UNSCRAMBLE EAT

(Programmer's Guide)

Written by:

Marie Francheska Masangkay
Ayla Nikki Lorreen Odasco

Code is divided into 3 modules: main, interface, engine. ‘engine’ contains all codes for checking, picking words, loading dictionary and computing scores, while ‘interface’ contains all codes for the display.(1)When the program starts, player is asked to input the name of the dictionary file. (‘main’ calls the function, ‘seedWords’ in ‘engine’, which prompts for the name of the dictionary file and loads its contents into a list (dictWords).) (2) ‘main’ then calls the function, ‘printInstructions’ in ‘interface’ to print instructions.

The module, ‘main’, has a function called ‘main’, which loops infinitely. It allows the player to choose mode (scrabble or anagram), and play the game over and over.

Function, ‘main’, in module ‘main’

The function calls ‘playGame’ in ‘interface’. The function, ‘playGame’, prints menu, and asks the user to choose a game mode (scrabble or anagram). Depending on the game mode chosen, the function calls ‘anagramWithRetries’ or ‘scrabbleWithRetries’ defined within itself.

Function, ‘anagramWithRetries’, in module, ‘interface

Overview (game):
A game consists of 3 rounds. For each round, player is given a string of letters. This string of letters is a shuffled word from the dictionary file. A player can only move on to the next round if all anagrams of the given shuffled word are guessed. A message saying how many anagrams are left to be guessed is displayed every time the player enters an input to keep track. For each round, player is allowed 3 mistakes. If the word was already guessed, no deductions will apply and the count of mistakes remain the same. Score is based on the scrabble points of the words. A player can attain the hughest possible score if the player enters have guessed all the anagrams of the word for all the rounds.

The program:

(1) The first thing that the function does is initiate a list called ‘userInputList’ which will store all the correct answers that the player has input. This will be helpful later on in computing the scores.

(2) The program then proceeds to pick 3 words from the dictionary using ‘engine.anagram_pickWords()’, since the function is in module, ‘engine’, and then append it to a list called ‘listOfPickedWords’. Inside this function:

(a) to randomly choose a word from the dictionary list (dictWords) the code ‘random.choice()’ was used. This code can only be used when random has been imported first (import random).

(b) Since this game mode is about anagrams, the function was made to choose unique words from the list that are not anagram of the others. This was done by first assigning a variable (aWord) to the randomly chosen word from dictWords. The variable aWord needs to be sorted for anagram checking however, strings cannot be sorted, so another variable has been created (aWordSorted) which will hold the sorted list of aWord. The now list, aWordSorted is then made to be a string using the code: ‘’.join(aWordSorted).

(c) The same thing is done for each elements in the listOfPickedWords. The result is another list (listOfPickedWordsSorted) with sorted words from the listOfPickedWords as elements.

(d) aWord will only be appended to the listOfPickedWords if and only if aWordSorted is not in listOfPickedWordsSorted. This was done using a for loop.

The function ends when the counter(i) for picking words reaches 3 (indicating 3 successful chosen words) and the program will go back to the next line of command in the 'interface' module.

(3) Outer while loop

Since this game mode has 3 rounds, a counter(i) has been setup. When i = 3, the program will stop entering the while loop and continues to compute the score of the player. Inside the while loop:

(a) The variable 'correctCounter' and 'mistakeCounter' have been initiated. These variables are helpful for the gameplay. The player can only enter the next round when the player has guessed all the possible valid (anagram entered is in dictWords) anagram of the given string. To know how many anagrams are left to guess in order to move on, a function 'engine.numberOfAnagrams' have been made. This function is in the 'engine' module.

> *A quick segue: In order to distinguished what word is currently being guessed or being displayed, a variable called 'word' has been made. The variable is assigned to the return value of 'engine.anagram_currentWord(i)' (where i is the index from the 'listOfPickedWords'). This function takes in the index (i) as an argument and chooses the word from the listOfPickedWords based on its position on the list. It returns a word (str type). However, on displaying the word in the game, the word from the 'listOfPickedWords' has to be shuffled. A function have been made for that which is also in the engine module but was called in the interface module, 'engine.anagram_shuffledWord(word)'. This function takes in the argument word (which has been discussed before) and return a shuffled str type. It turns the argment word into a list first, 'listWord' so that it can be shuffled using 'random.shuffle(listWord)', then makes it into a string again using the code: ''.join(listWord) and assigns it to a variable called, 'newWord'. It then returns 'newWord'.*

