SE 3XA3: Test Report
Rummy For Dummies

Lab 2 Group 7, Rummy For Dummies
Joy Xiao, xiaoz18
Benson Hall, hallb8
Smita Singh, sings59

April 09, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
April 1, 2021	1.0	Started on the test report
April 9, 2021	1.1	Finished the test report

This document contains the results of our testing based off of the test plan.

1 Functional Requirements Evaluation

1.1 Card Tests

1. FR-C-1: Test accessor for card suit

   Initial State: Card = new Card(Suit.D, 1)

   Input: Card

   Expected Output: Suit.D

   Results: Passed

2. FR-C-2: Test constructor with invalid rank

   Initial State: N/a

   Input: Card = new Card(Suit.D, -1)

   Expected Output: IllegalArgumentException

   Results: Passed

3. FR-C-3: Test accessor for point value of card (I)

   Initial State: Card = new Card(Suit.D, 11)

   Input: Card

   Expected Output: 10

   Results: Passed

4. FR-C-4: Test accessor for point value of card (II)

   Initial State: Card = new Card(Suit.C, 2)

   Input: Card

   Expected Output: 2

   Results: Passed

5. FR-C-5: Test String representation of card (I)

   Initial State: Card = new Card(Suit.H, 11)

   Input: Card

   Expected Output: ”Jh”

   Results: Passed

6. FR-C-6: Test String representation of card (II)

   Initial State: Card = new Card(Suit.C, 2)

   Input: Card

   Expected Output: ”2c”

   Results: Passed

7. FR-C-7: Test Symbol representation of card

   Initial State: Card = new Card(Suit.S, 11)

   Input: Card

   Expected Output: ”J spade symbol”

   Results: Passed

8. FR-C-8: Test getRank method

   Initial State: Card = new Card(Suit.D, 1)

   Input: Card

   Expected Output: 1

   Results: Passed

1.2 Discard Pile Tests

1. FR-DP-1: Test display of discard pile

   Initial State: DiscardPile = [5h, 9d]

   Input: DiscardPile.displayTopCard

   Expected Output: 9h displayed

   Results: Pass

2. FR-DP-2: Test drawing card from discard pile

   Initial State: DiscardPile = [5h, 9d]

   Input: DiscardPile.pop

   Expected Output: DiscardPile = [5h]

   Results: Pass

1.3 Stock Pile Tests

1. FR-SP-1: Test drawing from stock pile with 1 or more cards

   Initial State: StockPile = [As, 5d]

   Input: Stockpile.pop

   Expected Output: Stock pile = [As]

   Results: Pass

1.4 Computer Tests

1. FR-CP-1: Test computer make move method (I)

   Initial state: DiscardPile = [4h, 9c, 5d], Computer Hand = [4d, 6d, Ks, Js, Ad, 2d], StockPile = [Qs, 3d]

   Input: Computer.makeMove

   Expected Output: DiscardPile = [4h, 9c, Ks], Computer Hand = [4d, 6d, Js, Ad, 2d, 5d], StockPile = [Qs, 3d]. Returns False.

   Results: Pass

2. FR-CP-2: Test computer make move method (II)

   Initial state: DiscardPile = [4h, Jc, 10d, Ks], Computer Hand = [4d, 6d, Js, Ad, 2d, 5d], StockPile = [Qs, 3d]

   Input: Computer.makeMove

   Expected Output: DiscardPile = [4h, Jc, 10d, Ks, Js], Computer Hand = [4d, 6d, Ad, 2d, 5d, 3d], StockPile = [Qs]. Returns False.

   Results: Pass

3. FR-CP-3: Test computer make move method (III)

   Initial state: DiscardPile = [4h, Jc, 10d, As], Computer Hand = [4d, 6d, Js, Ad, 2d, 5d], StockPile = [Qs, 3d]

   Input: Computer.makeMove

   Expected Output: DiscardPile = [4h, Jc, 10d, Js], Computer Hand = [4d, 6d, Ad, 2d, 5d, As], StockPile = [Qs, 3d]. Returns False.

