Task No 3b – Testing Process

You are expected to test your code using the strategies studied during the module.
The testing section of your documentation indicates the approach you have taken to verifying and validating your system. Just as you should not convey the design of your system by presenting the code or even listing the classes, you should not merely list the tests performed. Rather, discuss how tests were selected, why they are sufficient, why a reader should believe that no important tests were omitted, and why the reader should believe that the system will really operate as desired when in use.

The testing process set a number of test approaches aimed at ensuring the validity of the product as per system requirements.

Test Case Tables in the document

REQ Test Cases	Functional Testing – User Requirements (black / white box)
UNI Test Cases	Functional Testing – Unit Testing (white box) - UNI001 – Class `Result` - UNI002 – Class `Board` - UNI003 – Class `GameDrive` - UNI004 – Class `SmartComputer` - UNI005 – Class `NaiveComputer`
FUN Test Cases	Non–Functional Testing - Generic GUI Functionalities (black box)

Functional Testing – User Requirements

At the high-level view of the application, the testing screened the scope of the application and it's compliance to the list of requirements stated in the user documentation (in this case, assignment details setting the needed functionalities of the game). The functional test aimed to confirm that each functionality is available in the system and that system as a whole follows the game logic specified in the documentation. The objectivity of the testing was ensured through engagement of independent testers. Considering the limited scope of functionalities, requirements testing proved as a major confirmation of the system's conformity with the list of requirements.

Furthermore, the user requirements related to class design structure were evaluated against the code structure and supporting documentation: class design document, code walk-through and formal *javadoc* documentation.

Functional Testing – User Requirements was set prior to marking the product as ready for the production / live environment.

Functional Testing – Unit Testing

The unit testing aimed to support any future change and maintenance of the code. This ensures that testing automation is introduced for the system code, thus decreasing time in debugging and re-running tests once the enhancements in the system are implemented.

The Unit testing for the application requires further work as the full (or acceptable) coverage of the code was not achieved. The development process did not apply the suitable method to create the unit testing in parallel to the code development. The project timeline was severely affected by prolonged (and unplanned) work on other parts of the product, so decreasing the available resources (time) that was invested on the unit testing.

Non - Functional Testing – Generic GUI Functionalities

The additional functional testing focused on the generic functionalities of the user interface. The aim was to ensure that GUI components function as expected.

Non – Functional Testing – Others

Testing also aimed to confirm the reliability of the system in different working environments. System was placed under run and functionality testing was repeated while the computer system hosting the application was placed under heavy performance load. It also included functional test repetition in various computer systems with higher and lower performance capabilities.

The scope of the application limited the availability of options for running the non-functional testing (e.g. limited range in running load, stress or security testing). The system scope also prevented the testing of the upper limits of the usual software components like database, hardware and network, since the application does not contain any.

The additional testing process aimed to close off any unforeseen loopholes in the way system performs and also review usability level of the system. This ad hoc / exploratory approach included user experience evaluation (external tester) while at the same time

looking for errors or application behavior that seems non-obvious. User was left free to attempt any functionality and/or number of functionality steps aimed to break the logic of the game or bring the unexpected performance to the fore.

Overall, the functional / non-functional testing process at application level ensures product performance that will confirm most of the requirements set at pre-project phase. The existing functionality of the system delivers functional system that could be pushed to the production environment.

On the other side, the future work on the application needs to conclude the unit testing conformity prior to attempting to introduce code changes. This will guarantee that at the smallest testable part, the application code meets its design and requirements while providing environment for code reuse (side-effect that stems from the fact that unit testing naturally forces modularity of the code), easier bug tracking and simplified integration of the system components.

1) User Requirements Testing

ID	Purpose of the Test	Pre-condition	Test Data	Steps	Expected Result	Actual Result
REQ01	Confirm that program offers the user multiple games; at any point the user can abandon a game and start a new game.	1) GUI for the game must be available; 2) GUI will offer the functionality to restart the game; 3) GUI will offer the functionality to display the results;	Game process start-to-finish	1) User starts the game by picking the first player; 2) User plays several games against computer; 3) User opts for the game to restart; 4) User repeats step 1 to 3 several times;	1) User is able to start the game by picking who starts as first player; 2) GUI display for game result will list number 0 for data Me:, Computer:, Draw: and Games:; 3 User and Computer play the game; 4) User is able to repeat step 1 after game comes to conclusion (win, lose, draw); 5) GUI display for result incremented based on game outcome; 6) User is able to restart the game set by using the functionality (button) for game restart; 7) Process starts from step 1;	Pass

