THE SPELLING BEE

by Serdar Biçici

01.06.2023

The Spelling Bee

Serdar Biçici
Artificial Intelligence and Data Engineering Department
Faculty of Computer and Informatics Engineering
İstanbul Technical University
150210331
bicici21@itu.edu.tr

Contents

***Abstract*—The project done by Serdar Biçici is a graphical user interface (GUI) application for the infamous word game by the New York Times, called ”The Spelling Bee”. The game generates a set of random letters and challenges the user to form valid words using those letters. The user can click on letter buttons to form words, and the program checks the validity of the words by comparing them to a list of filtered words. In addition to the original game, the application also provides features such as displaying the meanings of selected words, tracking the user’s progress, giving statistics of the user, and providing various hints. In the project, Python 3.11.3 version was used with additional libraries that are: *tkinter, random, mathplotlib, string, json, BeautifulSoup* and *time*.**

***Index Terms*—word, letter, game, Python**

I. Introduction

The project done by Serdar Biçici is a replica of the word game ”The Spelling Bee” with extra features. The game functions as follows:

- The game generates six random letters and another randomly chosen letter that is identified as the ”middle letter”.
- Later, the game filters all valid words from a .txt file that contains more than 58 thousand words. A valid word is a word that must contain the middle letter and combinations of other chosen letters with a minimum length of three and a maximum length of 20.
- By the user inputs, the game checks if an input is valid or not. If it is valid it displays in the left of the tab in a listBox and can be accessed anytime.
- For every valid input that is in the original list of words, a progress bar at the top of the screen proceeds, and when all words are found by the user the game displays a message that congratulates the user and advice him or her to save the game

The user can click on letter buttons to form words, click ”Enter” to check the validity of the word, click ”Delete” to delete current writing, click ”Save” to save the game with a nickname, click ”Analysis” to get analysis of current game progress with outputs: current score -which is the summation of total words’ length-, current average word length and current words’ found time in seconds graph. In addition to the original game, the application also provides extra features which are: displaying the meanings of words by clicking on them in the listBox, tracking the user’s progress and saving it, giving statistics of the user, giving additional letters in every correct seven words only for three guesses, and providing two types of hints with the first type is for every five correct words and the second type is after finding half of the words a hint about pangram in the case of existence -pangram is a special word that includes all letters in the game-.

II. Methods

A. Filtering the Word List

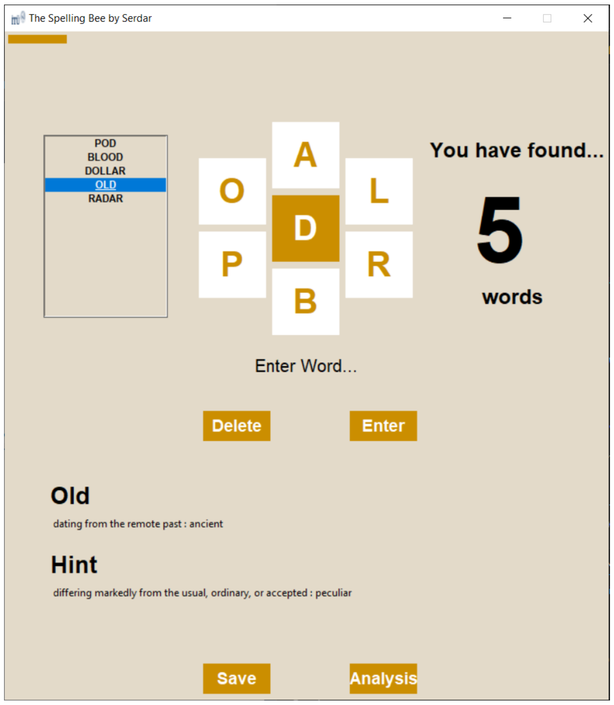
The words for the game are contained in a .txt file and were taken from the GitHub user SCOWL (Spell Checker Oriented Word Lists) and Friends. In total list contained 60388 different words including prefixed forms such as plural and past tense. However, not every word in the file has a dictionary meaning. In order to filter these words the following code was executed.

```python
import requests
from bs4 import BeautifulSoup

def get_dictionary_meanings(word):

invalid = list()
valid = list()