   Results: Pass

4. FR-CP-4: Test that computer never discards a meld card

   Initial state: DiscardPile = [4h, Jc, 10d, As], Computer Hand = [4d, 6d, Ad, 2d, Kc, Kd, Ks], StockPile = [Qs, 3d]

   Input: Computer.makeMove

   Expected Output: DiscardPile = [4h, Jc, 10d, 6d], Computer Hand = [4d, Ad, 2d, Kc, Kd, Ks, As], StockPile = [Qs, 3d]. Returns False.

   Results: Pass

5. FR-CP-5: Test that computer never discards a meld card

   Initial state: DiscardPile = [4h, Jd, 10d, As], Computer Hand = [4d, 6d, Ad, 2d, Jc, Qc, Kc], StockPile = [Qs, 3d]

   Input: Computer.makeMove

   Expected Output: DiscardPile = [4h, Jd, 10d, 6d], Computer Hand = [4d, Ad, 2d, Jc, Qc, Kc, As], StockPile = [Qs, 3d]. Returns False.

   Results: Pass

6. FR-CP-6: Test that computer knocks whenever it's hand has deadwood score of 10 or less

   Initial state: Computer Hand = [4d, Ad, 2d, Jc, Qc, Kc, As]

   Input: computer.makeMove

   Expected Output: Returns True.

   Results: Pass

7. FR-CP-7: Test computer move when discard pile has 1 card

   Initial state: DiscardPile = [4h], StockPile = [Qs, 3d], Computer Hand = [4d, 6d, Js, Ad, 2d, 5d]

Input: computer.makeMove

Expected Output: DiscardPile = [Js], StockPile = [Qs, 3d], Computer Hand = [4d, 6d, Ad, 2d, 5d, 4h]. Returns False.

Results: Pass

8. FR-CP-8: Test computer does not draw from empty stock pile

   Initial state: DiscardPile = [4h], StockPile = [], Computer Hand = [4d, 6d, Js, Ad, 2d, 5d]

   Input: computer.makeMove

   Expected Output: DiscardPile = [Js], StockPile = [], Computer Hand = [4d, 6d, Ad, 2d, 5d, 4h]. Returns False.

   Results: Pass

1.5 Hand Tests

1. FR-H-1: Test remove card from hand (I)

   Initial state: hand = [As, 2s, Qs, 3d, 5d]

   Input: ”As”

   Expected Output: hand = [2s, Qs, 3d, 5d]

   Results: Pass

2. FR-H-2: Test remove card from hand (II)

   Initial state: hand = [As, 2s, Qs, 3d, 5d]

   Input: ”Kc”

   Expected Output: hand = [As, 2s, Qs, 3d, 5d]

   Results: Pass

3. FR-H-3: Test remove card from hand (III)

   Initial state: hand = [As, 2s, Qs, 3d, 5d]

   Input: ”AS”

   Expected Output: hand = [2s, Qs, 3d, 5d]

   Results: Pass

4. FR-H-4: Test remove card from hand (IV)

   Initial state: hand = [As, 2s, Qs, 3d, 5d]

   Input: "KS"

   Expected Output: hand = [As, 2s, Qs, 3d, 5d]

   Results: Pass

5. FR-H-5: Test displaying of hand

   Initial state: Player object initialized with a non-empty hand

   Input: Hand

   Expected Output: All cards in player's hand is displayed correctly

   Results: Pass

6. FR-H-6: Test contains method (I)

   Initial state: hand = [As, 2s, Qs, 3d, 5d]

   Input: input = "As"

   Expected Output: Return True

   Results: Pass

7. FR-H-7: Test contains method (II)

   Initial state: hand = [As, 2s, Qs, 3d, 5d]

   Input: input = "Ks"

   Expected Output: Return False

   Result: Pass

8. FR-H-8: Test contains method (III)

   Initial state: hand = [As, 2s, Qs, 3d, 5d]

   Input: input = "AS"

   Expected Output: Return True

   Results: Pass

9. FR-H-9: Test contains method (IV)

   Initial state: hand = [As, 2s, Qs, 3d, 5d]

   Input: input = ”KS”

