Testing

Software testing is a process used to identify the *correctness, completeness and quality* of developed computer software. It includes a set of activities conducted with the intention of finding errors in software so that they can be corrected before the product is released to end users.

To ensure the security of my implementation, I have implemented throws for the different data-structures for things such as index out of bounds errors etc. Outside of the data-structures I developed the program so that the user's input would be validated before being able to be used on anything.

1.1 – System Testing

To verify that the overall final completed product conforms to its originally stated requirements we will preform a type of black-box testing known as System Testing. We will use various test cases which are a set of actions for the tester to perform that validate a particular function of the system. For each requirement we will test various conditions such as normal, boundary and error conditions. If a test fails, corrective action will be taken to ensure that the system functions according to its requirements.

1.1.1 - Streak.java

METHOD	validateChoice(Scanner input, String type, String[] choiceArray, int handLength);					
METHOD DESCRIPTION	Firstly, this method will print out a message depending on its type and add the users input to a String. Following that it compare the user's input to each element of the array, if the input is found in the array it will return that value, if it is not found it will print out a message to indicate that the input has not been accepted and recall the method.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing the choice methods validation and determining if it returns the correct value based on the users input when the type is *"menu"*, this will test: 1. Integers that are found in the array 2. Integers that are not found in the array 3. Characters 4. Null input 5. White space 6. An acceptable integer, with white space and a character.	*String[] digits = {"1", "2", "3", "9"};* *validateChoice(input, "menu", digits, 0);* 1. "1", "2", "3", "9" 2. "0", "4", "10" 3. "A" 4. null 5. " " 6. "1 A"	The message *"Enter Choice > "* will be printed out. • *1.* Input will be accepted and returned. • *2-6.* Input will be rejected, the message *"Input does not contain a choice from the list, please try again."* will be displayed and the method will be recalled.	The message *"Enter Choice > "* is printed out. • *1.* Input is accepted but the value returned is the ASCII value for the digit inputted instead of the actual value intended. • *2-5.* Input is rejected, the message *"Input does not contain a choice from the list, please try again."* Is displayed and the method is recalled. • *6.* "1" is accepted as the input and returned.	1. Fail 2. Pass 3. Pass 4. Pass 5. Pass 6. Fail	1. C.A.#1 6. C.A.#2
2	Testing the choice methods validation and determining if it returns the correct value based on the users input when the type is *"game"*, this will test:	*String[] digits = {"5", "6", "7", "8", "9", "10"};* *validateChoice(input, "game", digits, 0);*	The message *"How many cards do you want to play with (5 - 10) > "* will be printed out.	The message *"How many cards do you want to play with (5 - 10) > "* is printed out.	1. Pass 2. Pass 3. Pass 4. Pass 5. Pass	N/A

	1. Integers that are found in the array 2. Integers that are not found in the array 3. Characters 4. Null input 5. White space 6. An acceptable integer, with white space and a character.	1. "5", "6", "7", "8", "9", "10" 2. "4", "11" 3. "A" 4. null 5. " " 6. "1 A"	• *1.* Input will be accepted and returned. • *2-6.* Input will be rejected, the message *"Input does not contain a choice from the list, please try again."* will be displayed and the method will be recalled.	• *1.* Input is accepted and it returns the correct value. • *2-6.* Input is rejected, the message *"Input does not contain a choice from the list, please try again."* Is displayed and the method is recalled.	6. Pass	
3	Testing the choice methods validation and determining if it returns the correct value based on the users input when the type is *"card"*, this will test: 1. Characters that are found in the array. 2. Exiting the choice method. 3. Characters that are found in the array but not available due to the number of cards in the hand. 4. Lowercase characters. 5. Characters that are not found in the array. 6. Integers 7. Null input 8. White space 9. An acceptable character, white space and an integer.	*String[] letters = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"};* *validateChoice(input, "card", letters);* 1. "A", "B", "C", "D", "E" 2. "X", "x" 3. "F", "G", "H", "I", "J" *Int handLength = 5;* *validateChoice(input, "card", letters, handLength);* 4. "a", "b", "c", "d", "e" 5. "F" 6. 1 7. null 8. " " 9. "A 1"	The message *"Choose card to change or X to exit > "* will be printed out. • *1, 4.* Input will be accepted and return the position of the element in the array. • *2.* Input will be accepted and return the value of -1. • *3, 5-9.* Input will be rejected, the message *"Input does not contain a choice from the list, please try again."* will be displayed and the method will be recalled.	The message *"Choose card to change or X to exit > "* is printed out. • *1.* Input is accepted and returned the position of the element in the array. • *2.* Input is accepted and returned the value of -1. • *3.* Input is accepted and returned the position of the element in the array. However, this gives an index out of bounds error. • *4 - 9.* Input is rejected, the message *"Input does not contain a choice from the list, please try again."* Is displayed and the method is recalled.	1. Pass 2. Pass 3. Fail 4. Fail 5. Pass 6. Pass 7. Pass 8. Pass 9. Pass	3. C.A.#3 4. C.A.#4
4	Testing the choice methods validation and determining if it returns the correct value based on the users input when the type is *"replay"*, this will test: 1. Characters that are found in the array. 2. Lowercase characters. 3. Characters that are not found in the array.	*String[] letters = {"Y", "N"};* *validateChoice(input, "replay", letters, 0);* 1. "Y", "N" 2. "y", "n" 3. "A" 4. 1 5. null 6. " " 7. "Y 1"	The message *"See replay? (y/n) > "* will be printed out. • *1-2.* Input will be accepted and return the position of the element in the array. • *3-7.* Input will be rejected, the message *"Input does not contain a choice from the list,*	The message *"See replay? (y/n) > "* is printed out. • *1-2.* Input is accepted and returned the position of the element in the array. • *3-7.* Input is rejected, the message *"Input does not contain a choice from the list, please try again."* is	1. Pass 2. Pass 3. Pass 4. Pass 5. Pass 6. Pass 7. Pass	N/A