```



Old

dating from the remote past : ancient

Hint

differing markedly from the usual, ordinary, or accepted : peculiar

Fig. 1. Game interface

```python
with open("60kword.txt", "r") as file:
    for item in file:
        try:
            mean = get_dictionary_meanings(item)
            valid.append(mean[0])
            with open("filtered.txt", "a") as new:
                w = str(item).strip().upper()
                new.write(w + "\n")
        except IndexError:
            invalid.append(item)
```

Listing 1. Python word meaning filter

In this code, ”get_dictionary_meanings(word)” is a basic web scraper that gets the dictionary meanings of words from Merriam-Webster online dictionary. Then, the following code uses the function and gets the meaning of the given word. If the meaning list is not empty it can be added to the list for valid words, it opens another .txt file and appends it as uppercase and stripped. In the case of an empty meaning list, the code enters the ”except IndexError” part and therefore, appends it to the invalid list. Both lists are used for easy counting of words with the ”len(list)” function. After filtering, there are a total of 58299 words remained in the game.

B. Getting Random Letters

The letters in the game are generated randomly by the function *random_letters*. However, in the case of not commonly used letters -such as X, Q, and W- being in the game or multiple of them being in the game at the same time, the game becomes unlikely to be playable. In order to deal with these possible cases all words in our filtered word list were entered into an online frequency calculator that returns the frequency of letters. Only the middle letter is not chosen by its frequency to challenge the user more. All the other letters were chosen by their frequency with the following lines of code from *random_letters*:

```python
    other_letters = list()
    for i in range(6):
        l = random.choices(population=alphabet,
    weights=letter_weights, k=1)
        letter_weights.pop(int(alphabet.index(l[0]))
    )
        alphabet.remove(l[0])
        other_letters.append(l[0])
```

Listing 2. Python random letters generator

After this process, the problem of vowel letters raises. In the case of an insufficient amount of vowel letters in the game, the game’s playability decreases exponentially to even not being able to form any words. In order to overcome this problem *random_letters* checks if the game’s vowel count is less than two or higher than four. If this is the case it enters a cycle of recursion until satisfies the conditions.

```python
    vowel_count = 0
    for letter in all_letters:
        if letter in vowels:
            vowel_count += 1

    if vowel_count < 2 or vowel_count > 4:
        return random_letters()
```

Listing 3. Python vowel counter

Furthermore, giving the user an extra letter for every seven correct entries that lasts only three guesses is also done by a similar function that checks if the extra letter is not in the game and returns the extra letter or enters recursion until finding an appropriate letter.

```python
def random_extra_letter():
    alphabet = list(string.ascii_uppercase)

    extra_letter = random.choice(alphabet)

    if extra_letter in all_letters:
        return random_extra_letter()

    return extra_letter
```

Listing 4. Python vowel counter

C. Getting Words for the Game

After completing the process of choosing the letters, the sequential step is getting all possible words from the filtered word list. For being able to achieve that, *word_list_check* function can be used. The function basically reads all words in filtered.txt and filters the words that contain the middle letter. Then, it filters again with the condition of the word containing other letters as well as not containing any extra letters with the following code.

```python
def word_list_check():
    correct_words_list = list()
    global letters
    letters = [l1, l2, l3, l4, l5 ,l6, middle_letter
    ]

    with open("filtered.txt", "r") as file:
        for word in file:
            if str(middle_letter) in word:
                correct_words_list.append(word.strip
    ())

    correct_words_list = [word for word in
    correct_words_list if all(letter in letters for
    letter in word)]

    return correct_words_list
```

