CMP1127M Assessment 3: Testing Report

There are five main user requirements that had to be considered during black-box testing, and testing the program against each of these in a structured, systematic way ruled out the possibility of the program not behaving in the manner the user does not expect. However, there is some functionality that the program needs that is not covered in the application specification. This mostly includes error handling and the visualisation of the game. The latter of these was tested alongside the user requirements, as only an end-user can determine if the console readout is easy to understand and logical. To test error handling in the context of menu navigation, boundary value analysis had to be utilised. For this, tests must be designed to "exercise values that lie at the boundaries of an input equivalent class and for situations just beyond the ends" (Wenting Duan, 2017). This was done using both valid and invalid input data types.

White-box tests mostly consisted of multiple condition coverage. This is defined as writing “test cases to exercise all possible combinations of True and False outcomes of conditions within a decision” (Wenting Duan, 2017), and allowed for the number of individual tests to be reduced. The output of some methods did not depend upon the value of variables passed to or declared within a method. In such instances, the functionality of a method was tested by returning a value from the method, or by implementing the method in such a way that its effectiveness could be determined by the behaviour of the program as a whole.

Black-Box Testing: Test Description	Input Data	Expected Result	Actual Result
Test that the user can choose between ‘match play’ mode and ‘score play’ mode	• Option ‘1’ selected to enable ‘match play’ mode. • Option ‘2’ selected to enable ‘score play’ mode	When option 1 is selected, a match play version of the game will be initialised, where the first player to get 5 points wins the game. When option 2 is selected, a score play version of the game will start	As expected. Test passed
Test that three dice are rolled at the start of a player’s turn, with the number decreasing by on each time	• ‘Match play’ mode selected. • ‘Score play mode selected’	There dice are rolled at the start of a player’s turn, followed by two being rolled, and a single die being rolled in the final roll	As expected. Test passed
Test that a player’s score is calculated correctly, using the highest score after each dice roll	‘Score play’ selected.	The highest score after each set of dice rolls is used to calculate the total at the end of each player’s turn	As expected. Test passed
Test that the winner of the game is calculated in the correct way	• ’Match play’ mode selected. • ‘Score play’ mode selected	When ‘Match play’ mode is selected, the game ends when a player reaches a score of 5. When ‘Score play’ mode is selected, the winner has the highest cumulative score after 5 rounds	As expected. Test passed
Test that the player has a choice of playing against a second player, or a computerised opponent	• ‘Two-player game’ selected. • ‘Play computer’ selected	When a two-player game is selected, the program asks for to names from the user, before setting up a two-player game. When a game against the computer is selected, the second player’s name is set to “computer”	As expected. Test passed
Test that the program can handle a user choosing an invalid menu option when asked which type of game to play, and whether they want to pay the computer or not	• ’abcd’ selected on the ‘select game’ menu • ‘1’ selected on the ‘select opponent’ menu	The program will print ‘That is not a valid input. Try again.’ and prompt the user for another input	Failed. When a string is entered at any point, rather than a character, the program crashes. Test passed with other character inputs. Program was corrected to rectify this error
Test that information about each player is displayed correctly and at relevant points throughout the execution of the program	• ‘n’ selected when deciding whether to play computer • ‘Jack’ as player 1’s name • ‘Jeff’ as player 2’s name	At the start of a player’s turn, their name is clearly displayed. At the end of a round, each player’s score is displayed	As expected. Test passed

White-Box Testing: Function Tested	Variable Values	Expected Result	Actual Result
SelectGameType()	• gameChoice set to integers ‘1’ and ‘2’ • gameChoice set to character ‘a’ • gameChoice set to string ‘abcd’	• ‘1’ and ‘2’: SelectOpponentType() called • ‘a’: console output “That is not a valid choice. Try again” • ‘abcd’: console output “That is not a valid choice. Try again.”	As expected. Valid input requested until ‘1’ or ‘2’ is assigned to variable gameChoice. Test passed