	4. Integers 5. Null input 6. White space 7. An acceptable character, white space and an integer.		*please try again."* will be displayed and the method will be recalled.	displayed and the method was recalled.		
5	As this is a recursive method, I need to test wether it will still return a validated choice after it has failed atleast once followed by a successful input.	Since the method will do the same thing regardless of its type and choiceArray we will use "replay" as an test. *String[] letters = {"Y", "N"};* *validateChoice(input, "replay", letters, 0);* • The first input will be one that is rejected so in this case I will use null. • The second input will be "y" as it will be accepted.	The message *"See replay? (y/n) > "* will be printed out. Input will be rejected, the message *"Input does not contain a choice from the list, please try again."* will be displayed and the method will be recalled. The message *"See replay? (y/n) > "* will be printed out. Input will be accepted and return the position of the element in the array.	The message *"See replay? (y/n) > "* is printed out. Input is rejected, the message *"Input does not contain a choice from the list, please try again."* is displayed and the method is recalled. The message *"See replay? (y/n) > "* is printed out. Input is accepted and returned the position of the element in the array.	Pass	N/A

METHOD	validateName(Scanner input, int playerNum)					
METHOD DESCRIPTION	Firstly, this method will print out a message and add the users input to a String. It will preform three checks, one is if the length of the string is greater than 10, two is if the input is null and three if the input contains any letters. If it passes all three checks the string will be returned. If it fails a message will be displayed to indicate what the problem is and recall the method.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing the name methods validation and determining if it returns the correct value based on the users input, this will test: 1. A String that is less than ten characters in length, contains only letters and no spaces. 2. A String that is ten characters in length, contains only letters and no spaces. 3. A string that is greater than ten characters. 4. A string that contains characters that aren't letters.	*validateName(input, 1);* 1. "Jack" 2. "Montgomery" 3. "GreaterThanTen" 4. "123!@" 5. "Jack K" 6. null 7. " "	The message *"Enter Player 1 name > "* will be printed out. • *1-2.* Input will be accepted and return a String with the value of the input. • *3, 6.* Input will be reject, the message *"Please enter a name with 10 characters or less"* will be displayed and the method will be recalled. • *4-5, 7.* Input will be rejected, the message *"Please enter a name with only letters and no*	The message *"Enter Player 1 name > "* is printed out. • *1-3.* Input is accepted and returned a String with the value of the input. • *4-5, 7.* Input is rejected, the message *"Please enter a name with only letters and no spaces."* is displayed and the method is recalled. • *6.* Input is rejected, the message *"Please enter a name withs 10 characters or less"* is	1. Pass 2. Pass 3. Fail 4. Pass 5. Pass 6. Pass 7. Pass	3. C.A.#5

