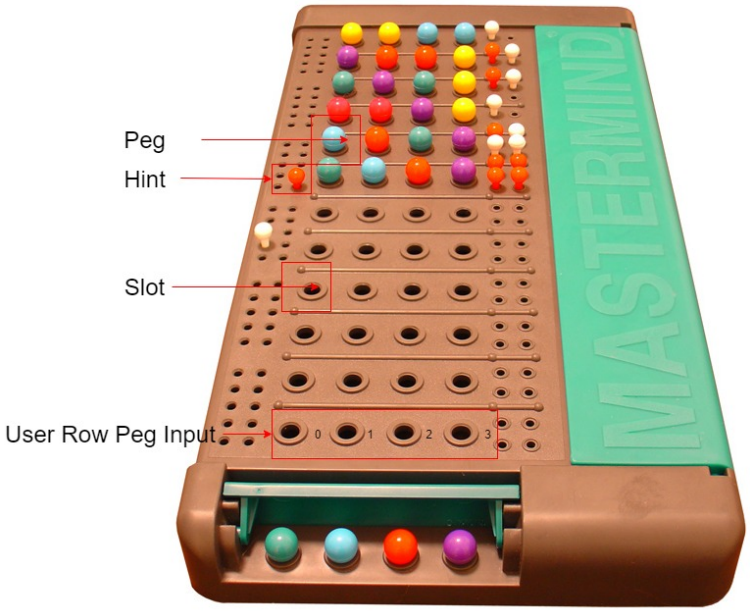

Figure 1: **Mastermind Game Board**

6.2 Symbolic Parameters

Table 5: **Symbolic Parameters**

Symbol	Value	Meaning
MIN_AGE	4	The minimum edge to play Mastermind
MAX_LATENCY	35ms	The maximum time it takes for the application to respond to input

6.3 Usability Survey Questions

The development team of Mastermind will make use of a usability survey to determine if the implementation is completed in accordance to the non-functional requirements. In addition this will also check if ~~our~~ the implementation is as usable as the command line mastermind that the project is based off of. Example questions from the survey are as follows:

1. In your opinion are the pegs intuitive to add to the board?
2. On a scale of 1-10 how easy did you find it to navigate the different menus?
3. At any point in playing the game were you confused with how to use the application, and if so, at what point?
4. On a scale of 1-10 how easily were you able to find the rules of the game?