SE 3XA3: Test Report
Scrabble Project

Team #214, The Trifecta
Kanakabha Choudhri, choudhrk
Raymond Tu, tur1
Lucia Cristiano, cristial

April 5, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
2020/04/01	1.0	Functional Requirements Completed
2020/04/05 2	1.1	Test Report Completed

1 Functional Requirements Evaluation

This document elaborates on the Scrabble testing results. Any changes made to the modules as a result of testing are identified, as well traces the test results to the Scrabble Modules with a traceability matrix.

- test-UI:1 Test rack size after exchanging tiles
- Results: The player rack is always 7 tiles after exchanging any number of tiles.
- test-UI2: Test labels update correctly with player names
- Results: The turn and score label correctly update with the inputted player names. The turn and score labels update after a valid move is made, a turn is skipped or tiles are exchanged.
- test-UI3: Inputted player names only have string characters
- Results: All inputted player names have the appropriate variable type.
- test-UI4: Test rack size after turn
- Results: Rack size is always 7 tiles are a turn is completed, which is the defined correct size.
- test-UI5: Test valid word
- Results: All successful moves have valid words that can be found in the dictionary file.
- test-UI6: Test invalid length of word
- Results: All successful word inputs were within the boundary of the board and had the correct length base don player rack.
- test-UI7: Test word not found in dictionary
- Results: When the inputted word was not in the dictionary file the move was rejected as an invalid move.
- test-UI8:Test valid dimensions and direction

- Results: Valid moves only accepted the correct direction inputs right and down. All valid moves fit within the boundaries of the board.
- test-UI9:Test invalid row and valid direction
- Results: Moves where incorrect directions were given, ie. left and up were rejected as invalid moves. Moves outside of the boundaries of the board were rejected.
- test-UI10:Test invalid column and valid direction
- Results: Moves where the direction was valid, ie. right and down but the column was outside of the boundaries of the board were still rejected as invalid moves.
- test-UI11:Test valid row, valid column and invalid direction
- Results: Tests where the row and column were within the boundary of the board, but the direction was invalid were rejected as invalid moves.
- test-UI12:Test for when the word inputted vertically forms a valid word with existing words/letters
- Results: Tests where words inputted vertically (ie. direction down) and formed other valid words were successfully accepted as valid moves with score updated accordingly.
- test-UI13:Test for when the word inputted vertically forms an invalid word with existing word/letter.
- Results: Tests where the input word is valid, and direction is down, but adjacent words formed are invalid are still rejected as invalid moves.
- test-UI14:Test for when the word inputted horizontally forms a valid word with existing word/letter.
- Results: Tests where the input word is valid, and direction is right, but adjacent words formed are valid are accepted as valid moves with the score updated accordingly with the new formed words.
- test-UI15:Test for when the word inputted horizontally forms an invalid word with existing word/letter.

- Results: Tests where the input word is valid, and direction is right, but adjacent words formed are invalid are still rejected as invalid moves.
- test-UI16:Test for standalone word score.
- Results: The score was correctly calculated for a word in a valid move.
- test-UI17:Test for inputted word sequence attached to existing word.
- Results: Similar to test results for UI12-15, the shared letters were correctly recognized with the correct output (score).
- test-UI18:Test for removal of rack when turn is finished.
- Results: After a valid move, the tiles used in the rack are removed and replaced to the correct rack size of 7 tiles from the bag.
- test-UI19:Test for end game state based on allowed words.
- Results: When there are no more possible moves to be made on the board, the win state is correctly checked and the score board appears with the winner of the game.
- test-UI20:Test for end game state based on internal bag count
- Results: When the bag has no more tiles (bag size is 0) the win state is checked and the scoreboard appears with the winner of the game.
- test-GE1: Test rack size after initial board loading
- Results: After the game starts, in the initial turn the first player has the correct number of tiles in their rack, with 7 tiles.
- test-GE2: Test board size and width after initial board loading
- Results: The board size and width remain constant throughout the game after any number of turns, with 14 tiles width and height.
- test-GE3: Test premium square locations after initial board loading
- Results: The premium square locations after any correct move remain constant, when a tile is placed on a premium square the label changes to display the letter tile.

- test-GE4: Test score count display after valid word is entered.
- Results: After a valid word is entered and placed on a normal tile or premium tile the score is correctly updated and appended on the score display.
- test-GE5: Test that upon arrival of the end state game exits board.
- Results: When the game reaches a win state, the score board is displayed and the game exits.

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel

- test-LF1: Users find interface warm and welcoming
- Results: The group of testers gave positive feedback regarding the aesthetics of the game, noting it reminded them of the original Scrabble board game and had a colour scheme that was easy to look at.