Listing 5. Python random letters generator

Nonetheless, filtering words is not done yet. The case of a letter being in the game but cannot form any words also should be considered. In order to deal with these cases following code part will be executed. The function will check the state of every letter being used to create at least one word.

```python
def letter_check():
    invalid_words = list()

    for i in letters:
        for word in valid_word_list:
            if i in word:
                pass
            else:
                invalid_words.append(word)

        if len(invalid_words) == len(valid_word_list
    ):
            return False
        else:
            invalid_words.clear()
    return True
```

Listing 6. Python word letter checker

Next, the code uses a basic *while* loop in order to get valid variables for the game.

```python
l = 0
while l < 10 or l > 50:
    valid_word_list = word_list_check()

    if len(valid_word_list) < 10 or len(
    valid_word_list) > 50:
        middle_letter, l1, l2, l3, l4, l5, l6 =
    random_letters()
        l = len(valid_word_list)
        if letter_check() == False:
            pass
    else:
        break
```

Listing 7. Python word list checker

In this code, ”l” is the length of the valid word list and it is pre-defined as zero to be able to enter the loop. Then, *while* checks if the total valid word count is not between 10 and 50. Inside the loop forms a valid word list and checks its length. If it does not satisfy the conditions, *random_letters* function is once again activated and generates new letters for trying in the next iteration of the loop. Also, the loop uses *letter_check* function as mentioned before.

D. Game Buttons

In order to make the game interactive, some buttons were added to the game with some functionalities. One of the simplest buttons in the game is the ”Delete” button. It simply makes the current word equal to the empty string and changes the label in *tkinter* window with this following code.

```python
def word_delete():
    global current_word
    current_word = ""
    word_label.config(text="Enter Word...")
```

Listing 8. Python delete button function

The letters in the game also appear as individual buttons and they work as follows. First, they check if the length of the word is higher than 20 characters, and if it is higher they pop a new window with a warning message. Secondly, they add the current word and letter with the ”+” operator and update the label with the function named *letter_button*(*word*, *letter*).

```python
def letter_button(word, letter):
    if len(word) > 20:
        word_label.config(text="Enter Word...")

        error_window = Toplevel(root)
        error_window.resizable(False, False)
        error_window.title("Error!")
        error_window.geometry("200x30")
        error_window_label = Label(error_window,
    text="You have exceeded character limit!",
    anchor="center")
        error_window_label.pack()

        return word_delete()

    global current_word

    word = word + letter
    current_word = word
    word_label.config(text=word)
```

Listing 9. Python letter button function

The listBox on the left side of the game window is also interactive. By clicking o desired words, the user can access the dictionary meanings with the help of the following code.

```python
def on_item_selected(*args):
    selected_word = correct_words_listbox.get(
    correct_words_listbox.curselection())
    meaning = get_dictionary_meanings(selected_word)

    word_meaning_header_label.config(text=str(
    selected_word).title())
    word_meaning_label.config(text=str(meaning
    [0][1::]).capitalize(), wraplength=600, justify=
    "left")
```

Listing 10. Python listBox function

E. Giving Hints

One of the additional features of the game is giving hints about the words. For this purpose, the function named *random_hinted_word* simply selects a random word from the valid word list and checks if it is found or not. If it is found it enters recursion, else it returns the word itself with the following code.

```python
def random_hinted_word():
    hinted_word = random.choice(valid_word_list)
    if hinted_word not in words_list:
        return hinted_word
    else:
        return random_hinted_word()
```

Listing 11. Python random word for hint

The second type of hint was about the pangram. In order to determine if the valid word list includes a pangram. The code checks if any word includes all the letters at the same time it returns *True* with the meaning of the word as the hint or it returns *False* with the error message *"This letters do not form any pangram"*.

