Delhi Technological University

Computer Networks Innovative Work

Guess the Longest Word

Submitted by:

Mrigank Badola 2K18/IT/073

Priyank Jain 2K18/IT/088

ACKNOWLEDGMENT

We would first like to thank the almighty God for giving us the power, strength and perseverance to complete this work.

We would also like to express our deep gratitude towards Mr. Jasraj Meena (Assistant Professor) and Mrs. Anamika Chauhan (Assistant Professor), Department of Information Technology, Delhi Technological University for their constant support and guidance, by holding online classes during the ongoing pandemic. This work would not have been possible without their support.

Nonetheless, we would also like to take the opportunity to thank our family members and friends for constantly motivating us to work harder.

INTRODUCTION

In this day and age, the English language is mostly used to communicate with people from other countries. Especially with a lot of MNCs trying to expand their business in developing countries like India, it is essential to have the basics of English language covered.

According to 2001 Census, although India is listed as the second largest English-speaking country (around 12.5 crore people), the numbers only account for merely 12.18% of our total population. It is expected that with increase in literacy rate there would be an increase in the number of people speaking English.

It has also been found that the percentage of people who can speak at least some English is greater in the South than in the North, as some of the schools in North India are Hindi-medium and hence some of the students who pass out in such schools usually take up tuition classes.

Basics of the English language can be broadly divided into two: grammar and vocabulary. While the traditional method is to go through books and getting taught by teachers, our project is aimed at making the experience of increasing the vocabulary a bit more fun, with a game-score concept for our users.

METHODOLOGY

Our project uses the concept of socket programming in order to make server and client communicate and pygame to provide a GUI for our game.

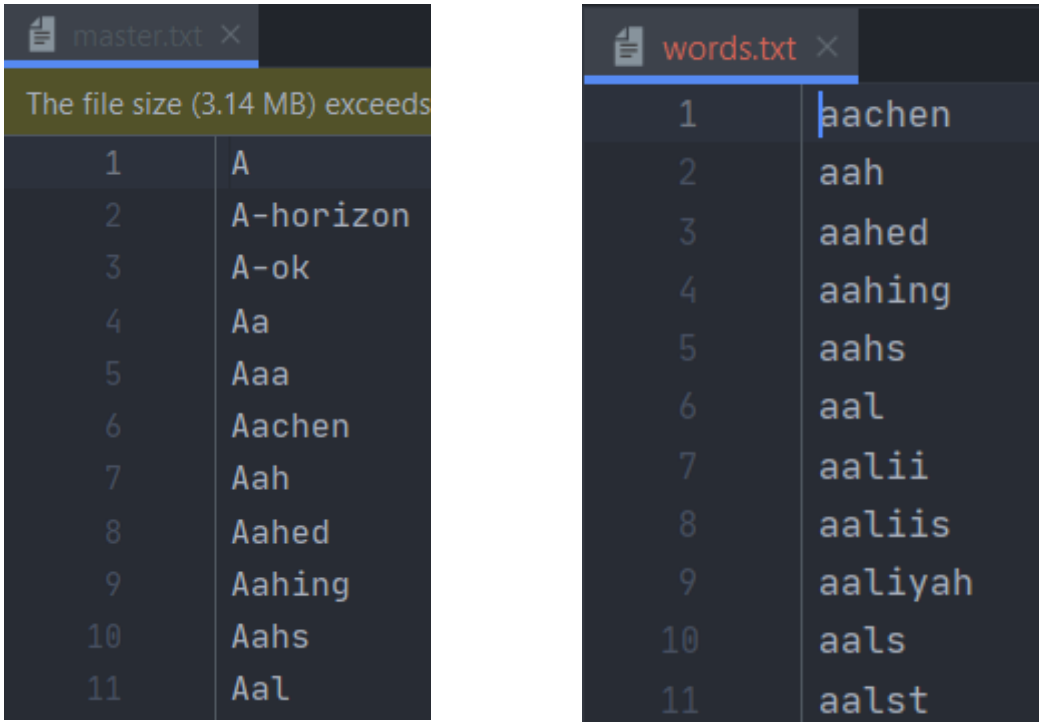
- Our server upon starting, first builds up the dictionary and sub-dictionaries by reading the words from words.txt file. After that it waits for a client's connection request.
- Upon connecting to the client, our server prints out the address and port on which the server-client connection is established, while our client window opens, prompting the user to start playing the game by pressing the *Enter* key (or *Return* key)
- Upon pressing the *Enter* key, the user will be prompted to choose 5 letters, with the freedom of selecting either a vowel or a consonant by pressing '0' or '1' key respectively.
- After entering 5 letters, it is sent to our server, where it tries to figure out what could be the longest word possible. Meanwhile, our user's task would be to make the largest word possible using these 5 letters as many times it wants.
- After entering the word, it is sent to the server where the following checks are made:
    - Is it possible to make a word with the given letters?
    - Is the word entered by user a valid word present in the dictionary?
    - Is the word satisfying the input constraints?
- Appropriate message is then sent from sever to client where it is displayed on the client screen with score awarded with respect to length of word entered and length of longest word.
- After awarding the score, the user is prompted to replay again with *Enter* key, and the game continues with saved score
- At any point, our user can exit the client window by pressing the *Cross* present on the top-right of client window.