	5. A string that contains a space. 6. Null input. 7. White space.		*spaces."* will be displayed and the method will be recalled.	displayed and the method is recalled.		
2	As this is a recursive method, I need to test wether it will still return a validated choice after it has failed atleast once followed by a successful input.	*validateName(input, 1);* • The first input will be one that is rejected so in this case it will be "" • The second input will be "Jack" as it will be accepted.	The message *"Enter Player 1 name > "* will be printed out. Input will be reject, the message *"Please enter a name with 10 characters or less"* will be displayed and the method will be recalled. The message *"Enter Player 1 name > "* will be printed out. Input will be accepted and return a String with the value of the input.	The message *"Enter Player 1 name > "* is printed out. Input is reject, the message *"Please enter a name with 10 characters or less"* is displayed and the method is recalled. The message *"Enter Player 1 name > "* is printed out. Input is accepted and returned a String with the value of the input.	Pass	N/A

METHOD	displayReplay(Scanner input, int index, String name)					
METHOD DESCRIPTION	This method will display the entire contents of the REPLAY and REPLAY_SELECTION queue until they are both empty, it will be formatted exactly to how the hand was originally displayed.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Displaying the contents of both queues after all possible cards have been replaced.	A standard hand array with a length of 5. When prompted to choose a card the users chooses a card for each time they are prompted (5 card selections). The user will then choose to see the replay.	For each hand to be displayed, it will display the players name, following with each Card object in the hand and its corresponding letter being displayed. This should print out 6 hands as 5 cards are changed. The selection of the card from the corresponding hand will be displayed. This should only be displayed 5 times as 5 choices were made leaving the last hand with no selection.	For each hand displayed, the players name is displayed, following this with each Card object in the hand and its corresponding letters being displayed. There is a total of 6 hands displayed. The selection that was made in the corresponding hand is displayed, this is repeated 5 times for every selection and the last hand is displayed with no selection.	Pass	N/A
2	Displaying the contents of both queues after only one card has been selected to be exchanged.	A standard hand array with a length of 5. When prompted to choose a card the user will do this once and the selection. The user will then choose to see the replay.	For each hand to be displayed, it will display the players name, following with each Card object in the hand and its corresponding letter being displayed. This should print out 2 hands as 1 card is changed. The selection of the card from the corresponding hand will be	For each hand displayed, the players name is displayed, following this with each Card object in the hand and its corresponding letters being displayed. There is a total of 2 hands displayed. The selection that was made in the corresponding hand is displayed for	Pass	N/A

			displayed. This should only happen once for the first hand leaving the last hand with no selection.	the first hand. The last hand does not have a selection.		
3	Displaying the contents of both queues after no cards have been replaced.	A standard hand array with a length of 5. When prompted to choose a card the user exits. The user chooses to see the replay.	The name of the player following each Card object of the hand along with is corresponding lettering for the card number will be displayed and a selection for the game will not be displayed as there was none made. Only one hand should be displayed.	The name of the player was printed, following this each Card object in the hand was displayed with its corresponding letters. There was no selection displayed. Only one hand was displayed	Pass	N/A

METHOD	displayScoreboard()					
METHOD DESCRIPTION	This method will display up to the first five entries of the scoreboard, if there are no entries to display a message will indicate this.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Displaying the contents of the scoreboard when there are 5 or more entries.	The SCOREBOARD data-structure will hold 6 entries. (A 2-Player game is played)	It will display the first five entrys held in the SCOREBOARD data-structure	Five entries from the scoreboard is displayed.	Pass	N/A
2	Displaying the contents of the scoreboard when there is 1 entry.	The SCOREBOARD data-structure will hold 1 entry. (A Single-Player game is played)	It will display the first entry held in the SCOREBOARD data-structure.	One entry from the scoreboard is displayed.	Pass	N/A
3	Displaying the contents of the scoreboard when there are no entries.	The SCOREBOARD data-structure will hold no entries. (No games are played)	There will be no entries held in the SCOREBOARD data-structure so it will print out the message *"There are no scores currently held in the scoreboard."*	No entries are displayed and the message *"There are no scores currently held in the scoreboard."* Is displayed.	Pass	N/A

