Task Sheet 2 - Game Improvements Introduction

This series of tasks focuses on improving the game so that it is possible to persist data between runs of the program and beginning the process of adding flexibility and customisation to the structure of the game.

Task 6 - Ace High or Low?

In most card games it is possible to decide whether you want the ace to be considered the lowest or highest value in the deck. Currently the program always considers the ace to be low. This needs to be changed to allow the user to make the decision about the ace's value.

It should work as follows:

The user selects the **options** choice from the main menu

They are then presented with the options menu

Having selected an option to modify, the user can then change the default

The main menu is then redisplayed

Questions

Answer the following questions:

Where would be a good place in the program to keep track of whether the ace is high or not? Remember that the aces' value will be set before a game is started and it will not change for the duration of the game. -As a gloabal variable, so it can be used where needed without feeding it in and out.

Name the function that will need to be altered so that 5. Options is part of the main menu. DisplayMenu. Name the function that will need to be modified so that card value comparisons will work correctly now that aces can be either high or low. -IsNextCardHigher

You will need four additional functions to make this improvement possible:

DisplayOptions() - presents the options menu GetOptionChoice() - gets the options choice from the user SetOptions(OptionChoice) - decides which option is to be modified based on the user's choice SetAceHighOrLow() - asks the user to decide whether the ace should be considered high or low and then sets this value Remember that validation should be included where necessary.

Write pseudo-code for each of the four functions identified above. Program code

def DisplayOptions(): OUTPUT'OPTION MENU' OUTPUT'1. Set Ace to be HIGH or LOW' OUTPUT'Select an option from the menu (or enter q to quit): ', end=''

def GetOptionChoice(): boolean -> False while not boolean: option -> input("") if option = "1" or option "q": boolean -> True else: OUTPUT"Please enter a

valid option" END WHILE return option.lower()[0]

def SetOptions(OptionChoice): IF OptionChoice = '1': SetAceHighOrLow() END IF

def SetAceHighOrLow(): global ACE_HIGH boolean -> False while not boolean: Ace-> input("Do you want the Ace to be (h)igh or (l)ow: ") Ace -> Ace.lower()[0] if Ace = "h" or Ace = "l": boolean -> True END WHILE

Make the following changes to the code in the program:

Amend the function you identified in Question 2 so that 5. Options is part of the main menu. Convert the pseudo-code you have written to Python code. Improve the function you identified in Question 3 so that comparisons work correctly. Testing

In the previous section you made significant changes to the program. These changes must be tested to ensure that the program functions correctly. Using the headings given below to help you, write a test plan for your changes.

Test Number	Test Description	Test Data Type	Expected Result	Actual Result
1	Main menu	5	Move to options	Went to options
2	Option Menu	1	Ask for high or low	Asked for high or low
3	High or low	h	Set ace as high and return to main menu	Made ace high and returned to main menu.

Remember to provide tests for normal, erroneous and boundary data.

Task 7 - Sorting scores (from highest to lowest)

Normally score tables for games are sorted - the highest score at the top and the lowest at the bottom. Current the recent scores table in the game is unsorted.

Pseudo-code

To sort the score table you will need to create a new function called BubbleSortScores(RecentScores).

Write the pseudo-code for this function, ensuring that the scores will be displayed from highest to lowest.

def BubbleSortScores(RecentScores): noMoreSwaps -> True listLen -> len(RecentScores) while noMoreSwaps = True: listLen -> listLen - 1 noMoreSwaps -> False for place in range(1,listLen-1): if RecentScores[place].Score < RecentScores[place+1].Score THEN temp -> RecentScores[place+1] RecentScores[place+1] -> RecentScores[place]

RecentScores[place] -> temp noMoreSwaps -> True

The BubbleSortScores(RecentScores) function should be called before the score table is displayed on screen. Implement the pseudo-code algorithm you have written and then call it in the correct place to sort the scores.

Task 8 - Saving scores to a text file

Currently recent scores are lost when the program is quit. It would be better if the program saved the most recent scores to a file so that they can be recalled later. This would mean that when you get a score you would have to beat the highest all time score and not just the highest score since the last time the program was run.

Program code

Make the following changes to the program, keeping in mind that you are dealing with text files:

Amend DisplayMenu() so that there is an additional option to save the scores. Write a new function called SaveScores(RecentScores) that will save the scores to a text file called save_scores.txt. Amend the main program so that SaveScores(RecentScores) can be called successfully. Task 9 - Loading scores from a text file

Having saved the scores to a file in the last task we need to enable the program to load in these scores automatically when the program runs.

Program code

Make the following changes to the program:

Write a new function called LoadScores() that will read in the scores from the text file and reconstructed the RecentScores list of TRecentScore() records. Improve the main program so that the RecentScores are loaded from the file but if the file is empty the program will still append the three blank scores that it currently does. Task 10a - Missing save file

Having added the code to load the file there is a problem - if this file doesn't exist it will crash the program!

Program code

Make the following changes to the program:

Improve the loading of the file so that it can deal with the exception of the save_scores.txt file being missing. Task 10b - More scores

The constant NO_OF_RECENT_SCORES provides the program with the number of scores that should be present in the high score list. Obviously it is possible to change this value between runs of the program.

Questions

Answer the following questions:

Explain why changing the constant NO_OF_RECENT_SCORES could cause a problem if there are already scores stored in the text file save_scores.txt. Program Code xt Make the following changes to the program:

Change the constant NO_OF_RECENT_SCORES so that 10 recent scores can be displayed. Make any necessary improvements to the program so that the issue identified in Question 1 is resolved. Next

The next set of tasks will continue to focus on ways to improve the game.