ALGORITHMS USED

1. Filtering source dictionary

Input text file: *master.txt,* Output text file: *words.txt*
File *master.txt* is opened in read-only mode. Then, for each *line* present in *master.txt* the following steps are taken:

i. It is lowercased
ii. Then it is checked whether a) Length of set of *line* is between 2 and 5 & b) There are no hyphens present in it
iii. If both the conditions are satisfied, then only it is output to *words.txt*



Comparison after applying algorithm

2. Finding the largest word present in dictionary

Two variables *maxima* (= 0) and *max_word* (= ‘’) are used in this algorithm. A for loop is run for each variable *words* present in the dictionary list.

i. The list of unique letters present in each *words* is stored in a set *S*
ii. Then it is checked whether the set *S* is a subset of set of *letters* chosen by the user.
iii. If it satisfies the condition given above, then it is checked whether the length of current word is greater than *maxima* or not
iv. If the above condition is satisfied, *maxima* is set to length of *words* and *max_word* is set to *words*.

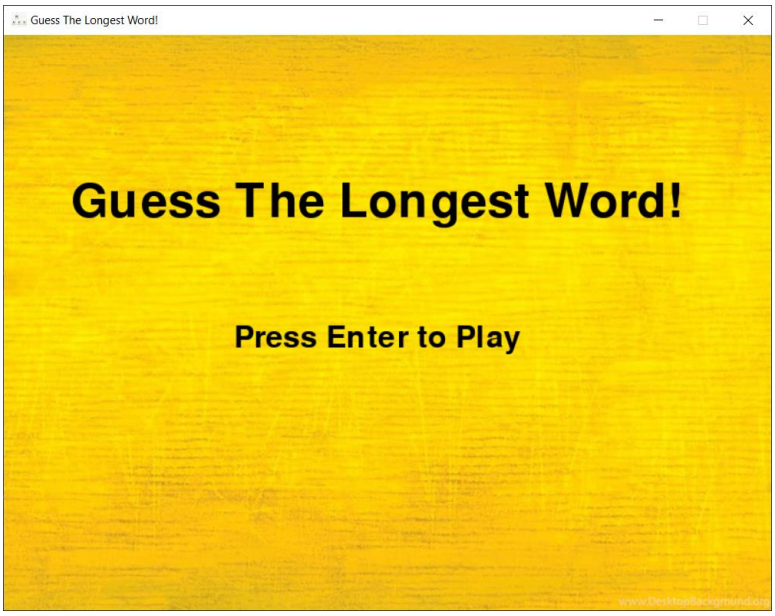
3. Validity of word entered by user

i. First it is checked whether *word* is present in sub-dictionary *word[0] or not*
ii. If the above condition is satisfied, it is checked whether set of *word* is a subset of set of *letters* or not. If yes, then the *word* is valid, else it fails the subset check.