Continuing on the discussion of the function, 'numberOfAnagrams(word)', the first thing that 'numberOfAnagrams(word)' does is to initiate a list called 'anagrams' and right after, the function clears that list using '.clear()', this code is useful for when the player wants to play a new game so that the previous elements from the past game would not interfere with the current game. The next step would be to initiate a variable called 'noOfAnagrams', this is an int type variable which the function will return. After that a for loop has been using to check for anagrams in dictWords from the word argument that has been discussed earlier. In order to save time, the function only checks the word from the dictWords if the length of the argument word is the same length as the word from dictWords. Then it turn the argument word into a string and sorts it using '.sort()', the ending list is called 'stringListPicked' and the same goes for the word in dictWords (stringListx). The two string are then compared. If they have the same elements, noOfAnagrams increases by 1. A variable 'n' was assigned for the return of the function 'numberOfAnagrams'. In addition, a variable called 'nCounter' have been made that equates to the value of 'n'. This 'nCounter' will be used to display the number of

anagrams left to guess by the player since 'n' must not be mutated because the while loop depends on its value.

(b) When the 'mistakeCounter' reaches 4 in one round, the program then continues to display 'GAME OVER!!!' and computes the score from the 'userInputList'. The process on how this works will be discussed further later on.

(4) Inner while loop:

The program then proceeds to the inner while loop of the function. In this inner while loop, the condition depends on the 'correctCounter' and the 'mistakeCounter' as what have been previously discussed. Inside the loop:

(a) The first thing is to display the shuffled word from earlier by calling the function 'displayWordAnagram(i,word)' within the 'interface' module. This function takes in the counter(i) to signify the round number and the shuffled word from earlier.

(b) It then prompts the player to input their answer using input("Enter answer:"). A variable called 'user_input' has been assigned to this. This variable will be helpful later on in checking if the players input is in the list of anagrams found by the function 'checkAnagrams(word)' in the 'engine' module. Additinally, the program also converts player's answers to lower case letters for easy checking and also, so that the program will accept the player's answers no matter what case it is in.

The program then proceeds to 3 conditions and another 1 in the case of game over.

(5) Conditions

Within the inner while loop is a nest of conditions used for checking the the player's inputs.

(a) The first condition is if the player just presses enter when the program is waiting for a str type as an answer. The display of the game will be replaced by a message saying, 'Not a real word! Please try again.' and then it shows the number of invalid guesses made by the player (NOTE: When this happens, the player's scores will not have any deductions and the number of mistakes would not increase.) and the number of anagrams left to be guessed by using the 'nCounter' variable that was mentioned earlier on. This condition just loops back to the inner while loop until player enters a character.

This condition also has 2 sub-conditions, however, it is simply just to make sure that the number of anagrams left to be guessed are displayed correctly.

(b) The second condition is for checking if the player's input is in the list of anagrams found by the program. This condition calls the function 'engine.checkUserAnswer_Anagram(user_input,word)' from the 'engine' module.

This function first assigns a variable, 'anagrams' to the function 'checkAnagrams(word)' which returns a list of anagrams found in the dictionary based on the argument word.

A quick segue (about checkAnagrams(word)): This function first initializes a list called 'anagrams'. It then clears the list by using '.clear()' which the purpose is already mentioned previously. This function is almost the same as the code and structure of the function 'numberOfAnagrams(word)' except that this function returns a list of anagrams; and instead, of incrementing the 'noOfAnagrams', when the function finds

that 'stringListPicked' is equal to 'stringListx', it appends the x(word) in dictWords to 'anagrams' list.

After generating a list of valid anagrams found in dictWords, the function , 'checkUserAnswer_Anagram(user_input,word)' then continues on to the condition that if the the player's input is found in the list of anagrams of the word, then the function will return true; otherwise, it will return false.

Going back to the second condition in the 'interface' module, if the 'engine.checkUserAnswer_Anagram(user_input,word)' function returns true, the program will proceed to two sub-conditions. The first sub-condition is is the 'user_input' is not already in the 'userInputList' then the player's answer is considered correct. The 'correctCounter' will increase by 1 and the user_input will be appended to userInputList. The nCounter will decreased by 1 and a new variable called, 'nCounterForOthers' is equated to nCounter to update the other messages on how many anagrams are left to be guessed. If the user_input is already in the userInputList then a message saying, "You've already guessed that word." will show up. However, the mistakeCounter will not increased and it will only loop to the inner while loop.

(c) *The third condition is signified by the else statement. This is when the player enters an incorrect word/anagram. The mistakeCounter will increased and a message saying, 'Word not in string.' will show up. The condition, again has two sub-conditions for displaying the correct number of anagrams left to be guessed.*

(d) *The fourth condition is only when mistakeCounter reaches 4. A message saying, "Game Over" will show up and the counter(i) for rounds is equated to 2 so that it will break the inner while loop and the outer while loop to continue to compute the score of the player.*

(6) <u>Computing the scores and the highest score</u>
In order to compute the score that the player attained, 'engine.anagram_computeScore(userInputList)' is called in the 'engine' module.