METHOD	sortScoreboard(ListChain<Scoreboard> scoreboard)					
METHOD DESCRIPTION	This method will sort the ListChain data-structure containg Scoreboard objects in descending order of score as determined by the Scoreboard ADTs compareTo operator.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Sorting the contents of the data-structure in descending order of score when there is 5 entries.	The SCOREBOARD data-structure will hold 5 entries with the scores *(1, 8, 7, 5, 3)*	The data-structure will be sorted in descending order of score, no entries will be removed. The scores held in the data-structure should now be *(8, 7, 5, 3, 1)*	The data-structure is sorted in descending order of score, no entries are removed. The scores held in the data-structure are now *(8, 7, 5, 3, 1)*	Pass	N/A
2	Sorting the contents of the data-structure in descending order of score when there is more than 5 entries.	The SCOREBOARD data-structure will hold 8 entries with the scores *(4, 5, 4, 2, 1, 3, 1, 6)*	The data-structure will be sorted in descending order of score, 3 entries will be removed. The	The data-structure is sorted in descending order of score, 3 entries are removed. The scores held in the data-structure are now *(6, 5, 4, 4, 3)*	Pass	N/A

			scores held in the data-structure should now be *(6, 5, 4, 4, 3)*			
3	Sorting the contents of the data-structure in descending order of score when there is less than 5 entries.	The SCOREBOARD data-structure will hold 3 entries with the scores *(4, 6, 5)*	The data-structure will be sorted in descending order of score, no entries will be removed. The scores held in the data-structure should now be *(6, 5, 4)*	The data-structure is sorted in descending order of score, no entries were removed. The scores held in the data-structure are now *(6, 5, 4)*	Pass	N/A
4	Sorting the contents of the data-structure in descending order of score when there are only 3 entries all of the same value.	The SCOREBOARD data-structure will hold 3 entries with the scores *(4, 4, 4)*	The data-structure will be attempted to be sorted in descending order of score and no entries will be removed. Since they are all of the same value no changed will be made so the scores held in the data-structure should still be *(4, 4, 4)*	The data-structure attempted to be sort the entries in descending order of score, however the values are the same so no changes are made. No entries are removed. The scores held in the data-structure remain as *(4, 4, 4)*	Pass	N/A
5	Sorting the contents of the data-structure in descending order of score when it has already been sorted.	The SCOREBOARD data-structure will hold 5 entries with the scores *(8, 7, 5, 3, 1)*	The data-structure will be attempted to be sorted in descending order of score and no entries will be removed. Since it is already in descending order no changes should be made.	The data-structure attempted to be sorted in descending order of score and no entries were removed. Since it is already in descending order no changes were made.	Pass	N/A

METHOD	sortHand(Card[] hand)					
METHOD DESCRIPTION	This method will sort the array of Card Objects into ascending order of the cards rank value, this is determined by the Card ADTs compareTo operator.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Sorting the contents of the hand in descending order of rank value when each Card objects rank value is different.	The hand array will hold 5 entries with the Card objects rank values being, *(6, 3, Ace, 8, Jack)*	The Card objects rank values in the hand array will be sorted as the following, *(3, 6, 8, Ace, Jack)*	The Card objects rank values in the hand array are sorted as the following, *(3, 6, 8, Ace, Jack)*	Pass	N/A
2	Sorting the contents of the hand in descending order of rank value when each there are Card object ranks of the same value.	The hand array will hold 5 entries with the Card objects rank values being, *(6, 6, 3, 3, Jack)*	The Card objects rank values in the hand array will be sorted as the following, *(3, 3, 6, 6, Jack)*	The Card objects rank values in the hand array are sorted as the following, *(3, 3, 6, 6, Jack)*	Pass	N/A
3	Sorting the contents of the hand in descending order of rank value when 4 of the Card Objects rank value are the same.	The hand array will hold 5 entries with the Card objects rank values being, *(6, 6, 6, 6, 3)*	The Card objects rank values in the hand array will be sorted as the following, *(3, 6, 6, 6, 6)*	The Card objects rank values in the hand array are sorted as the following, *(3, 6, 6, 6, 6)*	Pass	N/A
4	Sorting the contents of the hand in descending order of rank value when the hand array has already been sorted.	The hand array will hold 5 entries with the Card objects rank values being, *(3, 6, 8, Ace, Jack)*	No changes will be made to the hand array. The contents will remain as *(3, 6, 8, Ace, Jack)*	No changes are made to the hand array. The contents remain the same being *(3, 6, 8, Ace, Jack)*	Pass	N/A