   Expected Output: Return False

   Results: Pass

1.6 Meld Tests

1. FR-M-1: Test sequence melds with more than 3 cards

   Initial state: hand = Player hand has 4 cards in consecutive rank with same suit eg. 3S, 4S, 5S, 6S

   Input: input = Hand : [3s, 7d, Qh, 4s, Kc, Kd, Ac, Ad, 5s, 6s]

   Expected Output: [[3S, 4S, 5S, 6S]]

   Results: Passed

2. FR-M-2: Test melds with 3 cards of same rank

   Initial state: Hand has 3 cards of same rank

   Input: input = Hand: [3s, 7d, Qh, 4s, Kc, Kd, Ac, Ad, Ah, 6s]

   Expected Output: [[Ah, Ac, Ad]]

   Results: Passed

3. FR-M-3: Test melds with 4 cards of same rank

   Initial state: Player’s hand has 4 cards of same rank

   Input: Hands: [Kd,5d,4d,5c,Jc,Qd,5h,5s,3s,Js]

   Expected Output:[[5h,5s,5c,5d]]

   Results: Passed

4. FR-M-4: Test meld with 2 sequence melds and one group meld

   Initial state: Player’s hand has 2 sequence melds and one group meld

   Input: Hand with Cards: [3s,4s,5s, Kc,Jc,Qc,Ac,Ad,Ah,As]

   Expected Output: [[3s,4s,5s],[Jc,Kc,Qc],[Ah,As,Ac,Ad]]

   Results: Passed

5. FR-M-5: Test meld with 1 sequence and 1 group meld

   Initial state: Player's hand has 1 sequence melds and 1 group meld

   Input: Hand : [10c,Js, Ks, 10d, Qs, 10h, Ac, 4c, 6h, 9h]

   Expected Output:[[Js,Qs,Ks],[10h,10c,10d]]

   Results: Passed

6. FR-M-6: Test Melds with overlapping cards

   Initial state: Player's hand has a meld that can belong to a sequence and group meld

   Input: [Ac, Ad, Ah, As,2s,3s,Ks,Jh, Jd, 7s]

   Expected Output: [[As,2s,3s],[Ah,Ac,Ad]]

   Results: Passed

1.7 Player Tests

1. FR-P-1: Test accessor for player's name

   Initial State: Player p = new Player("P1");

   Input: p

   Expected Output: "P1"

   Results: Passed

2. FR-P-2: Test adding a card to player's hand

   Initial State: p.hand = [Ah, 2s, Qs, 3d, 5d];

   Input: p, new Card(Suit.S, 13)

   Expected Output: p.hand = [Ah, 2s, Qs, 3d, 5d, Ks];

   Results: Passed

3. FR-P-3: Test accessor for player's hand

   Initial State: p.hand = [Ah, 2s, Qs, 3d, 5d];

   Input: p

   Expected Output: [Ah, 2s, Qs, 3d, 5d]

   Results: Passed

4. FR-P-4: Test accessor for player's total score

   Initial State: p.totalScore = 12;

   Input: p

   Expected Output: 12

   Results: Passed

5. FR-P-5: Test accessor for player's deadwood score

   Initial State: p.hand = [Ah, Ac, Ad, As, 2s, 2d, 2c, 3d, Qs, 9h];

   Input: p

   Expected Output: 22

   Results: Passed

6. FR-P-6: Test accessor for player's melds

   Initial State: p.hand = [Ah, Ac, Ad, As, 2s, 2d, 2c, 3d, Qs, 9h];

   Input: p

   Expected Output: [[As, Ah, Ac, Ad], [2s, 2d, 2c]]

   Results: Passed

7. FR-P-7: Test discarding a card that exists in the player's hand

   Initial State: p.hand = [Ah, 2s, Qs, 3d, 5d];

   Input: p, "5d"

   Expected Output: p.hand = [Ah, 2s, Qs, 3d], new Card(Suit.D, 5) returned

   Results: Passed

8. FR-P-8: Test discarding a card that does not exist in the player's hand

   Initial State: p.hand = [Ah, 2s, Qs, 3d, 5d]

   Input: p, "jh"

   Expected Output: IllegalArgumentException()