(a) *The 'anagram_computeScore(userInputList)' function returns an int type. It first initializes the variable scrabblePoints. This function accesses a list containing letters that are grouped based on their scrabble points. Nested for loops were used in the computations of the score. For each element in userInputList, for each character in the element, scrabble points are assigned and at the end of the for loops, the scrabble points are computed.*

Then the highest score is computed by calling the function, 'engine.anagram_highestScore()'.

(b) *The 'anagram_highestScore()' function returns an int type. It first initializes a variable called, 'highestScrabblePoints' and a list called, 'anagramList'. Then it is followed by clearing the list using '.clear()'.This function uses the 'listOfPickedWords' and checks all its anagrams using the function, 'checkAnagrams(word)'. It does this using a for loop. It then appends all the anagrams found using the word to 'anagramList' and then moves on to the next word to append its anagrams. Once the 'anagramList' is completed, it follows the same code and structure from 'anagram_computeScore(userInputList)' except that the argument list is now 'anagramList'. When the player guessed all the anagrams, the score is equal to the highest score and a message saying, "Congratulations!" shows up.*

(7) When the game is finished, the module calls the function 'engine.anagram_clearAll()' which clears all the list that was used by using the '.clear()' so that the past elements of the list would not interfere with the new game.

Function, 'scrabbleWithRetries', in module, 'interface

Overview (game):
A game consists of 2 rounds. For each round, player is given a string of letters. Player is asked to form 3 words from string. Game ends if player inputs 4 invalid responses in a round, or if player manages to finish both rounds. A response is considered invalid if (1) the word is not in string or (2) the word is not in the dictionary. A valid response that has been input twice will not merit points the second time. Score is based on the scrabble points of all the valid answers. For example, if the player entered, 'cats' and 'cats' is a valid response, i.e., the word is in string and the word is in the dictionary, 6 points will be added to the player's total score. (In scrabble, the letter 'c' merits 3 points, while 'a','t' and 's' merit 1 point each.)

The program must therefore:

(1) Produce 2 different strings of letters

*(a) The function, 'playScrabbleWithRetries', in module 'interface' calls 'scrabble_pickWords' in 'engine' to pick 3 words from the dictionary. 'scrabble_pickWords' uses the function, 'choice' in module 'random' to randomly pick a word. The picked words are stored in a temporary list, 'scrabble_wordsToCombine'. To ensure that 3 different words will be chosen, the function first checks if the word is already in list before appending. The function, 'playScrabbleWithRetries', then calls 'scrabble_combineWords' in 'engine'. To produce the shortest string containing all the words, the function first gets all the letters in the words and stores them in a list. (The letter is appended to list only if it is not already there.) The list is then sorted alphabetically. (This is to avoid producing a second string that is merely a permutation of the first string- will be clearer later**.) In order to build the shortest string, the function iterates over the list of letters and finds the highest frequency, 'maxCount', each letter appears, i.e., if the words are 'creed', 'red' and 'bed', then the 'maxCount' of 'e' is 2, since it appears twice in 'creed' and only once in 'red' and 'bed'. The 'maxCount' of a letter is the number of times the letter should appear in the shortest string. The combined words are stored in a list, 'scrabble_combinedWords'. The string is appended to list, only if it is not already there. **The function, 'scrabble_combineWords' returns 'True' to interface if the string produced is already in list. This is to let the interface know if it needs to call 'scrabble_pickWords', 'scrabble_combineWords' and 'scrabble_clearWordsToCombine' (-interface calls this function after calling 'scrabble_combineWords' to clear the temporary list, 'scrabble_wordsToCombine') more than twice. (Obviously, the first time the function, 'scrabble_combineWords' is called, it will return false.)*

(b) After producing 2 different strings of letters, interface calls the function, 'scrabble_shuffleLetters' twice, which accepts 'count', initialized at 0, as a parameter. 'count' is the index (in list 'scrabble_combinedWords') of the string to be shuffled. The function, 'scrabble_shuffleLetters', uses the method 'shuffle' in 'random', so the function, 'scrabble_shuffleLetters' first creates a list, 'lettersToShuffle', containing all the letters in the string. After shuffling, the string is rebuilt and appended to list, 'shuffledLetters'.

(2) Display the word, check, and track the number of invalid answers in a round

(a) In order to display a string of letters, interface calls a function, 'scrabble_getWord' in 'engine' which accepts the index (the variable, 'count_2' initialized at 0) of the string in list 'shuffledLetters'. The function, 'scrabble_getWord' returns the string of letters to interface, enabling 'interface' to display the word. The string of letters is displayed and game prompts for user input while 'countOfCorrect' is less than 3 and 'countOfWrong' is less than 4. If 'countOfCorrect' reaches 3, player proceeds to next round. 'Count_2' updates and the next string of letters is displayed. If 'countOfWrong' reaches 4, game ends and program prints 'GAME OVER!!!'. Note that 'countOfCorrect' and 'countOfWrong' are initialized to 0 at the beginning of each round.

