**Brief description of aims of program**

The program I plan on creating is a Spelling game. The program will play words of similar difficulty out loud for the user to enter in the quickest time possible. Once the user has finished entering the words, their score will be calculated using the time they took and the number of words they got correct. This score will then become part of the high scores list if it was good enough.

**Analysis of requirements of the program**

The spelling game should begin at a starting menu which is visible on a Graphical User Interface. This menu should allow the user to either register a new user, log in with an existing username or visit the high scores menu by clicking buttons on the Graphical User Interface. There should be 3 text files to go along with the program. One to store usernames, one to store high scores and one to store the list of words for the program. As well as these text files, there should be sound files of the words within the words text file to allow the words to be played by the program.

If the user chooses to visit the high scores table, the Graphical User Interface should change to show a table containing the top 10 high scores. These high scores should be calculated by ordering all of the scores in the high scores file by their score with the highest first. Once the scores are sorted, the first 10 scores should be displayed within the high scores table for users to look at.

If the user chooses to register a new user, they should be able to enter their username of choice into an entry box and click a button to confirm it. If the user didn’t enter anything into the entry box, the program should display a message informing the user that they must enter a username. If the user entered a username which already exists in the usernames text file, the program should display a message informing the user that the username they entered already exists meaning it cannot be registered again. Finally, if the user enters a username which doesn’t currently exist within the usernames file, the username should be appended to the usernames file and the program should display a message informing the user that their username was successfully saved.

If the user chooses to log in as an existing user and begin the game, they should be able to enter their username into an entry box and click a button to log in and start the game. If the user didn’t enter anything into the entry box, the program should display a message informing the user that they must enter a username if they wish to log in. If the user entered a username which isn’t currently in the usernames file, the program should display a message informing the user that the username they entered doesn’t exist. However, if the user entered a username which is currently in the usernames file, the program should display a message informing the user that they successfully logged in and that the game will now begin, then the game should begin.

Before the game begins, the program should pick 5 words at random from a list of 10 within the words file. When the game begins, the Graphical User Interface should display a button which can be clicked to play a sound file of the word that they must enter out loud, an entry box to enter their answer and a button to confirm their answer and move on to the next word. The user should gain 50 points for each word they answer correctly. Whilst the user is entering their answers, a timer should be displayed informing the user how long they have taken. Once all 5 words have been attempted, the Graphical User Interface should change to an end screen where the user’s score is calculated by subtracting the time the user took in seconds away from their score, for example: if the user got all 5 words correct but took 20 seconds to answer the words, they would finish with a score of (50x5) - 20 = 230. The user’s score should then be appended to the high scores file and the score they achieved should be displayed on the Graphical User Interface.

**Design**

High scores

Create high scores list

Open high scores file in read binary mode

Set contents of high scores list to the contents of the high scores file

Sort high scores list in descending order of scores

Set top 10 high scores as the text of the high scores label

Registering a user

Open usernames file in read mode

Get contents of username entry

If the user entered nothing

Show error telling user to enter a username

Elif the username entered is in the usernames file

Show error telling user that the username already exists

Else

Open usernames file in append mode

Write the username entered into usernames file followed by a new line

Close the usernames file

Show a message telling the user that their username has been saved

Logging in

Open usernames file in read mode

Get contents of username entry

If the user entered nothing

Show error telling user to enter a username

Elif the username entered is not in the usernames file

Show error telling the user that the username doesn’t exist

Else

Show message telling the user they have logged in and the game will start

Set user to the username entered

Call the load question function

Choosing words at random

Open words file in read mode

Split each line of the word file and create a list containing each of the lines

Set number of words to be used in the game

Create a list that will store the words

Create a list that will store the file names of the sounds

For the number in the number of words variable

Set word to a random choice of the words in the lines of words file list

Remove the word randomly chosen from the lines of word file list

Append the word to the words list

Append the word to the sounds list followed by .wav to create the file name

Play words

Initiate pygame

Set sound to the item in the list that is in the position of the current question – 1

Create the pygame display

Play the sound

Get the program to sleep for 2 seconds to allow the sound to play

Stop the sound

Quit pygame display

Confirm answer

Set answer to the contents of the word entry box

If answer equals the item in the word list that is in the position of current question – 1

Add 50 to score

If current question equals 5

Subtract sec from score

Call the end screen function

Else

Add 1 to current question

Call the load question function

Write score to text file

Create an empty list to store the scores

Open the high scores file in read binary mode

Load the pickled high scores file into the list which will store the scores

Close the high scores file

Combine the username and score separated by a comma and append it to the high scores list