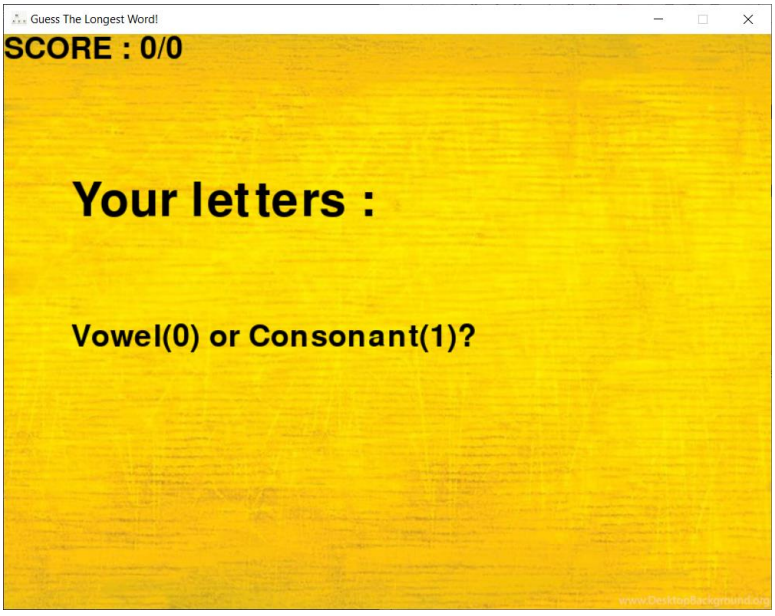
4. Scoring

i. If *max_word* is empty, this implies no word can be made from given *letters*. Hence the scoring this disabled for the current round.
ii. Else if *max_word* is not empty; the *word* is passed through Algorithm 2(i). If it is a negative outcome, then the score is awarded as follows: (0/length of *max_word*). If it is a positive outcome though, then it is passed through Algorithm 2(ii). If this gives a negative outcome, it is awarded 0 again else the score is awarded as follows: (length of *word*/length of *max_word*)