METHOD	playSingleplayer(Scanner input)					
METHOD DESCRIPTION	This method will create a hand array with the length of cards chosen by the user and validated by validateChoice. It will create a new deck and populate the hand with cards dealt from the deck. It will call the validateName method to get the players name and call the playRound() method to play out a round and get the score. A new Scoreboard object will be created with the name and score then added to the SCOREBOARD data-structure. Finally, it will ask the user if they want to see the replay, if not, the REPLAY and REPLAY_SELECTION data-structures will be cleared.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to see if the method creates a hand array with the length given.	*Card[] hand = new Card[numCards];* *numCards = 6;*	A hand array will be created with a length of 6	A hand array is created with a length of 6	Pass	N/A
2	Testing to see if the method populates the hand array with cards dealt from the deck.	Since this process is randomised, I have included debug messages in various classes to display the contents of the deck created, the card that was dealt from the deck and the contents of the hand.	The hand will be fully populated with cards dealt from the deck. The card being dealt from the deck will be at the top and increment down.	The hand is fully populated with cards dealt from the deck. The card being dealt from the deck is at the top and increment down.	Pass	N/A
3	Testing to see if the SCOREBOARD entry is added.	*Scoreboard scoreboardEntry = new Scoreboard(player, score);* *player = "Jack";* *score = 4;*	This entry will be displayed on the Hi-Score table.	This entry is displayed on the Hi-Score table.	Pass	N/A
4	Testing to see if the REPLAY and REPLAY_SELECTIONS queues are cleared if the user does not which to see the replay.	To determine if the queues were cleared I have included debug messages using the isEmpty operator.	A message will be displayed indicating that both queues are empty	A message is displayed indicating that both queues are empty.	Pass	N/A

METHOD	play2Player(Scanner input)					
METHOD DESCRIPTION	This method will create two hand arrays for each player with the length of cards chosen by the user and validated by the method validateChoice. For each round it will create a new deck and populate each hand alternating with cards dealt from the deck. It will call the validateName method to get both players name and call the playRound() method for both players to get the score for the round. A new scoreboard object will be created for both players with their name and their score then added to the SCOREBOARD data-structure after the round has been completed. There will be three rounds, at the end of each round the scores for both players will be displayed and at the end of the three rounds a match score will be displayed.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to see if the method creates two hand arrays with the length given.	*Card[] hand1 = new Card[numCards];* *Card[] hand2 = new Card[numCards];* *numCards = 6;*	Two hand arrays will be created with a length of 6.	A hand array is created with a length of 6.	Pass	N/A
2	Testing to see if the method populates both hand arrays alternating for each entry with cards dealt from the deck.	Since this process is randomised, I have included debug messages in various classes to display the contents of the deck created, the hand currently being populated and the card that was dealt from the deck and the contents of both the hands.	Both hands will be fully populated with cards dealt from the deck. The card being dealt from the deck will be at the top and increment down.	Both hands are fully populated with cards dealt from the deck. The card being dealt from the deck is at the top and increment down.	Pass	N/A
3	Testing to see if the SCOREBOARD entries for both players is added.	*Scoreboard scoreboardEntry1 = new Scoreboard(player1, score1);*	Both of these entries will be displayed on the Hi-Score table.	Both of these entries are displayed on the Hi-Score table.	Pass	N/A

		player1 = "Jack"; *score1 = 4;* *Scoreboard scoreboardEntry2 = new Scoreboard(player2, score2);* *player1 = "Amy";* *score1 = 3;*				
4	Test to determine if the match score for the player in question is the total of the 3 individual round scores combined	1. *score1 = 3* 2. *score 1 = 5* 3. *score 1 = 0* *matchScore1 += score1; x3*	The match score for the player in question displayed at the end of a match should be "8".	The match score for the player in question is displayed at the end and has the value of "8".	Pass	N/A