F. Controlling the Input

Checking if a word is valid is certainly the most important aspect of The Spelling Bee. To achieve correctly checking input and do many afterward operations, *word_check*(*word*) is being run every time the user clicks on *"Enter"* button. The *word_check*(*word*) works as follows:

- In order to manipulate variables, some variables are pre-defined as:

```
    score = 0
    found_word_number = 0
    extra_count = 3
    extra_state = False
    avg_time = list()
    start_time = time.time()
```

Listing 12. Python pre-defined variables

  Here *start_time* starts the timer for measuring the first word guess time and *extra_count* is defined as "3" in order to count down from it and then reset it.
- The first thing when the function runs is checking the length of the input. If it is equal to zero which means it is not a word and the user just clicked the button it returns *None* and finishes execution.
- When the input is a possible valid word that has not been found before, the code uses basic logic statements to identify the word. For this purpose, it uses the following line of code for entering inside the *if* statement and updating variables and the progress bar as well as resetting the timer and checking if the game is half over or not for giving the hint about pangram.

```
    if word in valid_word_list and word not in
    words_list:
```

Listing 13. Python checking valid input

- Another case to mention is the case of words constructed with the given extra letter. In those cases, searching the whole .txt file for words is not efficient, therefore, the code uses the same function for getting dictionary meanings and updating variables. However, not all variables are updated. Words with an extra letter only affect the user's score, not game progress. This means the user can get a higher possible score but not proceed with any progress to finish the game, so the progress bar is not updated. The following line of code represents the required conditions for being a valid word with an extra letter.

```
    elif len(meaning) != 0 and len(word) >= 3
    and word not in words_list and middle_letter
     in word:
```

Listing 14. Python checking valid input with extra letter

- To make the hint appear on the screen as well as give the user an extra letter in mentioned conditions, the function checks if correct input numbers and performs necessary tasks then reset variables if it is necessary.
- Lastly the function checks if the game is finished by comparing the length of the found word list and the valid word list. If it is true, the code displays a new tab with a message. However, if it is not the case, the code uses the *word_delete* function to clear the current word label. Moreover, it is important to note that all the variables that are manipulated in *word_check*(*word*) are declared as *global* which means can be accessible not only in the *def* from everywhere in code and manipulations will affect them anywhere.

```
    global extra_l
    global found_word_number
    global extra_count
    global extra_state
    global hint
    global score
    global start_time
    global end_time
```

Listing 15. Python global variables

G. Analysis Feedback

One of the most important features of the game that differs from the original one is giving feedback about user gameplay. When the *Analysis* button is clicked by the user, *plot_graph* starts to run and returns the score, average found word length, and a graph of word-finding time in seconds that have been visualized by *mathplotlib*.

