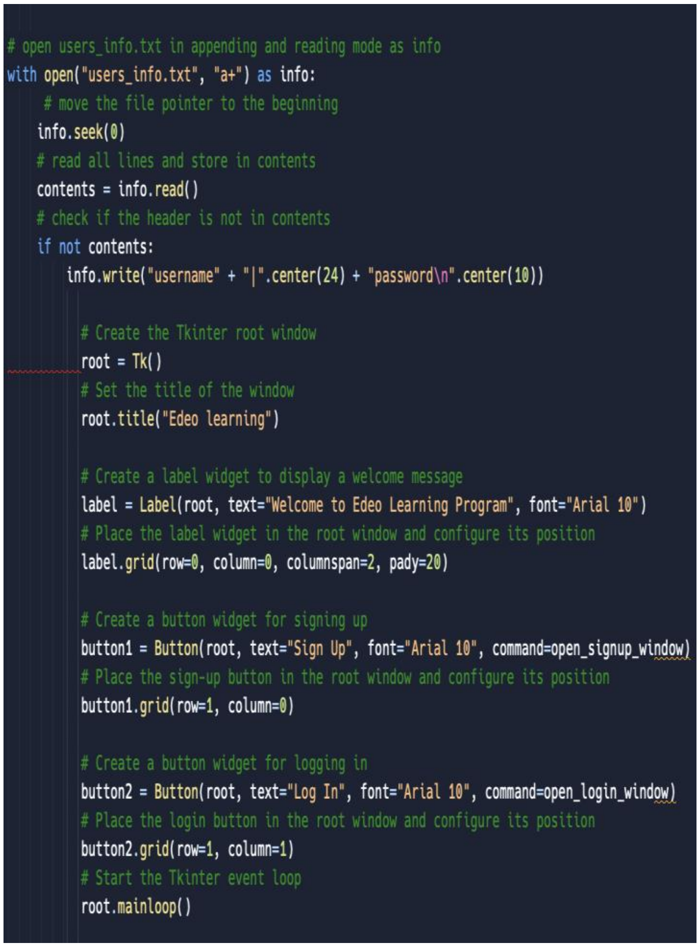
Issues and Fixes

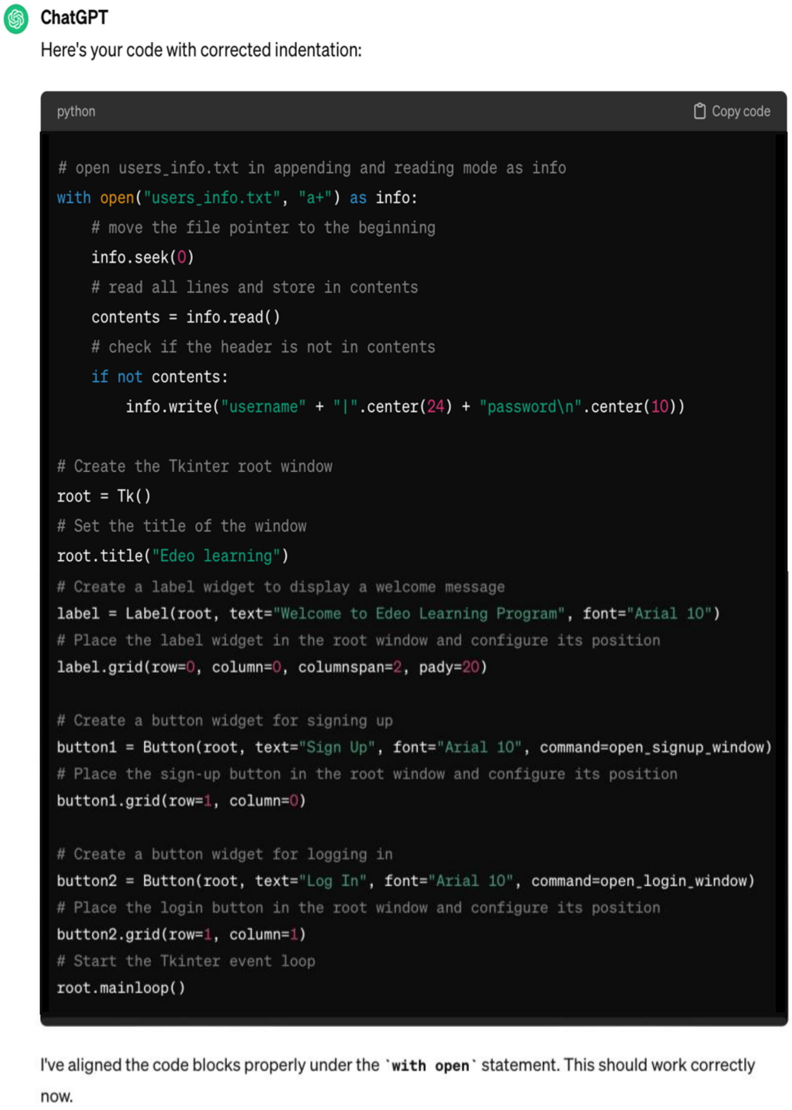
Indentation: it was the main issue that encouraged me, which I successfully resolved with the assistance of ChatGPT. The attached photo serves as evidence:

