readScore() tests

Test 1: Performs the readScore() function when “scores.txt” file does not exist.

Requirements: any existing “scores.txt” file is deleted beforehand.

Result: Message is displayed to show text file did not exist beforehand, a blank text file is then created for later use (confirm test ran correctly by checking for text file).

Test 2: Scores read from existing scores file.

Requirements: Scores text file exists beforehand (will test correctly even if file is not empty).

Result: New score is added to the scores file and then read back into the program. This score is confirmed by the test case itself, however, to be thorough it is also recommended to observe the actual text file for any errors (expected addition will be a player named “correct” with a score of 32).

writeScores() tests

Test 1: New scores written to an empty scores file.

Requirements: None, the test case clears any scores from the text file beforehand.

Result: New score is added to the scores file and then read back into the program for confirmation. This score is confirmed by the test case itself, however, to be thorough it is also recommended to observe the actual text file for any errors (expected addition will be a player named “correct” with a score of 32).

Test 2: New scores written to an existing scores file.

Requirements: Test case will create the text file then add multiple new scores.

Result: Multiple new scores are added to the scores file, it is then confirmed by the test case that the scores are correctly read back into the program from the scores file and are in the correct order. In order to be thorough, it is also recommended to confirm the text file has the correct scores in the correct order (correct player order is “correctfour”, “correcttwo”, then “correctthree”.

checkForTopScore() tests

Test 1: New score is added to an incomplete top ten (adds a single score which is then added into the empty top ten).

Requirements: none

Result: displayTopTen() function is run to display top ten scores, if only a single player named “four” with a score of 4 is on the list, the test ran correctly.

Test 2: New score is added to the middle of the top ten list.

Requirements: none

Result: Three new scores are added (and inevitably added to the incomplete top ten), the lowest score is added first, then the highest, and finally the middle-sized score is added. If this test ran correctly, the resulting top ten list displayed by the displayTopTen() method will have the three scores listed in descending order.

Test 3: New score is added to the end of a complete top ten list.

Requirements: none

Result: Test adds scores to the top ten list until it is full, then adds one more score that is smaller than all other scores in the list. If this test ran correctly, the resulting top ten list output by the displayTopTen() function will have a minimum score of one.

Test 4: New score is added to the beginning of a complete top ten list.

Requirements: none

Result: Test uses the complete top ten list from the last test, then adds one more score that is larger than all other scores in the list. If this test ran correctly, the resulting top ten list output by the displayTopTen() function will have a maximum score of twenty.

addScore() tests

Test 1: New score is added to the scores list.

Requirements: none

Result: Adds multiple scores to the scores list using the addScore() function. Test case then checks that all the scores were added and are in the correct order.

Test 2: New highest score is added to the scores list.

Requirements: none

Result: New highest score is added to the scores list using the addScore() function. Test case then confirms the new score is the head of the scores list (as it should be being the largest score).

Test 3: New lowest score is added to the scores list.

Requirements: none

Result: New lowest score is added to the scores list using the addScore() function. Test case then confirms the new score is the rear of the scores list (as it would be being the smallest score).

reorderScores() tests

Test 1: Scores in scores list are directly backwards

Requirements: none

Result: Test case artificially (since the functions automatically keep scores in order) creates the scores list with the scores in reverse order, then uses the reorderScores() function. The resulting scores list is then confirmed to be in descending order.

Test 2: A single score in the scores list is out of place.

Requirements: none

Result: Test case artificially creates the scores list with a single score out of place, then uses the reorderScores() function. The resulting scores list is then confirmed to be in descending order.

readPuzzles() tests

Test 1: Confirms puzzles are correctly initiated by running performPuzzle() function.

Requirements: none

Result: Test case runs the readPuzzles() function to initialize puzzles, then runs the performPuzzle() function to confirm the puzzles list is correctly initiated.

startUp() tests

Test 1: User does not enter a user name.

Requirements: Tester does not enter a user name when prompted.

Result: Program should prompt the user to enter a valid user name after the tester does not enter a user name (in order to further progress the tester should then enter a valid user name).

mainMenu() tests

Test 1: User enters an invalid menu selection.

Requirements: Tester enters an invalid menu selection when prompted.

Result: Program should prompt the user to enter a valid input (either 1, 2, or 3). Tester should enter an invalid input which should result in the program prompting the user to enter a valid input (tester should then enter a valid input in order to further progress).

inGameMenu() tests

Test 1: User enters an invalid menu selection.

Requirements: Tester enters an invalid menu selection when prompted.

Result: Program should prompt the user to enter a valid input (either 1, 2, 3, 4, or 5). Tester should enter an invalid input which should result in the program prompting the user to enter a valid input (tester should then enter a valid input in order to further progress).

moveCharacter() tests

Test 1: Iterates through moveCharacter() function under the same conditions until user defines the “nothing happens” encounter correctly occurs.