2.2 Usability

- test-UH1: Usable by MIN_AGE
- Results: Testers of MIN_AGE reported the application was easy to use, no other documentation was needed other than the one provided with the game. The overall rating was a 7 on a scale out of 10.
- test-UH2: Test Recognizable Layout
- Results: Peer testers were given the project and rated the game as simple to play and very intuitive after being provided documentation in the instructions. The overall rating average for all the users was 8.3 out of 10.
- test-UH3: Test Need for Tutorial

- Results: Peer testers were given the project and rated the game as simple to play and very intuitive after being provided documentation in the instructions. The overall rating average for all the users was 8.0 out of 10.

- test-UH4: Test Previous Players

- Results: The project developers who were experienced with the Scrabble game were able to easily remember the overall layout of the Scrabble game. 90% of users gave a rating of six or higher on a scale out of 10.

2.3 Performance

- test-PR1: Test for Turn Change Speed

- Results: After the tester inputted a valid word and direction and hit end move, they visually verified that the board was correctly updated within $MIN\_TURN\_CHANGE\_TIME$.

- test-PR2: Test for Playable State Between Updates

- Results: A tester tested the game was in a playable state between every update on the master branch, and confirmed it was playable and exhibited correct behaviour in validating moves. This was achieved through the development branch in the project handling all the bug-fixing and only pushing working copies to the master branch.

2.4 Operational and Environmental

- test-OE1: The Scrabble game takes no more than 20 minutes to install onto the host computer.

- Results: All testers were able to download and setup the Scrabble repository on their computer within 20 minutes.

- test-OE2: The Scrabble game takes no more than 10 seconds to load.

- Results: All testers were able to launch the Scrabble game within 10 seconds after running it from the command line terminal.

2.5 Maintainability and Support

- test-MS1: Test for Maintainability
- Results: The focus group for programmers rated the maintainability of the Scrabble game as a 4.1 due to it's modularity and low coupling between modules.
- test-MS2: Test for Implementation of New Features
- Results: The focus group of software design experts rated the ease of new feature implementation an 8.3. This is due to the MVC architecture of the Scrabble game easily allowing new modules or features to be added.
- test-MS3: Test for Playable State Between Python Updates
- Results: The questionnaire provided to a sample set of users returned an 8.4 out of 10 average score for the playability of the game between Python updates.

3 Comparison to Existing Implementation

In comparison to the original implementation of the Scrabble game, changes in the architecture of the program have improved the ease of testing and maintainability of the program. This is because in the original implementation all modules, including the model for the Tile, Bag, Rack, Player and Board class were included in a single Python main file that numbered nearly 1000 lines of code. This made it difficult to read and understand the code, as well made implementing new features or finding sources of bugs difficult. Because of this, along with the original goal of implementing a user interface using the Tkinter library, the original code was rewritten and split up into several modules following MVC architecture. This greatly enhanced the readability of the code, made implementing new features such as exchange tiles and additional board checks easier, as well made testing the code more feasible.

4 Unit Testing

Due to the nature of the Scrabble program, where the bag and rack of the player are random and the types of input are non-deterministic, it was infeasible to conduct unit testing for the majority of the modules. Because the majority of the program was tested through the front-end interface in Tkinter, most testing was exploratory testing where the aim was to cover the majority of situations that would occur in regular Scrabble gameplay (for example, placing a word that created more words). For more information regarding the back-end during execution of the moves during a playthrough, print statements to the terminal were used and evaluated after each test. During development, unit testing was conducted on modules in the Model classes, so Bag.py, Rack.py, Board.py, Player.py and Tiles.py. This was conducted with PyTest and was prior to developing the back-end for the Scrabble game to ensure it was correct before connecting it with the front-end. After the front-end to back-end implementation was complete the majority of testing was manual testing through the front-end.

5 Changes Due to Testing

5.1 Functional Requirements Evaluation

There were several changes made due to the testing of functional requirements. The major sources of issues were in boardChecks.py and wordChecks.py modules. The issues were found through manual testing where there were scenarios found that placing a word that connected with other words adjacently (for example, placing a word down could create short words from left to right horizontally), often these created words were not in the dictionary however the game still counted them as valid moves. These were in test cases UI12-UI15. This resulted in the creation of a new function in the boardChecks.py module, adjWordCheck that checks for the creation of these new words, and whether they are in the dictionary as well. This was done through iterating through the board for every tile and checking whether any combination of tiles found were in the dictionary through checkInDict in wordChecks.py. This was successful, and after 20 test cases where the tester deliberately tried to invoke this scenario, all 20 test cases passed.

5.2 Look and Feel Evaluation

There were no overall changes made to the user interface due to tests in the Look and Feel category.

5.3 Usability

There were no overall changes made to the user interface due to tests in the Usability category.