ID	Purpose of the Test	Pre-condition	Test Data	Steps	Expected Result	Actual Result
REQ02	For each game, the program will ask who should go first: the user or computer.	1) GUI for the game must be available; 2) GUI will offer the functionality to restart the game;	Game process start-to-finish	1) Game GUI available to player; 2) User starts the game by clicking the start button; 3) User is offered JOption pane input dialogue to pick who starts the game: Human or Computer; 4) User picks the choice and game starts; 5) Game concludes and process starts from step 3;	1) Game GUI offers the user JOption pane input dialogue; 2) User is able to start the game by picking who starts as first player; 3) User is able to repeat step 1 after game comes to conclusion (win, lose, draw) or if user decides to restart the game at any time;	Pass
REQ03	Throughout the game, the user plays by clicking a square on the grid. The display is immediately updated and a win or draw announced if this has happened.	1) GUI for the game must be available;	Game process start-to-finish	1) User starts the game by picking the first player; 2) User plays the move; 3) Computer plays the move; 4) Game continues until is finished (win, loss, draw); 5) New game starts from step 1;	1) User starts the game by picking the first player; 2) If user clicks the free area in the display, the display is populated by the character assigned to the user, and opposite of what computer move displays; 3) Illegal move (user clicks already-selected area) is prevented and user is warned; 4) Computer player plays the move immediately after human player made a move; 4) The outcome of the game (win, loss, draw) is automatically displayed by game GUI; 5) The game data is updated in the result display;	Pass

ID	Purpose of the Test	Pre-condition	Test Data	Steps	Expected Result	Actual Result
REQ04	Unless play has ended in a human win or a draw, the computer will choose a number. The display is immediately updated and a win or draw announced if this has happened.	1) GUI for the game must be available;	Game process start-to-finish	1) User starts the game by picking the first player; 2) User plays the move; 3) Computer plays the move; 4) Game continues until is finished (win, loss, draw); 5) New game starts from step 1 if Human win or draw; 6) If Computer win, Computer picks who starts first;	1) First game starts by Human picking the first player; 2) If game finishes by draw or Human win, Human player chooses who starts as first player in the next game; 3) If game finishes by Computer win, Computer player chooses who starts as first player in the next game;	Fail
REQ05	At any stage, a new game can be initiated, for instance, by clicking `New game', or the program exited. In case of a new game, the current game is aborted. The game state and displays should be correctly re-initialized on a new game.	1) GUI for the game must be available; 2) GUI will offer the functionality to restart the game; 3) GUI will offer the functionality to display the results;	Game process start-to-finish	1) User starts the game by picking the first player; 2) User plays several games against computer; 3) User opts for the game to restart; 4) User repeats step 1 to 3 several times;	1) User is able to start the game by picking who starts as first player; 2) GUI display for game result will list number 0 for data Me:, Computer:, Draw: and Games:; 3 User and Computer play the game; 4) User is able to repeat step 1 after game finishes (win, lose, draw); 5) GUI display incremented set on game outcome; 6) User can restart the game set by using the functionality (button) for game restart; 7) Process starts from step 1;	Pass

ID	Purpose of the Test	Pre-condition	Test Data	Steps	Expected Result	Actual Result
REQ06	For each game the program should randomly select either a \naive com-puter player" or a \smart computer player". The user must not be aware of which opponent s/he is facing. The naive computer player will make a random pick of the available numbers on each turn.	1) GUI for the game must be available; 2) GUI will offer the functionality to restart the game; 3) GUI will offer the functionality to display the results;	Game process start-to-finish	1) User starts the game by picking the first player; 2) User plays several games against computer; 3) User takes note of the game difficulty and evaluate the logic behind Computer moves;	1) User should encounter games when Computer move goes against the logic of winning the game so no strategy seems behind it. In such games, Human player should have an easy path to win. 2) User should also encounter games when Computer games employees strategy that displays the winning approach and blocks the Human player from winning. 3) There should not be any pattern or logic when Human player encounters smart or naive strategy by Computer player;	Pass
REQ07	The smart computer player will use a defensive and slightly aggressive strategy: (a) If one of the available cells will immediately result in a computer win, choose this; (b) otherwise, if one of the available cells would immediately give the human player win, choose this; (c) otherwise choose a random available cell.		Code for class SmartComputer	Manual evaluation of the code for class SmartComputer	To confirm the logic set in the game requirements (Purpose of the Test column)	Pass

