Test Report

December 9, 2015

SE 2XA3

Hui Chen	chenh43
Nareshkumar Maheshkumar	maheshn
Sam Hamel	hamels2

Group E

Revision History

Table 1: Revision History Table

Developer	Date	Change	Revision
Hui Chen	Dec 8 2015	Revised section 5 in the Test Report Document	10
Nareshkumar Maheshkumar	Dec 8 2015	Revised 2, 3, 4.2.4, 4.2.5, 8, 9, 15 and 16 in the Design Document	9
Hui Chen	Dec 8 2015	Revised section 2.1, 2.2.2, 2.2.3 in the Test Plan Document	8
Hui Chen	Dec 8 2015	Revised Non-functional Requirements in the Requirements Document	7
Nareshkumar Maheshkumar	Dec 4 2015	Revised 3.4, 4.8 and 5.8 in the Requirements Document	6
Hui Chen	Dec 2 2015	Revised: Fixed the numbering in 3.3 and 3.4 in the Requirements Document	5
Sam Hamel	Dec 2 2015	Revised: Number the Functional Requirements in the Requirements Document	4
Hui Chen	Nov 28 2015	Added Method of Testing/Update format	3
Nareshkumar Maheshkumar Sam Hamel	Nov 28 2015	Added Introduction and Automated Tests	2
Sam Hamel	Nov 28 2015	Add matrix coverage and Traceability Table	1
Hui Chen	Nov 28	Added Nonfunctional Quality Quality Test and Summary	0
Hui Chen	Nov 26	Started Test Report document	0

Contents

List of Tables

1 Introductions

1.1 Objective

The main functions that are being tested are Player moves, Player betting, management of chips, management of pot, card distribution and hand ranking; these are all tested using Junit. Also included in the test report is the results of manually testing the GUI and the results of a usability test, which was surmised through surveying five people. The testing produced no errors and the usability test result met the acceptable criteria stated in the requirements document.

1.2 Method of Testing

Testing will mostly be done through automated unit tests using JUnit, with some manual testing, and some whitebox testing in order to validate actual outputs with the expected outputs in order to verify quality, robustness, and functionality.
There will be many test cases including extreme and abnormal cases done through automated testing because the software should produce predictable and consistent results for most cases. Tests that cannot be tested using automated tests such as the user interface will be done through manual testing as automated tests is not feasible for the user interface. The manual tests will provide us with insight from the user on how it can be improved and also to test the interface to see if everything looks and feels the way it should.
The software can/will be installed on any system that runs Java all tests performed to date are done in Windows. The tests were performed through the Eclipse IDE and the operation environment will be the same in terms of the operating systems, but it will not be run through Eclipse. All automated tests are completed with the use of JUnit, which performs unit tests for individual method(s) within modules.

1.3 Coverage Matrix

Table 2: Coverage Matrix

Test 1: Game Start Speed		
Requirements	Results	Automation Coverage
3.4 1)	5.1: pass	None
3.4 2)	5.2: pass	None
3.4 3)	4: pass	Pass
3.4 4)	4.1-4.6, 4.12: pass	pass
3.4 5)	4: pass	pass
3.4 7)	4.1-4.6: pass	pass
3.4 8)	4.11: pass	pass

2 Nonfunctional Quality Tests

2.1 Usability Tests

Usability Tests are performed by recruiting 3 complete beginners with no knowledge of the game as well as 3 people who are already familiar with the game. The participants were given a series of tasks to perform. The amount of time required for them to complete the task as well as comments to the task are recorded. Participants will not be named to protect their identity. Further discussion of the results can be found in section 7.1.

Table 3: Usability Test Results Table

Task 1 - Play through the game until either the player or AI wins without help.		
Participant	Task Completion (sec)	Comments
Beginner 1	79	None
Beginner 2	20	I won without even knowing how to play
Beginner 3	53	I don't understand it.
Veteran 1	278	works
Veteran 2	38	None
Verteran 3	102	None
Task 2 - Help Page		
Beginner 1	28	None
Beginner 2	22	looks too plain
Beginner 3	30	None
Veteran 1	15	Needs more in depth explanations
Veteran 2	19	Simple and concise
Verteran 3	25	None
Task 3 - Rule Page		
Beginner 1	62	Decent explanation, wish it had some strategy guide as well.
Beginner 2	55	looks simple enough
Beginner 3	94	None
Veteran 1	13	It's missing something
Veteran 2	29	pretty basic
Verteran 3	10	None
Task 4 - Play through the game until either the player or AI wins after reading help and rules.		
Beginner 1	148	Makes a little more sense now
Beginner 2	54	I think it's a little easier now that I know what each button does and what the win conditions are, but that still doesn't change the fact that I don't know how to play this game
Beginner 3	495	None
Veteran 1	18	None
Veteran 2	91	None
Verteran 3	58	None