METHOD	playRound(Scanner input, Card[] hand, Deck deck, String name, Boolean isSingleplayer)					
METHOD DESCRIPTION	This method will display the current hand giving the user the prompt to exchange a card or exit, this will repeat up to the amount of cards able to be exchanged or when the user decides to exit. If it is a singleplayer game each Card Object in the hand array and the card selected from that array will be enqueued into its respective Queue data-structure.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Determining if the card selected by the user is successfully replaced in the hand when the hand length is 5. 1. First position in the hand 2. Last position in the hand 3. Middle position in the hand	1. *cardChoice = 0* 2. *cardChoice = 2* 3. *cardChoice = 4* *hand[cardChoice] = deck.deal();*	• *1-3.* The Card Object in the position specified in the hand array will be replaced by one dealt from the deck	• *1-3.* The Card Object in the position specified in the hand array is replaced by one dealt from the deck	Pass	N/A
2	Determining if the card selected by the user is successfully replaced in the hand when the hand length is 10. 1. First position in the hand 2. Last position in the hand 3. Middle position in the hand	1. *cardChoice = 0* 2. *cardChoice = 4* 3. *cardChoice = 9* *hand[cardChoice] = deck.deal();*	• *1-3.* The Card Object in the position specified in the hand array will be replaced by one dealt from the deck	• *1-3.* The Card Object in the position specified in the hand array is replaced by one dealt from the deck	Pass	N/A
3	Determing if the round is successfully completed when the user decides to exit.	*cardChoice = -1*	The round will complete and the final score for the last hand displayed will be returned	The round is completed and the final score for the last hand displayed is returned.	Pass	N/A
4	Determing if the final round score for the last hand is returned when a full round is completed (i.e., the final hand will not be displayed but the score for that hand will be returned)	To check that it has not returned the last seen hands score, we can simply compare the score for the last hand seen to the final round score.	The final round score will be different from the last displayed hands score.	The final round score is different from the last displayed hands score.	Pass	N/A
5	Determining if, in a Single-Player game a hand and the hand selection is successfully enqueued to its respective queue data-structure when:	The hand length with be 5 so the user can choose up to 5 cards to exchange. 1. A full round will be played with the user	1. *1-3.* Each Card Object in the hand array and each card selection will be enqueued successfully to its	*1-3.* Each Card Object in the hand array and each card selection is enqueued successfully to its respective data-structure and when viewed in the replay, matches what	Pass	N/A

	1. A full round is played 2. One card is exchanged 3. No cards are exchanged	selecting a card to exchanged when prompted. 2. Only one card will be selected when the user is prompted and then they will choose to exit 3. When given the prompt the user will choose to exit the selection. To determine if they are properly enqueued I will choose to view the replay ensuring that each hand is identical and the card selection is right.	respective data-structure and when viewed in the replay, should match was was displayed and chosen during the round.	was displayed and chosen during the round.		

METHOD	getScore(Card[] cards)					
METHOD DESCRIPTION	This method is used to determine the score for the round, it will call the getStreak method to get the streak of the current card, this process will repeat until every card in the array as been compared. This method gets the highest streak by determining if the current streak is greater than the last streak found, if this is true it will set the highest streak value to that of current streak and find the bonus for that streak. If there is a situation where they are equal, this will determine which has the highest bonus and apply that bonus.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Determine if, given a hand, if the method successfully returns a score when: 1. There is no streak in the hand. 2. There is one streak in the hand 3. There are multiple streaks in the hand of different lengths 4. There are multiple streaks in a hand with the largest two being of the same length 5. There are multiple streaks in the hand, one streak length is smaller but its bonus make its value larger than the highest streak.	Since this process is randomised, I have included debug messages in various classes to display the all the statistics on the current streak, this will provide the information required to understand if the system is operating functionally, for each test I will provide the length of the streak and its bonuses if applicable to record what was printed. 1. "1" 2. *"2"* 3. *"3", "2"* 4. *"2 + 1 bonus" "2 + no bonus"* 5. *"2 + 2 bonus", "3"*	• 1. The method will return the value of 1 as there is only 1 card in the streak. • 2 – 3. The method will return the length of the highest streak found • 4. The method will return the length of the streak value and the greatest bonus found in both streaks. • 5. The method will ignore the streak value and instead return the value of the highest streak.	1. The value "1" is returned 2. The value "2" is returned 3. The value "3" is returned 4. The value "3" is returned 5. The value "3" is returned	1. Pass 2. Pass 3. Pass 4. Pass 5. Pass	N/A

METHOD	getStreak(Card[] cards, int startingPos)

METHOD DESCRIPTION	This method will find a streak of cards in the hand given the starting position, it will compare each card in the hand and increment the streak counter until the current card is not a streak with the next card, then it is broken and return the value of the streak counter.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing if this method will successfully count how many cards are in a streak, given the starting position of the hand	*The values this method will be using is (2, 3 ,5, Jack, King)*	The For Loop will break when it reaches the value "5" therefore returning the result of 2	The For Loop was broken when it reached the value of "5" and return the result of 2	Pass	N/A

METHOD	getBonus(Card[] cards, int startingPos, in streakLength)					
METHOD DESCRIPTION	This method will find the bonuses to apply in a given streak, it will compare the current card to the next card to determine if it is of the same suit, if either is false it will set a Boolean two indicate this and if both are false there are no bonuses to apply and it will return the value of zero.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing this method to indicate if it will successfully determine if all the cards in the streak are: 1. Of the same colour 2. Of the same suit 3. Neither	*The values this method is using are:* *1. "Hearts", "Diamonds"* *2. "Hearts", "Hearts"* *3. "Hearts", "Spades"*	1. Returns the value 1 2. Returns the value 2 3. Returns the value 0	1. Returned the value 1 2. Returned the value 2 3. Returned the value 0	1. Pass 2. Pass 3. Pass	N/A