Open the high scores file in write binary mode

Dump the contents of the high scores list into the high scores file

Close the high scores file

Display a message informing the user that their score has been saved

**Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test # | Test Description | Expected outcome | Actual outcome | Evidence | Evidence of correction |
| 1 | Check that the GUI opens and displays the menu once the program is ran. | For the GUI to open and the menu to be displayed. | The GUI did not open. | - |  |
| 2 | Check that the user can visit the high scores table by clicking the high scores button. | For the high scores table to be displayed. | As expected. |  | - |
| 3 | Check that the user can navigate back to the menu from the high scores table by clicking the back button. | For the menu to be displayed. | As expected. |  | - |
| 4 | Check that the program displays an error if the user doesn’t enter a username and clicks the register button. | For an error informing the user that they must enter a username to be displayed. | The error was not displayed. |  |  |
| 5 | Check that the program displays an error if the user enters a username which already exists and clicks the register button. | For an error informing the user that the username they entered already exists to be displayed. | As expected. |  | - |
| 6 | Check that the program writes the username entered to the usernames file and displays a message if the user enters a username that doesn’t already exists and clicks the register button. | For the username to appended to the usernames file and for a message informing the user that the username has been saved to be displayed. | As expected. |  | - |
| 7 | Check that the program displays an error if the user doesn’t enter a username and clicks the log in button. | For an error informing the user that they must enter a username to be displayed. | As expected. |  | - |
| 8 | Check that the program displays an error if the user enters a username that doesn’t already exists and clicks the log in button. | For an error informing the user that the username they entered doesn’t exist to be displayed. | As expected. |  | - |
| 9 | Check that the program displays a message and starts the game if the user enters a username that exists and clicks the log in button. | For the program to display a message informing the user that they have successfully been logged in and that the game begins. | As expected. |  | - |
| 10 | Check that the GUI displays the timer and that the timer counts. | For the timer to be displayed and for it to count in seconds. | The timer did not start counting. |  |  |
| 11 | Check that the GUI displays a button to play the word to enter, an entry box to enter the word and a button to confirm the answer once the game begins. | For the button to play the word, the entry box to enter the word and the button to confirm the answer to be displayed. | As expected. |  | - |
| 12 | Check that the first word plays out loud once the user is on word 1 and the play word button is clicked. | For the first word to play out loud. | As expected. | - | - |
| 13 | Check that the second word plays out loud once the user is on word 2 and the play word button is clicked. | For the second word to play out loud. | As expected. | - | - |
| 14 | Check that the third word plays out loud once the user is on word 3 and the play word button is clicked. | For the third word to play out loud. | As expected. | - | - |
| 15 | Check that the fourth word plays out loud once the user is on word 4 and the play word button is clicked. | For the fourth word to play out loud. | As expected. | - | - |
| 16 | Check that the fifth word plays out loud once the user is on word 5 and the play word button is clicked. | For the fifth word to play out loud. | As expected. | - | - |
| 17 | Check that the GUI displays the end screen with the user’s score and writes the score to the high scores file. | For the end screen to be displayed with the user’s score and for the score to be written to the high scores file. | The score wasn’t displayed on the end screen. |  |  |
| 18 | Check that the user’s score is correctly calculated. | For the score to be correctly calculated giving 50 points for each correct answer then subtracting the time the user took in seconds. | As expected. |  | - |
| 19 | Check that the user’s score is displayed in the high scores table if it was one of the top 10 scores. | For the user’s score to now be visible in the high scores table if it was good enough. | As expected. |  | - |

Critique

Overall, I believe the program was a success as it does what I first set out for the program to do. The program enables players to create an account and log in using this account to play the spelling game. The program also enables players to compare their scores with others as the top 10 scores are displayed in the high scores table which the player can view at any time. Due to the scores being stored externally from the program itself, the high scores are not lost when the program is closed or restarted.

The program is very easy to use with a Graphical User Interface allowing users to click buttons and type in entry boxes instead of having to type in different commands to use the program. The fact that there is a Graphical User Interface ensures that the program does not look dull, as there is colour, but at the same time does not distract the player from the game itself. The program successfully has randomisation to ensure that each player is getting a fair list of words to have to spell.

The high scores table only shows the top 10 scores by players meaning it is impossible for a player to check if they have improved on their previous scores if the score they achieved was not in the top 10. Therefore, there should be separate high scores tables for each user as well as the top 10 scores.

Some things within the program that could have gone better, for example: anyone can log in to any user’s account by simply typing the username in when logging in to the program. This could result in unfair scores. In addition, the game is short meaning it does not give a great overview of the player’s spelling ability. There was also no variation in the difficulty of words meaning it is impossible to know how well the player can spell words of ranging difficulty.

If I were to create the program again, I would add in multiple levels to the spelling game. This would be a better representation of a spelling bee as they gradually increase in difficulty. I would begin with a level containing very simple words. Then over the course of 5 levels, the words would could slightly more difficult until the final level where the words are challenging to spell. To make the game more interesting, I would create it so that the player has to get a minimum score on each level to progress onto the next level.

I would also like to add an account that a teacher to log in to. This would enable them to set the list of words for the program to choose from. However, this would be difficult due to the program requiring a sound file of each word to be able to play the words aloud.