   Results: Passed

9. FR-P-9: Test adding a score to the total score

   Initial State: p.totalScore = 12

   Input: p, 15

   Expected Output: p.totalScore = 27

   Results: Passed

10. FR-P-10: Test getting deadwood cards from the player's hand

    Initial State: p.hand = [Ah, Ac, Ad, 2s, 9h, 10h, Jd]

    Input: p

    Expected Output: [2s, 9h, 10h, Jd]

    Results: Passed

11. FR-P-11: Test recalculation of deadwood score

    Initial State: p.hand = [Ah, Ac, Ad, 2s, 9h, 10h, Jd], p.deadwoodScore = 0

    Input: p

    Expected Output: p.deadwoodScore = 31

    Results: Passed

12. FR-P-12: Test accessor for player's melds after checking for melds

    Initial State: p.hand = [As, 2s, 3s, 4d, 5d, 6d, 7d, 10h], p.melds = [[As, 2s, 3s]]

    Input: p

    Expected Output: p.melds = [[As, 2s, 3s], [4d, 5d, 6d, 7d]]

    Results: Passed

13. FR-P-13: Test resetting hand

    Initial State: p.hand = [Ah, Ac, Ad, 2s, 9h, 10h, Jd]

    Input: p

    Expected Output: p.hand = []

    Results: Passed

14. FR-P-14: Test resetting deadwood score

    Initial State: p.deadwoodScore = 31

    Input: p

    Expected Output: p.deadwoodScore = 0

    Results: Passed

15. FR-P-15: Test resetting list of melds

    Initial State: p.melds = [[Ah, Ad, Ac]]

    Input: p

    Expected Output: p.melds = []

    Results: Passed

16. FR-P-16: Test resetting total score

    Initial State: p.totalScore = 12

    Input: p

    Expected Output: p.totalScore = 0

    Results: Passed

1.8 UserInputOps Tests

1. FR-UIO-1: Test that user inputting valid inputs for deciding on knocking results in successful termination

   Initial State: N/A

   Input: User input: 'Y'

   Expected Output: 'y'

   Results: Passed

2. FR-UIO-2: Test that user inputting invalid inputs for deciding on knocking is handled properly

   Initial State: N/A

   Input: User inputs: 'h', 'e', 'l', 'l', 'o', "othello", "No"

   Expected Output: 'n'

   Results: Passed

3. FR-UIO-3: Test that user inputting valid inputs for deciding on playing a new game of Gin-Rummy results in successful termination

   Initial State: N/A

   Input: User input: ’Y’

   Output: ’y’

   Results: Passed

4. FR-UIO-4: Test that the user inputting invalid inputs for deciding on playing a new game of Gin-Rummy is handled properly

   Initial State: N/A

   Input: User inputs: ”To”, ”be”, ”or”, ”not”, ”to”, ”Be”, ”yEs”

   Expected Output: ’n’ - note that ’not’ begins with n

   Results: Passed

5. FR-UIO-5: Test/Simulate user making a valid decision in a single round

   Initial State: N/A

   Input: User input: 3

   Expected Output: 3

   Results: Passed

6. FR-UIO-6: Test that when the user making a invalid decision in a single round, it is handled appropriately

   Initial State: N/A

   Input: User inputs: -1, 45, ”hello”, ’y’, ”tomorrow and tomorrow”, 2

   Expected Output: 2

   Results: Passed

7. FR-UIO-7: Test username is received properly

   Initisal State: N/A

   Input: User input: ”AC”

   Expected Output: ”AC”

   Results: Passed

1.9 GameOps Tests

1. FR-GO-1: Test that score is calculated correctly and given to the right player (I)

   Initial State: (Player) p1.deadwoodScore = 10, p1.totalScore = 0, (Player) p2.deadwoodScore = 15, p2.totalScore = 0

   Input: p1, p2

   Expected Output: p1.totalScore = 5, p2.totalScore = 0

   Results: Passed

2. FR-GO-2: Test that score is calculated correctly and given to the right player (II)

   Initial State: p1.deadwoodScore = 20, p1.totalScore = 0, p2.deadwoodScore= 9, p2.totalScore = 0

