Test Plan

November 28, 2015

SE 2XA3

Hui Chen	chenh43
Nareshkumar Maheshkumar	maheshn
Sam Hamel	hamels2

Group E

Contents

List of Tables

Table 1: Revision History Table

Revision #	Date	Team Member	Description of Change
0	October 20	Hui Chen	Started Test Plan document
0	October 21	Sam Hamel Hui Chen	Test Plan Revision 0 finished
0	October 22	Nareshkumar Maheshkumar	Proofread
1	November 28	Hui chen	Made changes to section 2.3 Usability and Learnability Tests

1 General Information

1.1 Purpose

The purpose of this testing plan is to build developer confidence for the correctness of the program and to receive qualitative feedback on the user interface. The plan will address the elements and items related to the production of the TexasHoldEm game. The objective is to detail the tests responsible for each task required of the game, to define the tools and methods needed to conduct the tests and to outline all tasks and its schedule.

1.2 Scope

The scope of the testing plan is to test functional and nonfunctional requirements of the program, and the testing of the GUI for the program. The test plan will not verify correctness and will not be exhaustive in nature, rather it will rely on testing normal and boundary cases of the program for a multitude of different situations in order to build confidence that all errors have been accounted for and dealt with appropriately.

1.3 Constraints

The program cannot be verified through exhaustive testing due to this being infeasible in the timeframe of completing the project. Verification of correctness through mathimatical forumalas or other means is also infeasible due to lack of experience of verification methods by the developers. Therefore there is no guaranteeing that the program will be error free.

1.4 Acronyms, Abbrebiations, and Symbols

2 Plan

2.1 Structural System

Recovery: Recovery testing will be manually and dynamically tested; the game will be run and will be manually crashed at random times, and when re-opened, the system will be manually checked by developers to see if it behaved as the requirements documents specified for recovery scenarios.

Table 2: Acronym/Abbreviation Table

Acronym/Abbreviation	Meaning	Description
GUI	Graphical User Interface	The GUI allows users to interact with the application through graphical icons
SRS	Software Requirements Specification	Document reporting objectives, intended features, risks and constraints of the product
Junit	N/A	A unit testing framework for Java applications
PoC	Proof of Concept	Presentation with a purpose to demonstrate that the project is feasible

Test case 1: save file doesnt exist, program process is closed through system OS Expected results: When program is re-opened the program is in its initial state.

Test case 2: save file exists, program process is closed through system OS Expected results: When program is re-opened the program is in its saved state.

Test case 3: save file doesnt exist, Computer is shut down while program is running Expected results: When program is re-opened the program is in its initial state.

Test case 4: save file exists, Computer is shut down while program is running Expected results: When program is re-opened the program is in its saved state.

2.2 Functional

2.2.1 Features not to be Tested

- no tests will be done for the cultural and political requirements stated in the SRS
- no tests will be done for the legal requirements in the SRS

- no tests will be done for the help page

2.2.2 Features to be Tested in PoC

All tests will be conducted through automatic testing, more specifically, unit testing with the use of Junit. These tests will be necessary to ensure the overall functionality of the product.

Management Of Player Bets, Chips And Pot

Test case 1: Starting a game and assigning chips to players
Input: player chooses bet option and chooses a bet amount
Initial State: Game started, players turn
Expected Output: player starts with 2000 chips and the amount entered for bet is added to the pot

Test case 2: Players have made their bet and one player folds
Input: N/A
Initial State: both player chose their move, one player folds
Expected Output: the player that didnt fold gets the pot added to his chips

Test case 3: Both players' hands are evaluated to be equal
Input: N/A
Initial State: both players have made their moves and the hands are evaluated
Expected Output: each player gets half of the pot if the pot amount is even, (pot-1)/2 if the amount is odd, and 1 is added to the next pot

Test case 4: player loses round
Input: Player1 hand, Player2 hand, pot where Player2 hand strength ¿ Player1 hand strength
Initial State: round over, players put down cards
Expected Output: Player2 chips increase by amount of pot, pot empties

Test case 5: player wins round
Input: Player1 hand, Player2 hand, pot where Player2 hand strength ¡ Player1 hand strength
Initial State: round over, players put down cards

Expected Output: Player1 chips increase by amount of pot, pot empties

Test case 6: Player1 bets more chips than Player2 has, Player2 wins
Input: Player1 bet, Player2 bet
Initial state: Player1 bet, Player2 calls
Expected Output: Player2 gains twice the amount of chips they bet, Player1 gains the rest of the chips

Test case 7: Game end
Input: a player chooses any option and uses up all their chips
Initial State: game in progress
Expected Output: At the end of the turn, the game should end and display the winner. At this point, the user can choose to play again.

Distribution and Handling of Cards

Test case 1: Distribute community and hand cards
Input: N/A
Initial State: Game start, shuffle deck
Expected Output: cards are shuffled, 2 cards are dealt to the player and 3 cards on community

Test case 2: Deal cards after both players choose an action
Input: N/A
Initial State: game in progress, players made their moves
Expected Output: The game should distribute one more card to the community hand

Test case 3: One player chooses to go all in
Input: one player choose all in
Initial State: game in progress
Expected Output: The other player will only have 2 options, call or fold. If the action is call, all community cards are dealt and both players hands are evaluated.