2.2 Performance Tests

Performance tests are to be performed using JUnit and a StopWatch library to measure the time required to execute certain actions. Each JUnit test will run 10 times and if the average is within acceptable range, then the test passes. It is important to note that the performance may vary depending on the user's hardware, so for consistency, the tests will be conducted with the same machine. Further analysis can be found in section 7.2.

Table 4: Performance Test Results Table

Test 1: Game Start Speed	
Test #	Results (ms)
1	777
2	62
3	76
4	77
5	49
6	57
7	48
8	67
9	51
10	56
avg:	132.0

2.3 Robustness Tests

To test for robustness, JUnit will be used to manually trigger events in which an error is produced which should be caught and displayed to the user. The results of such tests can be found under section 5.

3 Automated Unit Tests

Unless otherwise stated, all results here were generated using Automatic testing using junit. These tests included several inputs (both normal and boundary), as such only the types of inputs/outputs are stated here and only one specific input and specific output is shown as an example for each test.

3.1 Normal Betting Tests - Player, Pot, Game, AIOpponent Module

3.1.1 Validation Tests

Input: player chooses bet option and chooses bet amount (x = 200) Initial State: Game started, player's turn Output: Player.chips =2000, Player.bet = 200, Player.chips = 1800, Pot = 200

3.2 Verification Tests

Expected Output: 200; player starts with 1000 chips loses the amount of chips entered for the bet, x, which equals the amount in the pot All results are correct and expected outputs = actual outputs

3.3 Boundary betting Tests: Negatives - Player, Pot, Game, AIOpponent Module

3.3.1 Validation Tests

Input: player chooses bet option and chooses bet amount (x ¡ 0): x = -1 Initial State: Game started, player's turn Output: Window saying no negative inputs, state stays the same

3.3.2 Verification Tests

Expected Output: Window saying no negative inputs, state stays the same All results are correct and expected outputs = actual outputs

3.4 Boundary betting Tests: Over bet - Player, Pot, Game, AIOpponent Module

3.4.1 Validation Tests

Input: player chooses bet option and chooses bet amount (x ¿ Player.chips): x = 400, Player.chips = 100 Initial State: Game started, player's turn Output: Message window stating they bet too much, state stays the same

3.4.2 Verification Tests

Expected Output: Message window stating they bet too much, state stays the same All results are correct and expected outputs = actual outputs

3.5 Boundary Test empty pot distribution

3.5.1 Validation Tests

Input: Pot = 0, Player, Player1.Hand = null Initial State: Game in progress, Hands Evaluated Output: Error, Division by zero

3.5.2 Verification Tests

Expected Output: Error, Division by zero All results are correct and expected outputs = actual outputs

3.6 Boundary Test Empty Hand

3.6.1 Validation Tests

Input: pot, Players Initial State: Game in progress, Hands Evaluated Output: Error, Null pointer exception

3.6.2 Verification Tests

Expected Output: Error, Null pointer exception All results are correct and expected outputs = actual outputs

3.7 Folding Tests Player - Player, Pot, Hand, Game, AIOpponent Modules

3.7.1 Validation Tests

Input: Various combinations of betting/checking until player folds Initial State: Game in progress, players turn, Player1.chips = 400, AIPlayer.chips = 400, Player1's bet = 200, AIPlayer's bet = 200, Player1 checks, AIPlayer folds Output: Player.chips = AIPlayer.chips + pot; Player/AIplayer lose chips each time they call/bet/raise; Player1.chips = 600, AIPlayer.chips = 200

3.7.2 Verification Tests

Expected Output: Player.chips = AIPlayer.chips + pot; Player/AIplayer lose chips each time they call/bet/raise; Player1.chips = 600, AIPlayer.chips = 200 All results are correct and expected outputs = actual outputs