5.4 Performance

There were no overall changes made to the user interface due to tests in the Performance category.

5.5 Operational and Environmental

There were no overall changes made to the user interface due to tests in the Operational and Environmental category.

5.6 Maintainability and Support

There were some changes made during development due to the test cases MS1 and MS2 in Maintainability and Support. Our team realized that when receiving feedback and evaluating our own code the way we were implementing the front-end to back-end module did not have high cohesion and would make maintainability of the code difficult in the future. This was due to problems getting these modules to connect, resulting in them being combined as a temporary stop-gap to then test the correctness of the back-end implementation through functional testing. Later on we came back to this and restructured the module to have a separate front-end and back-end implementation, seen as main.py for the front-end and gameController.py for the back-end. This was successful and resulted in higher cohesion in the code and increased modularity, this made further testing simpler and easy to make changes to revise the code for the future. This overall improvement in the design of our program allowed us to get better user feedback and pass these test cases.

6 Automated Testing

Automated Testing was not originally planned for testing the Scrabble project due to the majority of testing being through the front-end interface, and the inputs not being constant or predictable. Due to the nature of receiving the output of the program through the front-end, automated testing would be infeasible and provide little benefit to proving the correctness of the implementation of the Scrabble game. Because there would be no benefit to validate the project through automated testing, the team concluded the cost of automated testing did not provide enough benefit and focused our efforts on manually testing the program.

7 Trace to Requirements

This section is a traceability matrix between the test cases and the functional and non-functional requirements outlined in the SRS.

Table 2: Traceability for Functional Requirements

Functional Requirement	**Test Cases**
FR2	test-GE1, test-GE2
FR4	test-UI2, test-UI3
FR5	test-UI1, test-UI4, test-GE1
FR6	test-UI5, test-UI6
FR7	test-UI6, test-UI7
FR8	test-UI8, test-UI9, test-UI10, test-UI11
FR9	test-UI12, test-UI13, test-UI14, test-UI15
FR10	test-UI16, test-UI17, test-GE4
FR11	test-UI18
FR12	test-UI19, test-UI20, test-GE5

Table 3: Traceability for Non-Functional Requirements

Non-Functional Requirement	**Test Cases**
NFR LF1	test-LF1
NFR UH1	test-UH1
NFR UH2	test-UH2
NFR UH4	test-UI3
NFR UH5	test-UH4
NFR PR1	test-PR1
NFR PR2	test-PR2
NFR OE1	test-OE1
NFR OE2	test-OE2
NFR MS1	test-MS1
NFR MS2	test-MS2
NFR MS3	test-MS3

8 Trace to Modules

This section shows the traceability matrix between the test cases and the modules. The modules are Main Game (M1), BoardChecks (M2), EndTurn (M3), WordChecks (M4), GameController (M5), Board (M6), Player (M7), Rack (M8), Tiles (M9), Bag (M10), WidgetCreation (M11).

Test Cases	Modules
test-UI1	M8
test-UI2	M1, M11
test-UI3	M1, M4
test-UI4	M8
test-UI5	M1, M2, M4, M5
test-UI6	M1, M2, M4, M5
test-UI7	M4, M5
test-UI8	M2, M5
test-UI9	M2, M5
test-UI10	M2, M5
test-UI11	M2, M5
test-UI12	M2, M4, M5
test-UI13	M2, M4, M5
test-UI14	M2, M4, M5
test-UI15	M2, M4, M5
test-UI16	M1, M3, M5
test-UI17	M2, M5
test-UI18	M8, M5
test-UI19	M3, M1, M2, M5
test-UI20	M3, M1, M2, M5
test-GE1	M1, M8
test-GE2	M1, M6, M2
test-GE3	M1, M6
test-GE4	M1, M2, M4, M5
test-GE5	M1, M3, M5
test-LF1	M1, M11
test-UH1	M1, M11
test-UH2	M1, M11
test-UH3	M1, M11
test-UH4	M1, M11
test-PR1	M1, M3
test-PR2	M1, M11
test-OE1	M1
test-OE2	M1
test-MS1	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11
test-MS2	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11
test-MS3	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11

Table 4: Trace Between Test Cases and Modules

9 Code Coverage Metrics

Our team has managed to produce roughly 92-95% code coverage through these test cases. This estimation is derived from inserting print statements throughout the entire program and evaluating the output to the terminal after all the test cases. This coverage metric includes statement coverage and branch coverage due to the placement of the print statements, branch coverage is covered by evaluating all the branches any given move can take. Due to the extensive manual testing for all modules in the program, the Trifecta team can confidently say that almost all of the statements and branches in the program are tested. These metrics are also corroborated by the traceability matrices in the trace to requirements and trace to modules.