   Input: p1, p2

   Expected Output: p1.totalScore = 0, p2.totalScore = 11

   Results: Passed

3. FR-GO-3: Test that going Gin works correctly

   Initial State: p1.deadwoodScore = 0, p1.totalScore = 0, p2.deadwoodScore= 15, p2.totalScore = 0

   Input: p1, p2

   Expected Output: p1.totalScore = 35, p2.totalScore = 0

   Results: Passed

4. FR-GO-4: Test that stock pile is created correctly

   Initial State: N/A

   Input: N/A

   Expected Output: Stock Pile with 52 unique cards (A to K, all suits)

   Results: Passed

5. FR-GO-5: Test that discard pile is created correctly

   Initial State: N/A

Input: N/A

Expected Output: Empty stack/discard pile

Results: Passed

6. FR-GO-6: Test that opening distribution of cards is done correctly

   Initial State: StockPile sp = createStockPile();

   Input: N/A

   Expected Output: Player and cpu hands of size 10, discard pile of size 1, stockpile contains the rest of the cards

   Results: Passed

7. FR-GO-7: Test that opening distribution of cards will only be done with a 52-card stock pile

   Initial State: Stock pile of size 28

   Input: N/A

   Expected Output: Rejected status

   Results: Passed. IllegalArgumentException raised.

8. FR-GO-8: Test interfacing playAgain method

   Initial State: N/A

   Input: User inputs: 'A', 'B', 'n'

   Expected Output: 'n'

   Results: Passed

9. FR-GO-9: Test process decision method

   Initial State: New game has started

   Input: User's decision based on possible moves

   Expected Output: Depends on move made - 1 will draw a card from the stockpile, 2 will draw a card from the discard pile, 3 will show melds, 4 will show deadwood score and prompt user if they wish to knock. If user knocks, return true, else return false

   Results: All combinations of inputs have passed

10. FR-GO-10: Test reset for a new deal

    Initial State: p.hand = [Ah, Ac, Ad, 6s], p.deadwoodScore = 6, p.melds= [Ah, Ac, Ad]

    Input: N/A

    Expected Output: p.hand = [], p.melds = [], p.deadwoodScore = 0

    Results: Passed

11. FR-GO-11: Test interfacing username method

    Initial State: N/A

    Input: "AC"

    Expected Output: "AC"

    Results: Passed

12. FR-GO-12: Test decision making when stock pile is empty

    Initial State: (StockPile) sp = new StockPile()

    Input: User inputs 1 when prompted

    Output: User should not be allowed to draw from the stock pile, and is prompted to make a new decision

    Results: Passed. User is forced to draw from the discard pile to carry on.

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel

NFR-LF-1: Test game appearance

Initial state: Cards dealt, player's hand and discard pile displayed

Input: N/A

Output: Cards displayed have a simplistic look

Results: Passed. The cards visually look like standard, real-life playing cards, hence it is intuitive for the user.

2.2 Usability

NFR-UH-1: Test game usability

Initial state: Cards dealt, player's hand and discard pile displayed

Input: N/A

Expected Output: Easy to set up and play the game

Results: Passed. Batch file was created and the user is no longer required to compile and run files on the command line to play game.

2.3 Performance

NFR-P-1: Test performance speed

Initial state: Start a new game

Input: User interactions with system

Expected Output: System should respond within 0.5 seconds of user input, according to the non-functional requirements

Results: Passed. The game responds immediately to player input. No noticeable delay.

NFR-P-2: Test game rules requirements

Initial state: N/A

Input: N/A

Expected Output: Legal moves (according to game rules) can be done without errors. User is prompted for another input if they are making an illegal move.

Results: Passed. No illegal moves can successfully be made.

2.4 Operational and Environmental

NFR-OE-1: Test operational requirements

Initial state: N/A

Input: N/A

Expected Output: Game can be played on various operating systems

Result: Passed. However, the batch file only works on Windows. To run the code on MacOS, it is necessary to either compile source code, or to run the associated JAR file through a command.