3.8 Round Evaluation - Player, Pot, Hand, Game, AIOpponent Modules

3.8.1 Validation Tests

Input: Various combinations of betting/checking and card combinations/Hands Player1.chips = 400, AIPlayer.chips = 400 Player1.bet = 200, AIPlayer.bet = 200 Player1.Hand = 10D,10C,10S,10H,2D AIPlayer.Hand = 9D,5C, 3S,8H,2D Initial State: Game in progress, Round evaluated; Output: Player with better hand wins, gains pot; Player1.chips = 600, AIPlayer.chips = 200

3.8.2 Verification Tests

Expected Output: Player with better hand wins, gains pot; Player1.chips = 600, AIPlayer.chips = 200

All results are correct and expected outputs = actual outputs

3.9 Round Evaluation - Tie Player, Pot, Hand, Game, AIOpponent Modules

3.9.1 Validation Tests

Input: Various combinations of betting/checking and card combinations where Hands are equal; Player1.chips = 400, AIPlayer.chips = 400 Player1.bet = 200, AIPlayer.bet = 200 Player1.Hand = 10D,2C,3S, 5H,6D AIPlayer.Hand = 10S,5C, 3S,8H,2D Initial State: Game in progress, Round evaluated Output: Players split pot, each player gets half the pot if even pot, if odd one chip is left in pot; Player1.chips = 400, AIPlayer.chips = 400, Pot = 0

3.9.2 Verification Tests

Expected Output: Output: Players split pot, each player gets half the pot if even pot, if odd one chip is left in pot; Player1.chips = 400, AIPlayer.chips = 400, Pot = 0 All results are correct and expected outputs = actual outputs

3.10 Player bets more chips than opponent has - Player, Pot, Hand, Game AIOpponent Modules

3.10.1 Validation Test 1

Input: Player betting amount ¿ Opponent.chips, Player.Hand ¡ Opponent.Hand, Player1.Chips = 400, Player1.bet = 200, AIPlayer.chips = 100, AIPlayer.bet = 100 Player1.Hand = 10D,2C,3S, 5H,6D AIPlayer.Hand = 11S,5C, 3S,8H,2D

Initial state: Game in progress, Round evaluated Output: Opponent gets twice the chips they bet, Player gets rest; Player1.Chips = 300, AIPlayer.chips = 200, Pot = 0

3.10.2 Validation Test 2

Input: Player betting amount ¿ Opponent.chips Player.Hand ¿ Opponent.Hand Initial state: Game in progress, Round evaluated, Player1.Chips = 400, Player1.bet = 200, AIPlayer.chips = 100, AIPlayer.bet = 100 Player1.Hand = 10D,2C,3S, 5H,6D AIPlayer.Hand = 9S,5C, 3S,8H,2D Output: Player.chips = Player.chips + pot, Player.chips = 500

3.10.3 Verification Tests

Validation 1 Expected Output: Player1.Chips = 300, AIPlayer.chips = 200, Pot = 0 Validation 2 Expected Output: Player.chips = 500 All results are correct and expected outputs = actual outputs

3.11 Game End - Game Module, Pot, Hand, Player, AIOpponent

3.11.1 Validation tests

Input: Player bets all chips; Player.hand ¡ Opponent.Hand Initial State: Game in progress, Round evaluated Output: Window saying opponent wins; Game ends

3.11.2 Verification tests

Expected Output: Window saying opponent wins; Game ends All results are correct and expected outputs = actual outputs

3.12 Deck/Card Distribution

3.12.1 Validation tests 1

Input: N/A Initial State: Game in Progress new round started Output: Each player gets two cards, deck is shuffled; Player1.Hand = 11S,12D AIPlayer.Hand = 4D,5S

3.12.2 Validation tests 2

Input: N/A Initial State: Game in Progress both players bet Output: three unique cards come on community board; Player1.Hand = 11S,12D, 3D,3S,4H, AIPlayer.Hand = 4D,5S,3D,3S,4H

3.12.3 Validation tests 3

Input: N/A Initial State: Game in Progress both players bet/call/check Output: one unique card comes on community board; Player1.Hand = 11S,12D, 3D,3S,4H,7S, AIPlayer.Hand = 4D,5S,3D,3S,4H,7S

3.12.4 Verification tests

4.12.1 Expected Output: All cards are unique, each player has two cards deck is shuffled at start of round; Player1.Hand = 11S,12D AIPlayer.Hand = 4D,5S 4.12.2 Expected Output: Three unique community cards are placed on board; Player1.Hand = 11S,12D, 3D,3S,4H AIPlayer.Hand = 4D,5S,3D,3S,4H 4.12.3 Each card given is unique Player1.Hand = 11S,12D, 3D,3S,4H,7S AIPlayer.Hand = 4D,5S,3D,3S,4H,7S

All results are correct and expected outputs = actual outputs

3.13 Class Methods

Each Classs methods were also automatically tested using Junit for expected values. All values met the expected criteria and the Junit tests came back error free.