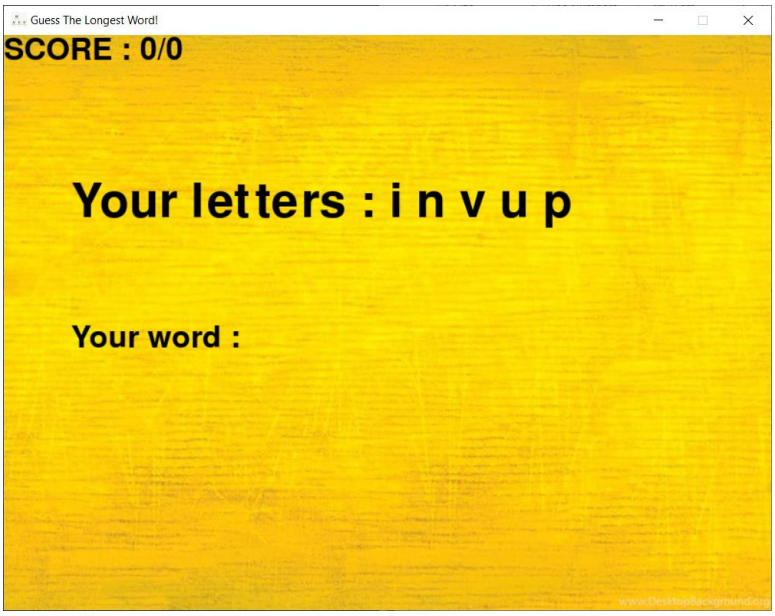
WORKING



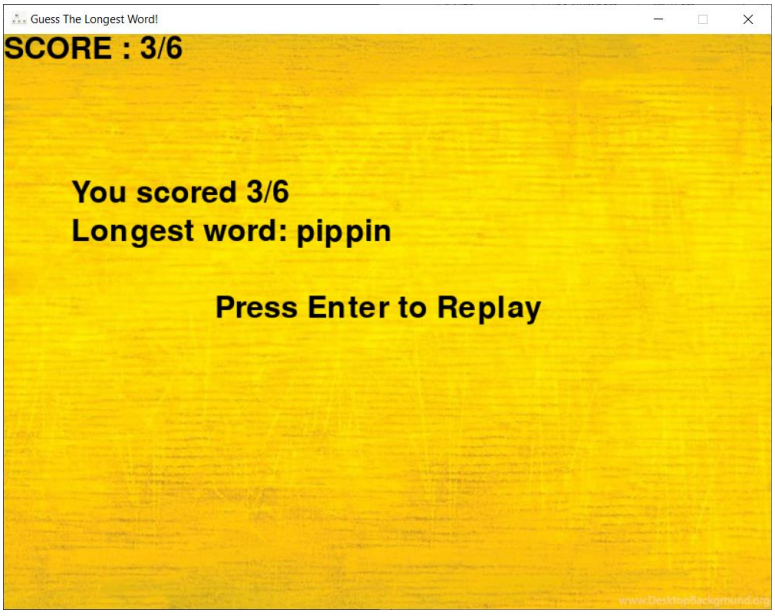
Main Screen



Letters enter prompt

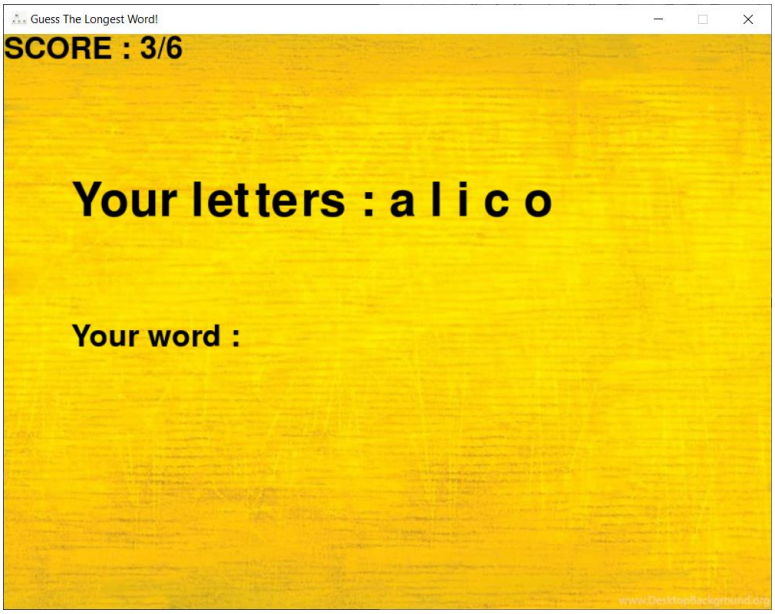


Word enter prompt

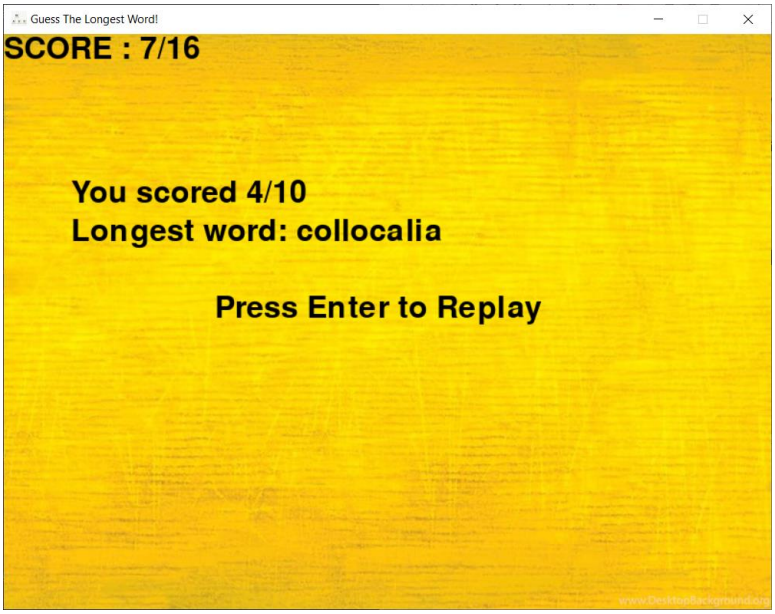


Result screen with word entered as ‘pun’