(b) [1] After player inputs a guess, 'interface' calls 'scrabble_checkIfCorrect' in 'engine' which accepts 'user_input' (player's guess) and 'givenString' as parameters. The function, 'scrabble_checkIfCorrect' returns 1 if 'user_input' is correct, 0 if 'user_input' has been previously entered (same round), -1 if 'user_input' is not in string and -2 if 'user_input' is not in the dictionary. The variables, 'countOfCorrect' and 'countOfWrong' in interface update based on the return value of this function. The message displayed after every input is also based on this functon's return value. In order to track player's correct answers for current round, player's correct answers for current round are stored in a list, 'correctAnswersOfPlayerForCurrentRound'. If 'user_input' is in string and 'user_input' is in the dictionary, the function checks if 'user_input' is in list, 'correctAnswersOfPlayerForCurrentRound'. If not, the function appends 'user_input' to 'correctAnswersOfPlayerForCurrentRound' and 'correctAnswersOfPlayer' (-this list is for computing scores later) and returns 1 to interface. Else, if 'user_input' is already in 'correctAnswersOfPlayerForCurrentRound', the function returns 0.

[2] CHECKING IF THE WORD IS IN STRING. The function, 'checkWord', defined in 'scrabble_checkIfCorrect', accepts: 'user_input' and 'givenString' as parameters and returns true to 'scrabble_checkIfCorrect' if 'word' (in '' because 'user_input' may not be a real word) is in string. In order to check if 'user_input' is in 'givenString', the function creates two lists: (1) lettersInInput and (2) lettersInGivenString. For letter in 'lettersInInput', the function checks if count (frequency) of letter in 'lettersInInput' is greater than count (frequency) of letter in 'lettersInGivenString'. If yes, the function, returns false.

[3] CHECKING IF THE WORD IS IN THE DICTIONARY. The function, 'checkInDict' defined in 'scrabble_checkIfCorrect' accepts 'user_input' as a parameter and returns true if word is in 'dictWords' (the list containing all the words in the dictionary).

(c) After every round, 'interface' calls a function in 'engine', 'scrabble_clearCorrectAnswersOfPlayerForCurrentRound' to clear list, 'correctAnswersOfPlayerForCurrentRound'.

(3) Compute the score

After the game ends, 'interface' calls a function in 'engine', 'scrabble_computeScore' to compute the scrabble points of all the words in list, 'corrrectAnswersOfPlayer' and return the total to 'interface'. For every word in list, 'correctAnswersOfPlayer', the function goes over each letter, determines the equivalent scrabble points and adds to total scrabble points earned (variable name: scrabblePoints).

(4) Compute the highest possible score for game

'interface' also calls 'scrabble_highestScore' to compute the highest possible score for game. For each 'stringOfLetters' in 'shuffledLetters' (the list containing the 2 given strings), function goes over the list, 'dictWords' and checks which words are in string ('stringOfLetters'). The words in string are appended to a list, 'wordsInString', initialized as empty at the beginning of the loop (-inside loop, not outside loop). The function then computes the scrabble points of each word in list, 'wordsInString' and stores them in a list, 'pointsForRound'. The list, 'pointsForRound' is sorted to get the highest possible points for round. (Since player is required to input 3 words that can be formed from string in each round, function must total the scrabble points of the 3 words earning the highest scrabble points per round). The highest possible points for round is added to the total, 'highestScrabblePoints', initialized at 0 in the beginning, before any for-loop. The function, 'scrabble_highestScore' returns the computed highest possible score for game to 'interface'.

(5) Print score and highest possible score

'interface' prints player's score, as well as the highest possible score for game.

(6) Clear all lists, except for 'dictWords'

After, printing the score, 'interface' calls a function in 'engine', 'scrabble_clearAll' to clear the lists, 'scrabble_combinedWords', 'shuffledLetters' and 'correctAnswersOfPlayer' to allow the player to play another game.

Sources:

K. Kirk, "Python Clear Screen Function," 2015. [Online]. Available: https://www.youtube.com/watch?v=bguKhMnvmb8. [Accessed 3 November 2018].

"Python String center() Method," tutorialspoint, [Online]. Available: https://www.tutorialspoint.com/python/string_center.htm. [Accessed 4 November 2018].

"Python List clear()," Programiz, [Online]. Available: https://www.programiz.com/python-programming/methods/list/clear. [Accessed 4 November 2018].

"4000 Most Common English Words," World Class Learning, [Online]. Available: https://www.worldclasslearning.com/english/4000-most-common-english-words.html. [Accessed 4 November 2018].