ID	Purpose of the Test	Pre-condition	Test Data	Steps	Expected Result	Actual Result
REQ08	Confirm the logic of the game, that board selections of player Human and player Computer correctly result in Draw, Win, Lose outcome (3 identical vertical, horizontal or diagonal should result in Win outcome, otherwise Draw outcome once the board selection options are full);	1) GUI for the game must be available; 2) GUI will offer the functionality to restart the game; 3) GUI will offer the functionality to display the results;	Game process start-to-finish	1) User starts the game by picking the first player; 2) User plays several games against computer; 3) After each game conclusion, board is surveyed for input setup and compared with the result registered by system;	1) Game concludes if board is full or one of the players manages to select the winning combination (3 identical vertical, horizontal or diagonal inputs); 2) Game cannot continue if one of the players gets the winning combination;	Pass

2) Generic GUI Testing

ID	Purpose of the Test	Pre-condition	Test Data	Steps	Expected Result	Actual Result
FUN01	Test main frame of the game, Minimize button	1) Game GUI available to user;		1) User clicks the minimize button;	1) Game GUI minimized;	Pass
FUN02	Test main frame of the game, Maximize button	1) Game GUI available to user;		1) User clicks the maximize button; 2) User clicks that same button and brings frame size to its original size;	1) Game GUI engulfs the full view of the user monitor	Pass
FUN03	Test main frame of the game, Restore Down button	1) Game GUI available to user;		1) User clicks the maximize button; 2) User clicks that same button and brings frame size to its original size;	1) After user maximized the frame size, user will click the same button now called Restore Down and Game GUI will resize to its original pixel size;	Pass
FUN04	Test game design, different frame sizes	1) Game GUI available to user; 2) Game frame can be resized by using bottom-right corner click & drag functionality;		1) User clicks and holds bottom-right corner of the game frame; 2) User resizes game frame to various size options;	1) Design and order of frame components resizes properly, maintaining the components' position and size in relation to frame's width and height;	Pass
FUN05	Test JOptionPane, Cancel button	1) Game GUI available to user; 2) Start button at main game frame fires JOptionPane;		1) User clicks the Cancel button;	1) After user clicks Cancel button, JOptionPane is removed from the view; 2) User is unable to select first move nor will the first move be selected by Computer player;	Pass

ID	Purpose of the Test	Pre-condition	Test Data	Steps	Expected Result	Actual Result
FUN06	Test JOptionPane, human Player pick plus OK button	1) Game GUI available to user; 2) Start button at main game frame fires JOptionPane;		1) User clicks the Player option in the dropdown menu; 2) User selects OK button;	1) JOptionPane is removed from the GUI view after OK button is fired; 2) Functionalities at main frame visible and accessible; 3) Human player starts the game so board remains empty until user selects first option in the game;	Pass
FUN07	Test JOptionPane, Computer player pick plus OK button	1) Game GUI available to user; 2) Start button at main game frame fires JOptionPane;		1) User clicks the Computer option in the dropdown menu; 2) User selects OK button;	1) JOptionPane is removed from the GUI view after OK button is fired; 2) Functionalities at main frame visible and accessible; 3) Computer player starts the game so the board is automatically populated with the first game move by Computer player;	Pass
FUN08	Test selection of already-selected game field	1) Game GUI available to user; 2) Game started and waiting for input;		1) User selects the game choice already selected by either Human or Computer player;	1) Selection of already-populated game field fires warning message that user selected the invalid choice; 2) While JOptionPane warning is displayed, all functionalities in the game GUI are unabled;	Pass
FUN09	Test Start button	1) Game GUI available to user;		1) User clicks the Start button;	1) Clicking the Start button, user fires the JOptionPane that offers the user the choice of who starts as first player in the game;	Pass

ID	Purpose of the Test	Pre-condition	Test Data	Steps	Expected Result	Actual Result
FUN10	Test New Game button	1) Game GUI available to user; 2) Game is in play mode;		1) User clicks the New Game button;	1) Clicking the New Game button, user fires the JOptionPane that asks the user if he / she wants the game result to be refreshed; 2) User opts for Cancel button will bring the game to status identical to before step 1; 3) User opts for OK button refreshes the result data and forces the user to start the new game via Start button;	Pass