METHOD	getBonus(Card[] cards, int startingPos, in streakLength)					
METHOD DESCRIPTION	The main method will display a repeating menu to allow the user to navigate through the various options provided until they decide to exit the system.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing each case in the switch to determine if they do as intended 1. Singleplayer game 2. 2-Player game 3. View Hi-Score Table 4. Exit	*1. "1"* *2. "2"* *3. "3"* *4. "9"*	1. Calls the playSingeplayer method 2. Calls the play2Player method 3. Calls the displayScoreboard method 4. Sets the value of menu to false, therefore the menu wont loop and the program will exit	1. Called the playSingleplayer method 2. Called the play2Player method 3. Called the displayScoreboard method 4. Set the value of menu to false and the program exits	1. Pass 2. Pass 3. Pass 4. Pass	N/A

1.1.2 Card.java

OPERATOR	compareTo(Card otherCard)					
OPERATOR DESCRIPTION	This operator will compare this card to another card given and return the result depending on wether the rank value is less than, greater than or that it is equal too					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to determine, if given the value of two cards, will return the intended value when: 1. The first card is greater than the second 2. The first card is less than the second 3. The first and second card are equal.	*Card1 = "4 of Hearts"* *Card2 = "2 of Diamonds"* 1. *Card1.compareTo(Card2)* 2. *Card2.compareTo(Card1)* 3. *Card1.compareTo(Card1)*	1. Returns the result of 1 as Card1 is greater than Card2 2. Returns the result of -1 as Card2 is less than Card1 3. Returns the result of 0 as Card1 is equal too Card1	1. Returned the result 1 2. Returned the result -1 3. Returned the result 0	1. Pass 2. Pass 3. Pass	N/A

OPERATOR	isStreak(Card otherCard)					
OPERATOR DESCRIPTION	This operator will return a Boolean value if this card is the same position in the RANKS array as the next card minus one.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to determine, if given the value of two cards, will return the intended value when: 1. The first card and the second card are a streak 2. The first card and the second card are not a streak 3. The first card and the second card are of the same value.	*Card1 = "2 of Hearts"* *Card2 = "3 of Diamonds"* 1. *Card1.isStreak(Card2)* 2. *Card2.isStreak(Card1)* 3. *Card1.isStreak(Card1)*	1. Will return the result of true 2. Will return the result of false 3. Will also return the result of false.	1. Returned the result true 2. Returned the result false 3. Returned the result false	1. Pass 2. Pass 3. Pass	N/A

OPERATOR	getColour					
OPERATOR DESCRIPTION	This operator will return the colour of this card.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to determine, if given the value of two cards, will return the intended value when: 1. The colour is Red 2. The colour is black	*Card1 = "2 of Hearts"* *Card2 = "3 of Spades"* 1. *Card1.getColour()* 2. *Card2.getColour()*	1. Will return a String value of "Red" 2. Will return a String value of "Black"	1. Returned the String "Red" 2. Returned the String "Black"	1. Pass 2. Pass	N/A

OPERATOR	getColourBonus(Card otherCard)					
OPERATOR DESCRIPTION	This operator will return a Boolean value depending on if this card is the same colour as the next card given.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to determine, if given the value of two cards, will return the intended value when: 1. The cards are of the same colour 2. The cards are not of the same colour	*Card1 = "2 of Hearts"* *Card2 = "3 of Spades"* *1. Card1.getColourBonus(Card1)* *2. Card1.getColourBonus(Card2)*	1. Will return true as they are the same colour 2. Will return false as they are not the same colour	1. Returned true 2. Returned false	1. Pass 2. Pass	N/A

OPERATOR	getSuitBonus(Card otherCard)					
OPERATOR DESCRIPTION	This operator will return a Boolean value depending on if this card is the same suit as the next card given.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to determine, if given the value of two cards, will return the intended value when: 1. The cards are of the same suit 2. The cards are not of the same suit	*Card1 = "2 of Hearts"* *Card2 = "3 of Spades"* *1. Card1.getSuitBonus(Card1)* *2. Card1.getSuitBonus(Card2)*	1. Will return true as they are the same suit 2. Will return false as they are not the same suit.	1. Returned true 2. Returned false	1. Pass 2. Pass	N/A