4 Manual Tests

4.1 Rules and App Info Tests - TexasHoldem Module and AboutInfo Module

4.1.1 Validation tests 1

Input: User presses button for rules game menu Initial State: Game start, Game in progress or Game Output: Rules popup window appears

4.1.2 Validation tests 2

Input: User presses button for App info Initial State: Game start, Game in progress or Game Output: App info popup window appears

4.1.3 Verification Tests

Expected Output: Rules and App Info popup windows show up when pressed on the game menu All results are correct and Expected Output = Actual Output

4.2 GUI Tests - TexasHoldem Module

4.2.1 Validation Tests GUI Button Tests 1

Input: Buttons pressed Initial State: Game in progress Output: Buttons respond and perform functions

4.2.2 Validation Tests GUI Button Tests 2

Input: Buttons pressed Initial State: Game in Ended Output: Buttons do not respond

4.2.3 Validation Tests Window Close

Input: Close window is pressed/ program is quit Initial State(s): All states Output: Window closes

4.2.4 Verification tests

All outputs are equal to the expected outputs

5 Changes Due to Testing

The tests reported here are only for revision 1 of the software, each previous version of the software after revision zero also had these same tests applied to them. The software was only deemed complete and up to the standards set out by the requirements document when all tests returned expected results. Therefore each test that produced results that were not deemed correct changed the software until it was fixed. Additional rules are added to better capture the overall core gameplay. The UI was slightly changed to display more effective information. New functions such as flipping the dealer's cards at the end of the round was added.

6 Test Summary

6.1 Usability Tests

Six participants were timed and observed while they perform tasks given by us within the program. We found that the results were well received, with minor complaints about the user interface.

For the game playing portion, the participants felt that the program was easy to navigate and the elements of the game were not hard to find on the user interface. The gameplay was smooth with minor issues that we have later resolved.

The help and rule page was met with little difficulty, the participants were easily able to pull up either pages and find the basic rules they needed to play the game. Some of the participants have found that the game or the pages are too simple. While we agree that it may be simple, but the idea is to not overwhelm the player with an absurd amount of information to digest, especially not with the beginners. It is true that Texas HoldEm is a complex game, we feel that if a player is armed with the basic knowledge and continuously practice by playing the game, they can understand the game and slowly but eventually come up with strategies on their own. Of course, for more advanced tactics, they would have to do their research but this game is not about having the best and smartest opponent to face but more of something that can be used to ease the transition between beginner and a seasoned player.

The participants were overall satisfied with the current state of the game and have made suggestions toward the improvement of the user interface. Their suggestion will be taken into consideration however there are no plans to implement any changes to the user interface at the present time.

6.2 Performance Tests

The performance tests were conducted in order to determine whether the user can the program in a reasonable time frame and also to see if the program can output the required actions in reasonable time. Our tests have shown that the software passes in these aspects and we have ideas on how to optimize the performance further however there is no plan to implement the ideas at the present time as the current system have exceeded the expected results.

6.3 Robustness Tests

The tests were conducted in JUnit where all possible and foreseeable errors were manually triggered and the results recorded. We found that we were able to catch and output all errors that can occur and no game actions were performed when errors occurred, therefore keeping the system error free. During manual testing, no errors were found that are not already included in our tests.

7 Traceability to Requirements/Modules

Table 5: Traceability Table

Test 1: Game Start Speed			
Module	Requirements	Testing Cases	Status
M9	3.4 1)	5.1	pass
M1	3.4 2)	5.2	pass
M3	3.4 3)	2.1-2.4	pass
M2,M3	3.4 4)	4	pass
M6,M7	3.4 5)	4	pass
M5,M3	3.4 6)	4	pass
M2	3.4 8)	4.1-4.5	pass
M1	3.4 8)	4.11	pass
n/a	4.1	3.1	pass
n/a	4.2	3.2	pass
n/a	4.3	n/a	n/a
n/a	4.4	1.2	pass
n/a	4.5	1.2	pass
n/a	4.6	n/a	n/a
n/a	4.7	n/a	n/a
n/a	4.8	n/a	n/a