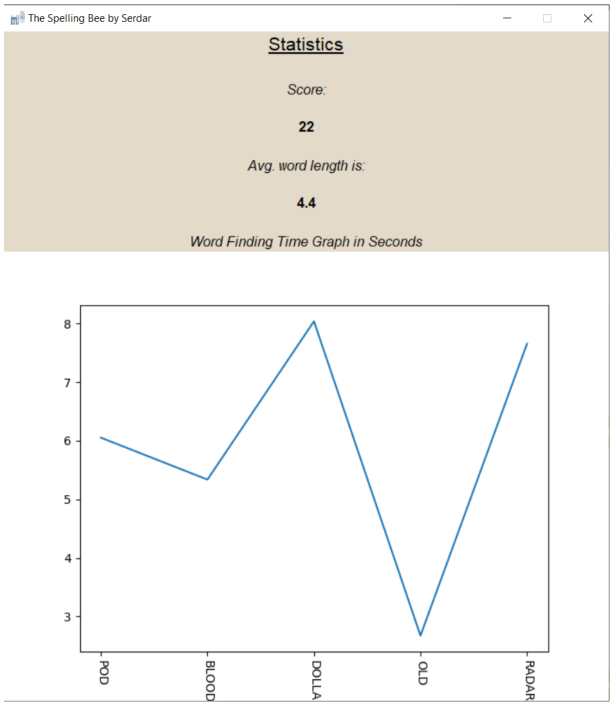


Fig. 2. Statistics window

H. Save Files

The game also includes the feature of saving current game progress that can be shown to the user anytime. However, by saving the game, the user is considered to yield from the game and cannot continue playing with that game progress. The saving process is quite simple and is accomplished by the following lines of code.

```
    def save_username():
    try:
        with open('usernames.json', 'r') as file:
            existing_data = json.load(file)
    except json.decoder.JSONDecodeError:
        existing_data = list()

    user_data = dict()
    username = username_entry.get()

    user_data["username"] = username
    user_data["letters"] = [l1, l2, l3, l4, l5, l6]
    user_data["middle_letter"] = middle_letter
    user_data["word_time_list"] = avg_time
    user_data["found_words"] = words_list
    user_data["all_words"] = valid_word_list
    user_data["score"] = score

    existing_data.append(user_data)

    with open('usernames.json', 'w') as file:
        json.dump(existing_data, file)
```

Listing 16. Python saving game

The *save_username* function basically reads all existing save files in a .json file and stores it in Python format of *list* because if there are any existing save files, this indicates that this function has been executed before and therefore because this function writes in the format of *list*, existing save files also have to be in the format of *list*. Sequentially, the function gets all the desired information of the current game progress and stores them in the format of *dictionary*. Later, the function appends the new save file to the existing save files list and dumps them back to the .json file.

I. Front-End Structures

The front-end structures are default *tkinter* features that have been customized for this specific game. The color palette consists of honey, bone white, white and black. An example button customization and placement can be seen in the following code snippet.

```
button = Button(root,
                activebackground="#cb8e00",
                activeforeground="#FFFFFF",
                background="#FFFFFF",
                foreground="#cb8e00",
                borderwidth="0.1",
                width=3,
                font=("Arial", 30, "bold"),
                command=lambda: letter_button(**
    kwargs),
                text="text")

button.place(x=0, y=0)
```

Listing 17. Python example button

Lastly, in order to run *tkinter* the following one line of code is written.

```
root.mainloop()
```

Listing 18. Python running main loop

III. Discussion

The current state of the project is considerably stable. The user can easily play the game without any major bugs and inconsistencies. The game offers exactly 58229 words with more than 4.5 million possible combinations of letters. The user can access previous game data with the already entered nickname and can easily see statistics of his or her game included with which letters are provided by the game. As mentioned before, one of the additional features of the game was getting dictionary meanings of the words via the internet. Even though this feature seems useful, it makes the game impossible to play without an internet connection and it is important to note that the users should be able to access the internet at all times while playing the game.

IV. Conclusion

To conclude, the project has fulfilled all clauses that were mentioned in the project proposal. The project creates an easily understandable game environment even for first time players. Specific adjustments and extensions were added to the game in order to make the game less challenging for native speakers of English. The possibility to access dictionary meanings of the words anytime boosts learning aspects of the game. Efficient use of *tkinter* features such as comboBoxes and listBoxes makes the game more appealing. Also, data cleaning and preparation for the game have taught Serdar Biçici a lot about web scrapping and data management techniques. The adjustments in the gameplay such as giving hints and showing game progress make the game easier to complete, therefore, giving a more pleasurable experience to the users.

V. Recommendations

As mentioned before, the game has fulfilled desired conditions. However, there is always room for further adjustments and extensions. Some of Serdar Biçici's possible future objectives are itemized below. The new objectives are aforethought based on current gameplay and user feedback.

- Fully accessing previous games. This means the user is able to continue where he or she left off the game and writes on the same save file.
- The constant need for internet access can be challenging for some users. In order to overcome this situation by web scrapping techniques all dictionary meanings may be saved with existing words.
- A scoreboard may be implemented in the game with given letters and scores.
- New game modes may be added such as racing against time, competing with multiple users using the same letter combinations, users choosing their own letters for the game and etc.