Player Choices

Test case 1: check/call

Input: player chooses check/call on their turn
Initial State: game in progress
Expected Output: The game should automatically deduct the same number of chips from the player as the other players bet and add the amount to the pot, and the game continues. The system should be able to keep track of the other players bet in order for the automatic deduction to work.

Test case 2: fold
Input: a player chooses to fold
Initial State: game in progress
Expected Output: the other player adds the pot to their chips and a new round should start.

Test case 3: raise
Input: Player raises by a certain amount thats less than the players chips
Initial State: game in progress
Expected Output: The number of chips raised will be added to the pot and deducted from the player

Test case 4: raise error
Input: Player raises by an amount thats greater than the players chips
Initial State: game in progress
Expected Output: The game should prompt the user with an error message and return to a state before the message appeared.

Save and Load System

Test case 1: Saving a game
Input: user requests save
Initial State: Game in progress
Expected Output: The game is saved onto a local file with the relevant information regarding chips, pot, game state, turn, etc.
Test case 1: Loading a game
Input: user requests load
Initial State: any state
Expected Output: The game is loaded with all relevant information required for the game to resume
Test case 1: Loading error

Input: user requests load but no file is found
Initial State: any state
Expected Output: The game should display an error

The GUI will be tested through manual testing. Individual elements such as buttons and images are tested for correctness, reliability and speed. Making sure that the GUI elements update promptly and accurately will help satisfy the speed and appearance requirements. The appearance requirement will be tested through a survey of players where they are asked to play the game and have a series of questions for the user regarding their experience with the interface recorded. The speed requirement can be tested through automatic unit testing, where the amount of time to execute commands is recorded.

2.2.3 Functional Unit Testing

All unit testing will be done automatically using Junit

Player:

Each one of Players methods will be tested for correct outputs using various inputs including boundary conditions Each one of Players fields will be tested for correct outputs

Deck:

Each one of Decks methods will be tested for correct outputs using various inputs including boundary conditions Each one of Decks fields will be tested for correct outputs

Game:

Each one of Games methods will be tested for correct outputs using various inputs including boundary conditions Each one of Games fields will be tested for correct outputs

Card:

Each one of Cards methods will be tested correct outputs using various inputs including boundary conditions Each one of Cards fields will be tested for correct outputs

2.3 Functional System

2.3.1 Performance Requirements:

Speed requirements:

Speed requirements will be automatically tested using using Junit and Java stopwatch utility.
Test case 1: Start program
Initial State: N/A
Input: N/A
Expected Output: stopwatch returns that program window starts up in less than 3 seconds

Test case 2: Computers play method is called
Initial state(s): game started, player has made move, computer's turn
Input: Card hand, chips, pot
Expected results: stopwatch returns that play method returns results in less than 2 seconds

Test case 2.1: Empty hand error
input: Card hand (empty), chips, pot
expected results: null pointer exception is returned

Test case 2.2: Zero chip error
input: Card hand, zero chips, pot
expected results: an error should be called

Test case 2.3: Zero pot error
input: Card hand, chips, zero pot
expected results: division by zero error

Precision requirements:
Precision requirements will be automatically tested using Junit and a Texas HoldEm hand probability database.

Test case 1: Hand strength method is called
initial state(s): game has started, player has made move, computers turn

inputs: Card hand (all possible card hand combinations are used as inputs)
expected results: Hand Strength returns a double value, rounded to 4 decimals, equal to corresponding double (rounded to 4 decimals) in Texas HoldEm hand probability database.

Test case 2: Card accuracy
Initial state: game begins
Input: N/A
Expected Output: The two players hands are dealt along with the community cards. There should not be any duplicate cards in the deck.

Availability requirements:
Availability requirements testing will consist of all testing specified in this document, using machines running OSX, Windows, and Linux.
Expected results: All results are consistent across all the platforms that were tested on

Regression Tests:

Regression will be tested using automatic testing. With the use of Junit, any change to the system that invalidate the previous positive test results will appear as a fail and therefore shows that the change needs to be revised.

Error Handling Tests:

Error handling will be tested using manual and automatic tests. When an error occurs, the system should display some form of error prompt to alert the user, and have the error logged. The logged error will be tested automatically for accuracy and the prompt will be manually tested for accuracy and the follow up action should resolve the error.

Usability Tests:

Usability requirements will be tested through manual testing. A small sample (1-10 people) of non beginners (people who are familiar with the rules of Texas Hold'em, and have played before), will play a round of the game to completion. They will be timed and later be asked to comment on their experience with the task.

Learnability Tests:

Learnability will be tested through manual testing. A small sample (1-10 people), of beginners will be selected and they will be required to perform tasks such as going through the help page and then playing a game based on what theyve learned from the help page. They will be timed and later be asked to comment on their experience with the task.

Playability Tests:

Playability will be tested through dynamic, unit system testing. If the system tests prove to be adequate and conclusive, then the game will be playable.

3 Schedule

Table 3: Schedule Table

Date	Team Member	Task
October 30	Nareshkumar Maheshkumar	Write unit test cases relating to PoC
November 2	All member	Execute Functional Testing Plan
November 8	All member	Create unit test for final product features
November 14	All member	Conduct manual testing on the game
Ongoing	All members	Update and/or revise test cases when new methods or classes are added
Ongoing	All members	Static testing for every method and class