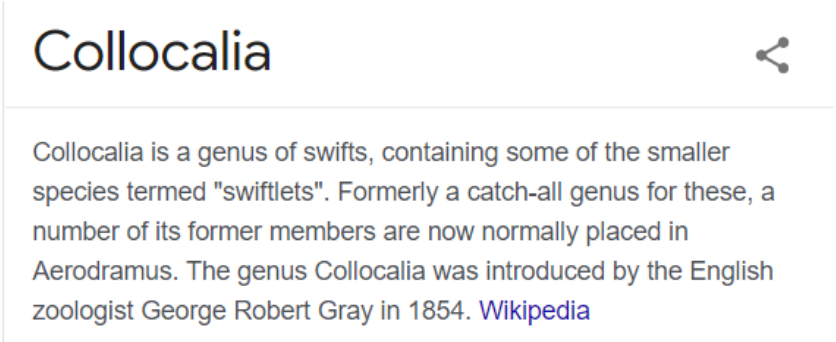
Definition of pippin given by Google



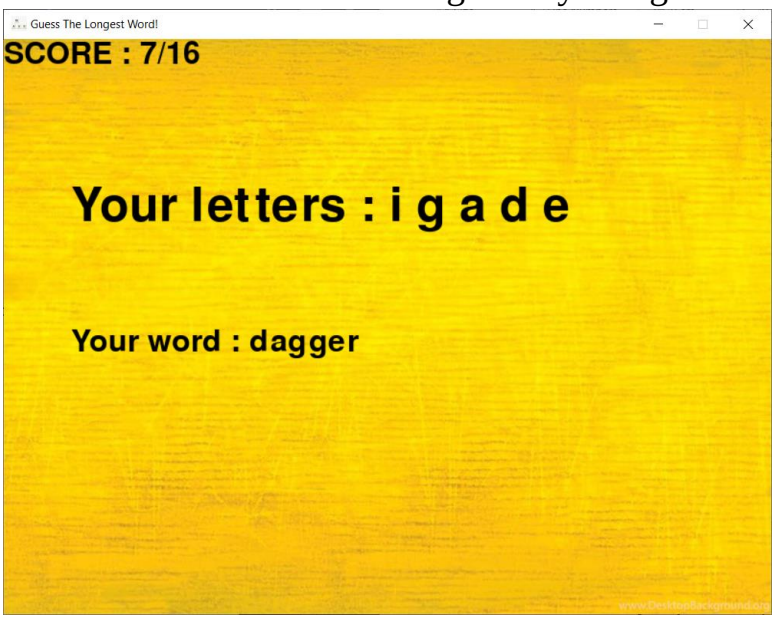
Playing again with retained score



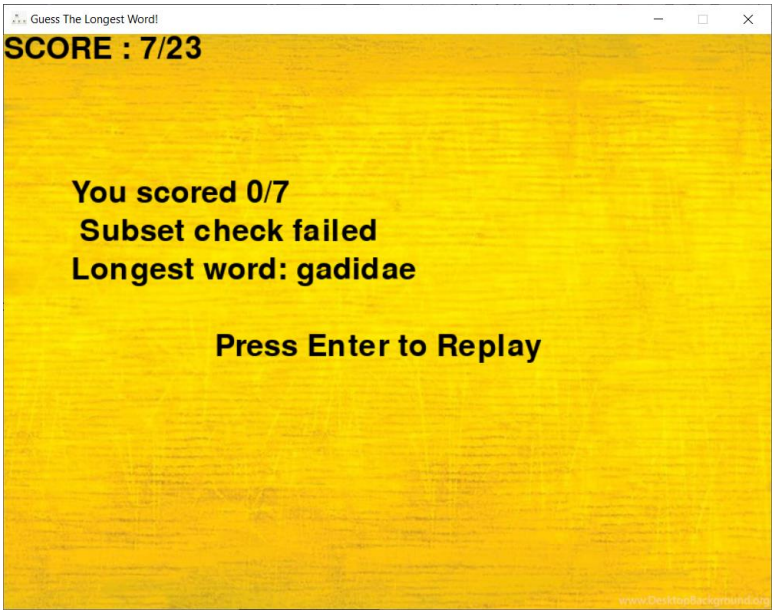
Result screen with updated score with word ‘coal’



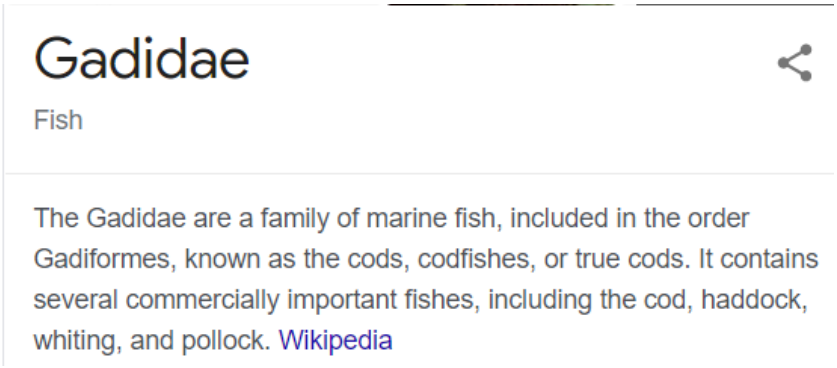
Definition of Collocalia given by Google