SelectOpponentType()	• compChoice==‘y’ and ‘n’ • compChoice==‘1’ • compChoice==‘abcd’	• ‘y’ and ‘n’: SetNames() called • ‘1’: “That is not a valid choice. Try again” printed by console • “abcd”: “That is not a valid choice. Try again” printed by console	As expected. Valid input is requested until the values ‘y’ or ‘n’ are assigned to variable gameChoice. Test passed
SetNames()	• compChoice==‘y’ • compChoice==‘n’ • gameChoice==‘1’ • gameChoice==‘2’	• compChoice==‘y’: p1.GetPlayerName() called and p2.name set to “Computer” • compChoice==‘n’: GetPlayerName() called for both player objects • gameChoice==‘1’: MatchPlay() called • gameChoice==‘2’: ScorePlay() called	As expected. The GetPlayerName() method is correctly called for each opponent type, and correct methods are called for both game types. Test passed
GetPlayerName()	• name==‘Jack Barber’ • name==””	• name==’Jack Barber’: Console outputs ‘Thanks Jack Barber!’ • name==””: Console outputs “A name cannot be blank. Try again”	As expected. Test passed
MatchPlay()	• p1.name==“Jack”, p2.name==“Jeff”	• Whilst p1.score and p2.score are less than 5: PlayGame(), UpdateScore(), PrintEndOfRoundScores() called • GetWinner() and CheckIfPlayAgain() called after loop is stopped	As expected. Test passed
ScorePlay()	• p1.name==“Jack”, p2.name==“Jeff”	• Whilst the ‘round’ variable is less than 5: PlayGame(), UpdateScore(), PrintEndOfRoundScores() called • GetWinner() and CheckIfPlayAgain() called after loop is stopped	As expected. Test passed
PlayGame()	N/A	• During each loop: PrintRoundInfo(), RollMultipleDie(), Player.UpdateRollTotal(), and PrintEndOfRoundScores() called	As expected. Test passed
PrintRoundInfo()	• playerTurn==1, p1.name==’Jack’, round==1 • playerTurn==2, p2.name=’Jeff’, round==2	• Console outputs ‘Round 1. Jack’s turn’ • Console outputs ‘Round 2. Jeff’s turn’	As expected. Test passed. Works for all rounds
UpdateRollTotal	rollScore has value 3, 1, 4 in succession	rollTotal increased to 8	As expected. Test passed
PrintEndOfRoundScores()	• gameChoice==’1’, p1.name==’Jack, p2.name==’Jeff’, p1.score==1, p2.score==0, p1.rollTotal==8, p2.rollTotal==5 • gameChoice==’2’, p1.name==’Jack, p2.name==’Jeff’, p1.score==1, p2.score==0, p1.rollTotal==8, p2.rollTotal==5	• Console outputs ‘Jack's score at the end of the round is: 1’ and ‘Jeff's score at the end of the round is: 0’ • Console outputs ‘Jack's score at the end of the round is: 8’ and ‘Jeff's score at the end of the round is: 5’	As expected. Test passed
RollMultipleDie()	• Highest rolls are 3, 5, 2	• GenerateNumber() called when each dice rolled • Method returns integer 10	As expected. Test passed
UpdateScore()	• p1.currentRollScore==15, p2.currentRollScore==17 • p1.currentRollScore==20, p2.currentRollScore==7	• p2.score incremented by 1 • p1.score incremented by 1	As expected. Test passed
PrintEndOfTurnScores()	• playerTurn==1 • gameChoice==’2’ • p1.name==’Jack’ • p1.currentRollScore==5	• Console outputs ‘Jack’s score increased by 5’	As expected. Test passed. Relevant output also for playerTurn==2 and p2 assigned to Player
GetWinner()	• gameChoice==’1’, p1.score==5, p1.name=’Jack’ • gameChoice==’2’, p1.score==85, p2.score==65 p1.name==’Jack’	• Console outputs ‘Jack wins!’ • Console outputs ‘Jack wins!’	As expected. Test passed • Relevant output for p2.score==5 • Relevant output for p2.score==5
CheckIfPlayAgain()	• playAgain==’y’ • playAgain==’n’	• SelectGameType() called • Environment.Exit() called	As expected. Test passed • New input requested if invalid