2.5 Maintainability and Support

NFR-MS-1: Test maintainability requirements

Initial state: N/A

Input: N/A

Expected Output: Well-documented source code, game rules and description on how to run the game for users

Results: Passed. All the source code for the game has Doxygen and JavaDoc comments. ReadME files have instructions on running the game.

2.6 Cultural

NFR-C-1: Test cultural requirements

Initial state: N/A

Input: N/A

Expected Output: No offensive images or text in source code or displayed to users.

Results: Passed. Manually tested that there are no offensive images or text.

NFR-C-2: Test cultural requirements

Initial state: N/A

Input: N/A

Expected Output: Game console output is all in English

Results: Passed. Checked that everything is written in English in the implementation, documentation, and anything visible to the user.

2.7 Legal

NFR-L-1: Test legal requirements

Initial state: N/A

Input: N/A

Expected Output: Project adheres to copyright properties

Results: Passed. Checked the MIT License requirements and that the project adhered to all the requirements.

3 Comparison to Existing Implementation

These tests compare the program to the Go implementation.

- NFR-LF-1 Test Game Appearance
- NFR-MS-1 Test Maintainability Requirements

4 Unit Testing

Unit tests were written with JUnit framework.

5 Changes Due to Testing

Through the use of system testing by playing the game, errors in the code were discovered. These faults were fixed, and additional tests were created to ensure that the faults were fixed.

An example of a change was preventing either participant in the game from drawing from the stock pile if the stock pile was empty. Tests FR-CP-8 and FR-GO-12 were created to assert that neither player draws from an empty stock pile.

Another example test that was added after system testing was FR-CP-7. An EmptyStackException was being thrown, as the algorithm for the computer attempted to peek on the top of the stack after the one card on it had been removed. This was patched and a new test was added to confirm the fix.

During system testing, it was apparent that testers were prone to making mistakes in typing user inputs, which would often lead to the program experiencing formatting exceptions and crashing. To address this issue and improve robustness, several different methods were implemented to address invalid user inputs, such as implementing guards, or taking advantage of exceptions raised by the program to prompt the user again for a valid input.

An IllegalArgumentException added to the constructor of the Card class to handle situations when an invalid rank was used to instantiate a Card object.

Changes were made to the SortByRank class to ensure that melds will always be displayed in a certain order.

6 Automated Testing

Automated testing was used for this project using JUnit. Most of the tests performed at the unit level were automated tests as well as some integration tests which combined some components together.

7 Trace to Requirements

Table 2: Traceability Matrix: Functional Requirement

Test IDs	**Requirement IDs**							
	BE1	**BE2**	**BE3**	**BE4**	**BE5**	**BE6**	**BE7**	**BE8**
FR-C-1	X	X	X	X	X	X		
FR-C-2	X	X	X	X	X	X		
FR-C-3	X	X	X	X	X	X		
FR-C-4	X	X	X	X	X	X		
FR-C-5	X	X	X	X	X	X		
FR-C-6	X	X	X	X	X	X		
FR-C-7	X	X	X	X	X	X		
FR-C-8	X	X	X	X	X	X		
FR-DP-1			X			X		
FR-DP-2			X			X		
FR-SP-1		X						
FR-CP-1		X						
FR-CP-2		X						
FR-CP-3		X						
FR-CP-4		X						
FR-CP-5		X						
FR-CP-6		X						
FR-CP-7		X						
FR-CP-8		X						
FR-H-1						X		
FR-H-2						X		
FR-H-3						X		
FR-H-4						X		
FR-H-5	X	X	X		X	X		
FR-H-6						X		
FR-H-7						X		
FR-H-8	X					X		
FR-H-9						X		
FR-P-1	X							
FR-P-2		X	X					
FR-P-3		X	X	X	X	X	X	