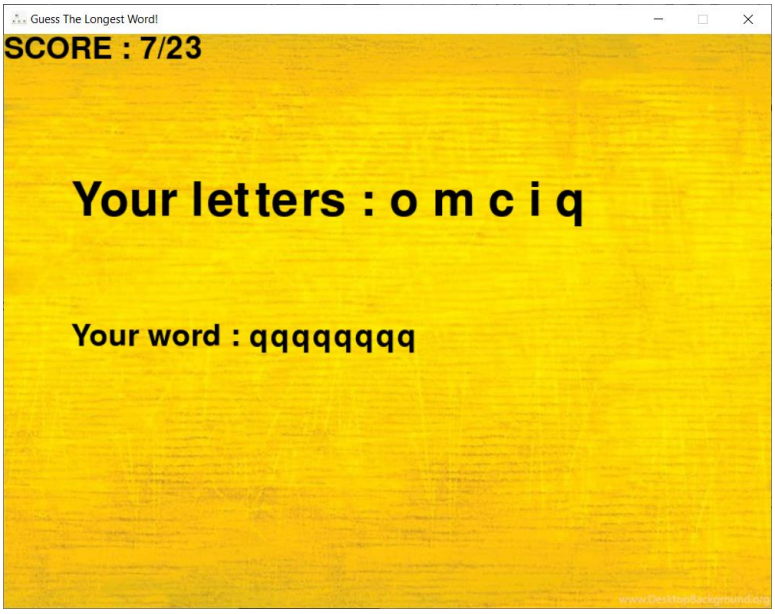
Entering a proper word which fails subset check



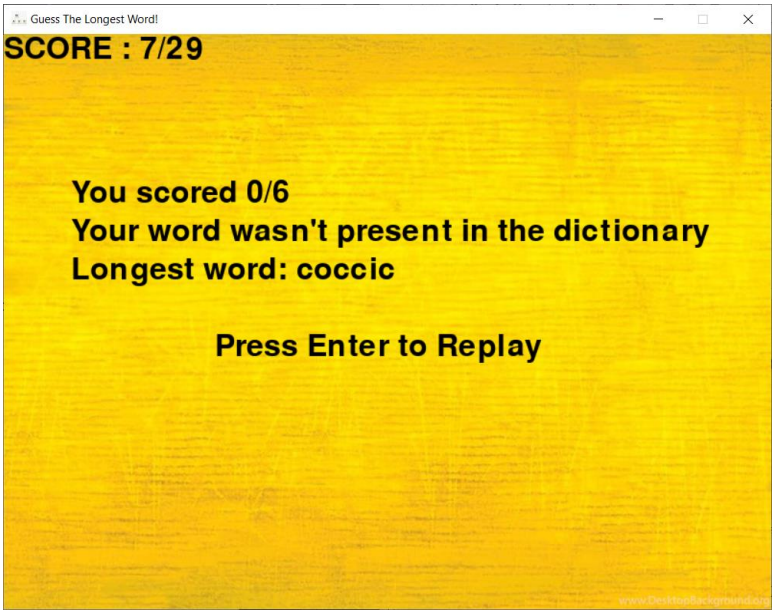
Subsequent result



Definition of Gadidae given by Google



Entering an invalid word



Subsequent result

COCCUS (redirected from *coccic*)
Also found in: **Dictionary**, **Thesaurus**, **Encyclopedia**.

COCCUS [kok´us] (pl. *coc´ci*) (*L.*)

a spherical bacterium, usually slightly less than 1 µ in diameter, belonging to the Micrococcaceae family. It is one of the three basic forms of bacteria, the other two being **BACILLUS** (rod-shaped) and **SPIRILLUM** (spiral-shaped). A pathogenic coccus can almost always be classified as either a **STAPHYLOCOCCUS** (occurring in clusters), or a **STREPTOCOCCUS** (occurring in short or long chains). Both staphylococci and streptococci are gram-positive and do not form spores.

Definition of Coccic given by Medical Dictionary

Implementation Details

Programming Language(s): Python

Operating System: Windows

Library Packages: re, socket, random, pygame

Interface Design: Pygame GUI

Servers Used: Locally hosted server

Dictionary: Wordlist repository, with a few modifications: all words are not hyphenated and length of set of words is at least 2 and at most 5.

Limitations and Future Prospects

Since we are using an old GitHub repository as a source for our dictionary, the upcoming new words would not be present in it.
Since there is no official text format for official dictionaries like Oxford or Merriam Webster, we would have to resort to some new and updated repository.

The dictionary also contains abbreviations like AAS for Associate in Applied Science, which are not technically words, and hence user can get confused.

The dictionary has the same word in both British and American English, for e.g. catalog and catalogue and hence region-specific checking cannot be done.

If somehow, we are able to source some official dictionary in either json/txt file format, we can also include the description of the word which can include its type (noun, verb, adjective etc), meaning, origins etc.