1.1.3 Scoreboard.java

OPERATOR	compareTo(Scoreboard otherScore)					
OPERATOR DESCRIPTION	This operator will compare this score to another score given and return the result depending on wether the score is less than, greater than or that it is equal too					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to determine, if given the value of two cards, will return the intended value when: 1. The first card is greater than the second 2. The first card is less than the second 3. The first and second card are equal.	*Scoreboard1 = "2"* *Scoreboard2 = "3"* *1. Card1.compareTo(Card2)* *2. Card2.compareTo(Card1)* *3. Card1.compareTo(Card1)*	1. Returns the result of 1 as Scoreboard1 is greater than Scoreboard 2 2. Returns the result of -1 as Scoreboard 2 is less than Scoreboard 1 3. Returns the result of 0 as Scoreboard 1 is equal too Scoreboard1	1. Returned the result 1 2. Returned the result -1 3. Returned the result 0	1. Pass 2. Pass 3. Pass	N/A

1.1.4 Deck.java

METHOD	Shuffle()					
METHOD DESCRIPTION	This method is used to shuffle all the cards in the deck.					
TEST CASE #	**TEST DESCRIPTION**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS/ FAIL**	**C.A. #**
1	Testing to see if the method throughoutly shuffles the deck of cards as intended	Since this process is randomised, I have included debug messages in various classes to display what cards are in the deck before being shuffled, the cards and positions that have been selected to be swapped and the deck after being shuffled.	The deck will be fully shuffled, the amount of swaps will be for the amount of entries in the deck, and there will be no duplicates of entries.	The deck is fully shuffled, the amount of swaps was for the amount of entries in the deck and there were no duplicate entries.	Pass	N/A

1.2 – Corrective Action

C.A. #1

In *Acceptance Test 1.1 of the choice method* it returns the ASCII value for the integer inputted instead of the value the user intended. This happens because the method returns an integer value, originally I had it return the character at position 0 in the string which returns the ASCII value instead. So, to correct this I will use Integer.parseInt() in order to convert the full string to an integer.

Before Corrective Action:
return choice.charAt(0);

After Corrective Action:
return Integer.parseInt(choice);

C.A. #2

In *Acceptance Test 1.6 of the choice method* it returns the value "1" when the full string is "A 1". This happens because the scanner present in the method was originally only scanning the next input. So, to correct this I will scan the next line instead so a space cannot break it.

Before Corrective Action:
String choice = input.next();

After Corrective Action:
String choice = input.nextLine();

C.A. #3

In *Acceptance Test 3.3 of the choice method* it accepts the choice given as it is found in the array but causes an index out of bounds error after the value has been returned because the number of cards in the hand is less than the choice selected. So, to correct this I will incorporate the length of the hand into the choice method, if the choice selected is in the array but is greater than the length of the hand it will print out a message and recall the choice method.

Before Corrective Action:

```
public static int validateChoice(Scanner input, String type, String[] choiceArray, int handLength) {
        …
}
```

After Corrective Action:

```
public static int validateChoice(Scanner input, String type, String[] choiceArray, int handLength) {
        …
        if(type.equals(“card”) && i > handLength - 1) {
                System.out.println(“Input does not contain a choice from the list, pleast try again.”);
                return choice(input, type, choiceArray, handLength);
        }
}
```

C.A. #4

In *Acceptance Test 3.4 of the choice method* it doesn't accept the lower case values for the options provided, in order to correct this I will create a temporary array with lowercase values for each element of the original array. This new array also needs to be compared to each character in the string alongside the original to include this I will simply use an OR (||) statement inside the If statement that is used to determine if the character is found in the array.

Before Corrective Action:

```
if (… && choiceArray[i].equals(choice)) {
        …
}
```

After Corrective Action:

```
String[] lowercaseChoiceArray = new String[choiceArray.length];
…
if (type.equals("card") || type.equals("replay")) for (int i = 0; i < choiceArray.length; i++) lowercaseChoiceArray[i] = choiceArray[i].toLowerCase();
…
if (… && (choiceArray[i].equals(choice) || lowercaseChoiceArray[i].equals(choice))) …
```

C.A. #5

In Acceptance Test 6 of the name method it accepts a null input and returns this input as a String. The method should not allow for this so, to correct this I will implement a check to ensure the String is not blank.

Before Corrective Action:

```
if (name.length() > 10) {
        …
}
```

After Corrective Action:

```
if (name.length() > 10 || name.equals(“”)) {
        …
}
```