Requirements: Tester will indicate to the program that they wish to re-iterate this test until the “nothing happens” encounter is confirmed to have occurred correctly (tester then inputs the number 1 to define the wish to continue testing).

Results: The “nothing happens” encounter should occur resulting in the player (who will have their statistics continuously reset after each iteration) having their remaining time set to 2 and their remaining steps set to 2 at the end of the turn.

Test 2: Iterates through moveCharacter() function under the same conditions until user defines the “Encounter a puzzle” encounter occurs.

Requirements: Tester will indicate to the program that they wish to re-iterate this test until the “encounter a puzzle” encounter is confirmed to have occurred correctly (tester then inputs the number 1 to define the wish to continue testing).

Results: The “encounter a puzzle” encounter should occur resulting in the player having their statistics change after completing the puzzle. Due to the fact that the changes to statistics vary based on which puzzle occurs (and that the performPuzzle() tests later on will test further) this test can be confirmed to run correctly if the player’s statistics are altered at all and the puzzle occurs in and of itself (initial player stats are 3 in all categories).

Test 3: Iterates through moveCharacter() function under the same conditions until user defines the “Encounter a professor” encounter occurs.

Requirements: Tester will indicate to the program that they wish to re-iterate this test until the “encounter a professor” encounter is confirmed to have occurred correctly (tester then inputs the number 1 to define the wish to continue testing).

Results: The “Encounter a professor” encounter should occur resulting in the player having they time reduced to 1 (from both the occurrence and the time required to move) and possibly their intelligence increased (this increase randomly occurs so it is not really not worthy for testing).

Test 4: Iterates through moveCharacter() function under the same conditions until user defines the “Encounter a graduate student” encounter occurs.

Requirements: Tester will indicate to the program that they wish to re-iterate this test until the “encounter a graduate student” encounter is confirmed to have occurred correctly (tester then inputs the number 1 to define the wish to continue testing).

Results: The “Encounter a graduate student” encounter should occur resulting in the player having their remaining time reduced by a variable amount (this change is random so it may not occur at all) as there is a chance nothing occurs on this encounter, this test is mainly to confirm the encounter occurs at all rather than if it has any effect.

Test 5: Iterates through moveCharacter() function under the same conditions until user defines the “Attacked by grunt work” encounter correctly occurs.

Requirements: Tester will indicate to the program that they wish to re-iterate this test until the “attacked by grunt work” encounter is confirmed to have occurred correctly (tester then inputs the number 1 to define the wish to continue testing).

Results: The “attacked by grunt work” encounter should occur resulting in the player having their remaining time reduced to one (due to the combination of the encounter and the time taken to move the character) and their intelligence reduced to two.

Test 6: Iterates through moveCharacter() function under the same conditions until user defines the “Grade papers” encounter correctly occurs.

Requirements: Tester will indicate to the program that they wish to re-iterate this test until the “grade papers” encounter is confirmed to have occurred correctly (tester then inputs the number 1 to define the wish to continue testing).

Results: The “grade papers” encounter should occur resulting in the player having their remaining time reduced to one (due to the combination of the encounter and the time taken to move the character) and their money increased to four.

performPuzzle() tests

Test 1: Presents a puzzle, enter the correct answer. Will then confirm overall statistics have changed.

Requirements: Tester will enter the correct answer to the puzzle that will be displayed (noting that answers must be all lower case).

Results: Puzzle should be accurately displayed to the user, then after entering the correct answer, the program should display the changes that are made to the character’s statistics as a result and said changes should be made (this amount varies based on the puzzle).

Test 2: Presents a puzzle, enter an incorrect answer. Confirms overall statistics are changed afterwards.

Requirements: Tester will enter an incorrect answer to the puzzle that will be displayed.

Results: Puzzle should be accurately displayed to the user, then after entering an incorrect answer, the program should display the changes that are made to the character's statistics as a result and said changes should be made (this amount varies based on the puzzle, however, it can at least be confirmed that answering a puzzle incorrectly has no positive effects).

gameOver() tests

Test 1: Player's intelligence reached 0.

Requirements: none

Results: Output should display a message informing the tester that their remaining intelligence has reached 0 resulting in their death. The test case will then confirm that the player score in such a scenario is zero indicating the game was not completed.

Test 2: Player's money reached 0.

Requirements: none

Results: Output should display a message informing the tester that their remaining money has reached 0 resulting in their death. The test case will then confirm that the player score in such a scenario is zero indicating the game was not completed.

Test 3: Player's time reached 0.

Requirements: none

Results: Output should display a message informing the tester that their remaining time has reached 0 resulting in their death. The test case will then confirm that the player score in such a scenario is zero indicating the game was not completed.

Test 4: Player successfully completed the game.

Requirements: none

Results: Output should display a message informing the tester that they have completed the game followed by their earned score. The test case will then confirm that the score associated to the resulting player object is correct.