Code Snippets

1. filter_dictionary.py

```
f = open('master.txt', 'r')
line = f.readline().replace('\n', '')
while line:
    line = line.lower()
    if 2 <= len(set(line)) <= 5 and '-' not in line:
        print(line)
    line = f.readline().replace('\n', '')
```

2. server.py

```
import socket as s


class Dictionary:
    def __init__(self):
        self.data = {
            "a": list(), "b": list(), "c": list(), "d": list(), "e": list(), "f": list(), "g": list(), "h": list(),
            "i": list(), "j": list(), "k": list(), "l": list(), "m": list(), "n": list(), "o": list(), "p": list(),
            "q": list(), "r": list(), "s": list(), "t": list(), "u": list(), "v": list(), "w": list(), "x": list(),
            "y": list(), "z": list(), "all": list()
        }

    def insert(self, char, word_inp):
        self.data[char].append(word_inp)

    def refer(self, char):
        return self.data[char]


dictionary = Dictionary()
f = open('words.txt', 'r')
while True:
    line = f.readline().replace('\n', '')
    if line:
        dictionary.insert(line[0], line)
        dictionary.insert("all", line)
    else:
        break

socket = s.socket(s.AF_INET, s.SOCK_STREAM)
socket.bind(('127.0.0.1', 80))
socket.listen()
```

```
while True:
    client, address = socket.accept()
    print(f"Connected to {address}")

    while True:
        message = client.recv(1024).decode("utf-8")
        if message == "2":
            break
        letters = set(message)
        if ' ' in letters:
            letters.remove(' ')

        size = 0
        word = client.recv(1024).decode("utf-8")
        if word == "2":
            break
        word_set = set(word)

        maxima = 0
        max_word = ''
        for words in dictionary.refer("all"):
            S = set(words)
            if S.issubset(letters):
                if maxima < len(words):
                    maxima = len(words)
                    max_word = words

        if max_word == '':
            client.send(bytes('It is not possible to make a valid word!'
                              'Hence this round won\'t be counted', 'utf-8'))
        else:
            if word in dictionary.refer(word[0]):
                if word_set.issubset(letters):
                    client.send(bytes(f"You scored {len(word)}/{len(max_word)}!"
                                      f"Longest word: {max_word}", 'utf-8'))
                else:
                    client.send(bytes(f"You scored 0/{len(max_word)}! "
                                      f"Subset check failed!Longest word: {max_word}", 'utf-8'))
            else:
                client.send(bytes(f'You scored 0/{len(max_word)}!Your word wasn\'t present in the dictionary!'
                                  f'Longest word: {max_word}', 'utf-8'))

    client.close()
```