Test IDs	Requirement IDs							
	BE1	**BE2**	**BE3**	**BE4**	**BE5**	**BE6**	**BE7**	**BE8**
FR-P-4					X			
FR-P-5							X	
FR-P-6				X				
FR-P-7						X		
FR-P-8						X		
FR-P-9							X	
FR-P-10					X		X	
FR-P-11					X		X	
FR-P-12				X			X	
FR-P-13	X							
FR-P-14	X				X			
FR-P-15				X				
FR-P-16	X				X			
FR-M-1				X			X	
FR-M-2				X			X	
FR-M-3				X			X	
FR-M-4				X			X	
FR-M-5				X			X	
FR-M-6				X			X	
FR-UIO-1							X	
FR-UIO-2							X	
FR-UIO-3	X							
FR-UIO-4	X							
FR-UIO-5		X	X	X	X		X	
FR-UIO-6		X	X	X	X		X	
FR-UIO-7	X							
FR-GO-1							X	
FR-GO-2							X	
FR-GO-3							X	
FR-GO-4	X							X
FR-GO-5	X							X
FR-GO-6	X							
FR-GO-7	X							
FR-GO-8								X
FR-GO-9		X	X	X	X		X	
FR-GO-10	X							

Test IDs	**Requirement IDs**							
	BE1	**BE2**	**BE3**	**BE4**	**BE5**	**BE6**	**BE7**	**BE8**
FR-GO-11	X							
FR-GO-12		X						

Table 3: Traceability Matrix: Non-Functional Requirement

Test IDs	**Requirement IDs**								
	LF1	**UH1**	**P1**	**P2**	**OE1**	**MS1**	**C1**	**C2**	**L1**
NFR-LF-1	X								
NFR-UH-1		X							
NFR-P-1			X						
NFR-P-2				X					
NFR-OE-1					X				
NFR-MS-1						X			
NFR-C-1							X		
NFR-C-2								X	
NFR-L-1									X

8 Trace to Modules

The module IDs are from the Rummy For Dummies Module Guide.

Table 4: Traceability Matrix: Modules

Test IDs	Module IDs									
	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**
FR-C-1							X			
FR-C-2							X			
FR-C-3							X			
FR-C-4							X			
FR-C-5							X			
FR-C-6							X			
FR-C-7							X			
FR-C-8							X			
FR-DP-1						X				
FR-DP-2						X				
FR-SP-1					X					
FR-CP-1				X						
FR-CP-2				X						
FR-CP-3				X						
FR-CP-4				X						
FR-CP-5				X						
FR-CP-6				X						
FR-CP-7				X						
FR-CP-8				X						
FR-H-1								X		
FR-H-2								X		
FR-H-3								X		
FR-H-4								X		
FR-H-5								X		
FR-H-6								X		
FR-H-7								X		
FR-H-8								X		
FR-H-9								X		
FR-P-1										X
FR-P-2										X

Test IDs	Module IDs									
	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**
FR-P-3										X
FR-P-4										X
FR-P-5										X
FR-P-6										X
FR-P-7										X
FR-P-8										X
FR-P-9										X
FR-P-10										X
FR-P-11										X
FR-P-12										X
FR-P-13										X
FR-P-14										X
FR-P-15										X
FR-P-16										X
FR-M-1									X	
FR-M-2									X	
FR-M-3									X	
FR-M-4									X	
FR-M-5									X	
FR-M-6									X	
FR-UIO-1			X							
FR-UIO-2			X							
FR-UIO-3			X							
FR-UIO-4			X							
FR-UIO-5			X							
FR-UIO-6			X							
FR-UIO-7			X							
FR-GO-1		X								
FR-GO-2		X								
FR-GO-3		X								
FR-GO-4		X								
FR-GO-5		X								
FR-GO-6		X								

Test IDs	**Module IDs**									
	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**
FR-GO-7		X								
FR-GO-8		X								
FR-GO-9		X								
FR-GO-10		X								
FR-GO-11		X								
FR-GO-12		X								
NFR-LF-1		X				X		X		
NFR-UH-1		X	X			X		X		
NFR-P-1		X		X					X	
NFR-P-2		X	X							
NFR-OE-1	X									
NFR-MS-1		X	X	X	X	X	X	X	X	X
NFR-C-1		X				X		X		
NFR-C-2		X				X		X		
NFR-L-1		X	X	X	X	X	X	X	X	X

9 Code Coverage Metrics

The tests have produced a 90% code coverage. This is based on the overlapping coverage of the modules during testing. To see this overlapping coverage, refer to the Trace to Modules table.