3) Unit Testing

ID	Point to @Test	Purpose of the Test	Pre-condition	Test Data	Expected Result
UNI001_01	public void testResult_1() throws Exception	Test the instantiation of constructor for the class Result			Create class instance (object of the class Result)
UNI001_02	public void testClearScore_1() throws Exception	Run the void clearScore() method test	Object of the class Result instantiated	Game number data / instance variable games	Instance instance variables comWinNum, humanWinNum and games set to 0;
UNI001_03	public void testComWin_1() throws Exception	Run the void comWin() method test	Object of the class Result instantiated	Computer win data / instance variable comWinNumber	Data incremented by one after each method call
UNI001_04	public void testGamesPlus_1() throws Exception	Run the void gamesPlus() method test	Object of the class Result instantiated	Game number data / instance variable games	Data incremented by one after each method call
UNI001_05	public void testGetComWinNum_1() throws Exception	Run the int getComWinNum() method test	Object of the class Result instantiated	Computer win data / instance variable comWinNumber	Data incremented by one after each method call
UNI001_06	public void testGetGameTimes_1() throws Exception	Run the int getGameTimes() method test	Object of the class Result instantiated	Game number data / instance variable games	Return number of games equal to number of times method gamesPlus() was called
UNI001_07	public void testGetHumanWinNum_1() throws Exception	Run the int getHumanWinNum() method test	Object of the class Result instantiated	Human win data / instance variable humanWinNum	Return number of human wins equal to number of times method humanWin() was called
UNI001_08	public void testHumanWin_1() throws Exception	Run the void humanWin() method test	Object of the class Result instantiated	Human win data / instance variable humanWinNum	Data incremented by one after each method call

ID	Point to @Test	Purpose of the Test	Pre-condition	Test Data	Expected Result
UNI002_01	public void testBoard_1()throws Exception	Run the Board() constructor test			Create class instance (object of the class Board
UNI002_02	public void testCheckMove_1() throws Exception	Run the boolean checkMove(int) method test	Object of the class Board instantiated	Lower boundary value / -1	checkMove() method should return boolean false
UNI002_03	public void testCheckMove_2() throws Exception	Run the boolean checkMove(int) method test	Object of the class Board instantiated	Top boundary value / 9	checkMove() method should return boolean false
UNI002_04	public void testCheckMove_3() throws Exception	Run the boolean checkMove(int) method test	Object of the class Board instantiated	Lower boundary value / 0	checkMove() method should return boolean true
UNI002_05	public void testCheckMove_4() throws Exception	Run the boolean checkMove(int) method test	Object of the class Board instantiated	Top boundary value / 8	checkMove() method should return boolean true
UNI002_06	public void testGetSquares_1() throws Exception	Run the char[][] getSquares() method test	Object of the class Board instantiated	Values for char[][] result	2D Arrays (3 x 3) instantiated with no data assigned
UNI002_07	public void testGetWinner_1() throws Exception	Run the char getWinner() method test	Object of the class Board instantiated	Return value from method getWinner()	getWinner() method should return char '-'
UNI002_08	public void testGetWinner_2() throws Exception	Run the char getWinner() method test	Object of the class Board instantiated	Return value from method getWinner()	getWinner() method should return char 'X'
UNI002_09	public void testIsFull_1() throws Exception	Run the boolean isFull() method test	Object of the class Board instantiated	Return value from method isFull()	isFull() method should return boolean false
UNI002_10	public void testIsFull_2() throws Exception	Run the boolean isFull() method test	Object of the class Board instantiated	Return value from method isFull()	isFull() method should return boolean true
UNI002_11	public void testReset_1() throws Exception	Run the void reset() method test	Object of the class Board instantiated	Return value from method getSquares()	reset() method should clear all data assigned by method setMove()
UNI002_12	public void testSetMove_1() throws Exception	Run the void setMove(char,int) method test	Object of the class Board instantiated	Value and location set by method setMove()	char 'X' correctly set to the board location determined by int value

ID	Point to @Test	Purpose of the Test	Pre-condition	Test Data	Expected Result
UNI003_01	public void testMain_1() throws Exception	Run the void main(String[]) method test			Game GUI and functionality run

ID	Point to @Test	Purpose of the Test	Pre-condition	Test Data	Expected Result
UNI004_01	public void testSmartComputer_1() throws Exception	Run the SmartComputer(char) constructor test			Create class instance (object of the class SmartComputer)
UNI004_02	public void checkMove_1() throws Exception	Run the boolean checkMove(int num, char squares[][]) method test	Object of the class SmartComputer instantiated	Lower boundary value / -1	checkMove() method should return boolean false

ID	Point to @Test	Purpose of the Test	Pre-condition	Test Data	Expected Result
UNI004_01	public void testNaiveComputer_1() throws Exception	Run the NaiveComputer(char) constructor test			Create class instance (object of the class NaiveComputer)