3. client.py

```
import random
import re
import socket as s

import pygame

socket = s.socket(s.AF_INET, s.SOCK_STREAM)
socket.connect(('127.0.0.1', 80))
letters = ''
word = ''
message = ''
vowels = 'eeeeeeeeeeeeaaaaaaaaaiiiiiiiiiiooooooooouuuu'
consonants = 'nnnnnnrrrrrrttttttllllssssddddgggbbccmmppffhhvvwwyykjxqz'
my_score = 0
total_score = 0
i = 0
v = 0

# initialisation of game
pygame.init()

res = (800, 600)
screen = pygame.display.set_mode(res)

# background, title, icon
background = pygame.image.load("bg.jpeg")
pygame.display.set_caption("Guess The Longest Word!")
icon = pygame.image.load("icon.png")
pygame.display.set_icon(icon)

game_state = "ready"
```

```
# other text
heading = pygame.font.Font('freesansbold.ttf', 50)
subheading = pygame.font.Font('freesansbold.ttf', 32)
titleX = 70
titleY = 150


def score_update(msg):
    global my_score
    global total_score
    if '/' in msg:
        scores = re.findall(r'\d{1,2}/\d{1,2}', msg)[0]
        scores = scores.split('/')
        my_score += int(scores[0])
        total_score += int(scores[1])


def title(x, y):
    name = heading.render("Guess The Longest Word!", True, (0, 0, 0))
    start = subheading.render("Press Enter to Play", True, (0, 0, 0))
    screen.blit(name, (x, y))
    screen.blit(start, (x + 170, y + 150))


def display_letters(x, y, w):
    input_letters = heading.render("Your letters : " + str(w), True, (0, 0, 0))
    screen.blit(input_letters, (x, y))


def ask(x, y):
    ques = subheading.render("Vowel(0) or Consonant(1)?", True, (0, 0, 0))
    screen.blit(ques, (x, y))
```

```
def display_word(x, y, w):
    input_word = subheading.render("Your word : " + str(w), True, (0, 0, 0))
    screen.blit(input_word, (x, y))


def display_score():
    score = subheading.render(f"SCORE : {str(my_score)}/{str(total_score)}", True, (0, 0, 0))
    screen.blit(score, (0, 0))


def final(w):
    w = w.split('!')
    y = 150
    for line in w:
        result1 = subheading.render(line, True, (0, 0, 0))
        screen.blit(result1, (70, y))
        y += 40

    start = subheading.render("Press Enter to Replay", True, (0, 0, 0))
    screen.blit(start, (220, y + 40))


# game loop
running = True
while running:
    screen.fill((255, 255, 255))  # RGB
    screen.blit(background, (0, 0))

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            socket.send(bytes("2", "utf-8"))
            running = False
```

```
    # detecting keystrokes
    if event.type == pygame.KEYDOWN:
        if game_state == "ready":
            # start choosing
            if event.key == pygame.K_RETURN:
                game_state = "choosing"

        elif game_state == "choosing":
            if event.key == pygame.K_0:
                ind = random.randint(0, len(vowels) - 1)
                while vowels[ind] in letters:
                    ind = random.randint(0, len(vowels) - 1)
                letters += vowels[ind] + " "
            elif event.key == pygame.K_1:
                ind = random.randint(0, len(consonants) - 1)
                while consonants[ind] in letters:
                    ind = random.randint(0, len(consonants) - 1)
                letters += consonants[ind] + " "

        elif game_state == "waiting":
            if event.key == pygame.K_a:
                word += 'a'
            elif event.key == pygame.K_b:
                word += 'b'
            elif event.key == pygame.K_c:
                word += 'c'
            elif event.key == pygame.K_d:
                word += 'd'
            elif event.key == pygame.K_e:
                word += 'e'
            elif event.key == pygame.K_f:
                word += 'f'
```

```
            elif event.key == pygame.K_g:
                word += 'g'
            elif event.key == pygame.K_h:
                word += 'h'
            elif event.key == pygame.K_i:
                word += 'i'
            elif event.key == pygame.K_j:
                word += 'j'
            elif event.key == pygame.K_k:
                word += 'k'
            elif event.key == pygame.K_l:
                word += 'l'
            elif event.key == pygame.K_m:
                word += 'm'
            elif event.key == pygame.K_n:
                word += 'n'
            elif event.key == pygame.K_o:
                word += 'o'
            elif event.key == pygame.K_p:
                word += 'p'
            elif event.key == pygame.K_q:
                word += 'q'
            elif event.key == pygame.K_r:
                word += 'r'
            elif event.key == pygame.K_s:
                word += 's'
            elif event.key == pygame.K_t:
                word += 't'
            elif event.key == pygame.K_u:
                word += 'u'
            elif event.key == pygame.K_v:
                word += 'v'
```

```
                elif event.key == pygame.K_w:
                    word += 'w'
                elif event.key == pygame.K_x:
                    word += 'x'
                elif event.key == pygame.K_y:
                    word += 'y'
                elif event.key == pygame.K_z:
                    word += 'z'
                elif event.key == pygame.K_BACKSPACE:
                    x = len(word) - 1
                    temp = word[:x]
                    word = temp
                elif event.key == pygame.K_RETURN:
                    if not word:
                        word = "dummystring"
                    game_state = "result"
            elif game_state == "finished":
                if event.key == pygame.K_RETURN:
                    game_state = "choosing"

    if game_state == "ready":
        title(titleX, titleY)

    elif game_state == "choosing":
        display_score()
        display_letters(titleX, titleY, letters)

        if len(letters) < 10:
            ask(titleX, titleY + 150)
        elif len(letters) >= 10:
            game_state = "waiting"
            socket.send(bytes(letters, "utf-8"))
    elif game_state == "waiting":
        display_score()
        display_letters(titleX, titleY, letters)
        display_word(titleX, titleY + 150, word)

    elif game_state == "result":
        display_score()
        socket.send(bytes(word, "utf-8"))
        message = socket.recv(1024).decode("utf-8")
        score_update(message)
        print(message)
        final(message)
        letters = ""
        word = ""
        game_state = "finished"

    elif game_state == "finished":
        display_score()
        final(message)

    pygame.display.update()
```

4. GUI's background, icon, master.txt words.txt
   Available on the [GitHub repository](#).