Probelm



```
# open users_info.txt in appending and reading mode as info
with open("users_info.txt", "a+") as info:
     # move the file pointer to the beginning
    info.seek(0)
    # read all lines and store in contents
    contents = info.read()
    # check if the header is not in contents
    if not contents:
        info.write("username" + "|".center(24) + "password\n".center(10))

          # Create the Tkinter root window
          root = Tk()
          # Set the title of the window
          root.title("Edeo learning")

          # Create a label widget to display a welcome message
          label = Label(root, text="Welcome to Edeo Learning Program", font="Arial 10")
          # Place the label widget in the root window and configure its position
          label.grid(row=0, column=0, columnspan=2, pady=20)

          # Create a button widget for signing up
          button1 = Button(root, text="Sign Up", font="Arial 10", command=open_signup_window)
          # Place the sign-up button in the root window and configure its position
          button1.grid(row=1, column=0)

          # Create a button widget for logging in
          button2 = Button(root, text="Log In", font="Arial 10", command=open_login_window)
          # Place the login button in the root window and configure its position
          button2.grid(row=1, column=1)
          # Start the Tkinter event loop
          root.mainloop()
```

solution



ChatGPT

Here's your code with corrected indentation:

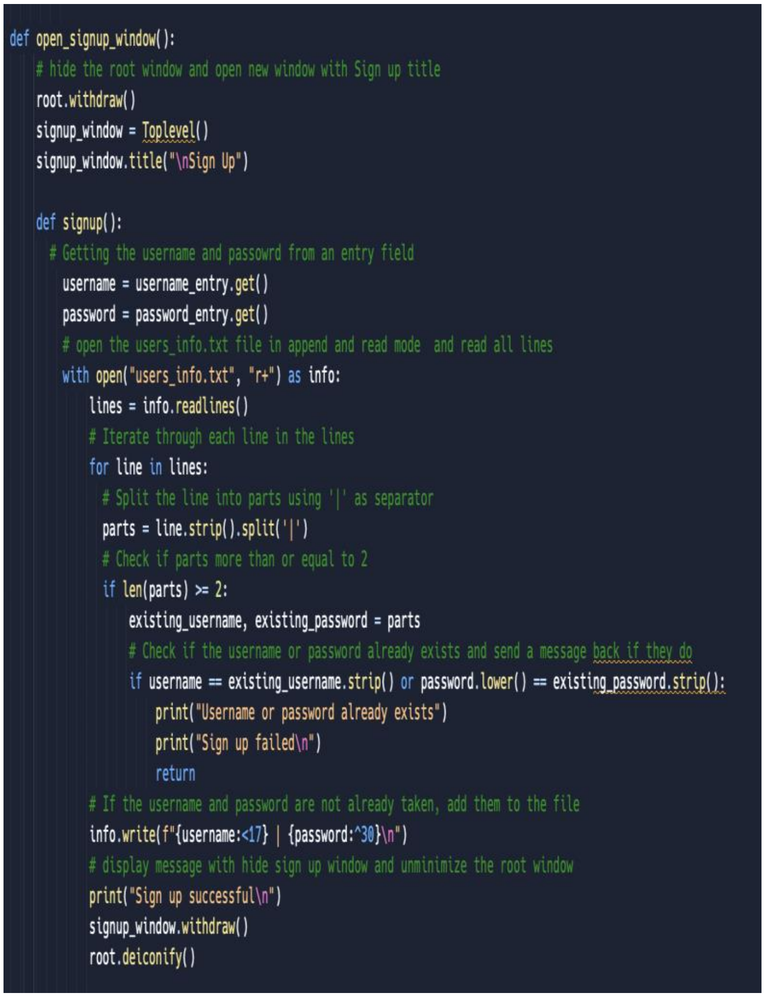
```python
# open users_info.txt in appending and reading mode as info
with open("users_info.txt", "a+") as info:
    # move the file pointer to the beginning
    info.seek(0)
    # read all lines and store in contents
    contents = info.read()
    # check if the header is not in contents
    if not contents:
        info.write("username" + "|".center(24) + "password\n".center(10))

# Create the Tkinter root window
root = Tk()
# Set the title of the window
root.title("Edeo learning")
# Create a label widget to display a welcome message
label = Label(root, text="Welcome to Edeo Learning Program", font="Arial 10")
# Place the label widget in the root window and configure its position
label.grid(row=0, column=0, columnspan=2, pady=20)

# Create a button widget for signing up
button1 = Button(root, text="Sign Up", font="Arial 10", command=open_signup_window)
# Place the sign-up button in the root window and configure its position
button1.grid(row=1, column=0)

# Create a button widget for logging in
button2 = Button(root, text="Log In", font="Arial 10", command=open_login_window)
# Place the login button in the root window and configure its position
button2.grid(row=1, column=1)
# Start the Tkinter event loop
root.mainloop()
```

I've aligned the code blocks properly under the `with open` statement. This should work correctly now.

This code starter is essential for the proper execution of the code, as without proper indentation, the code will not function correctly.

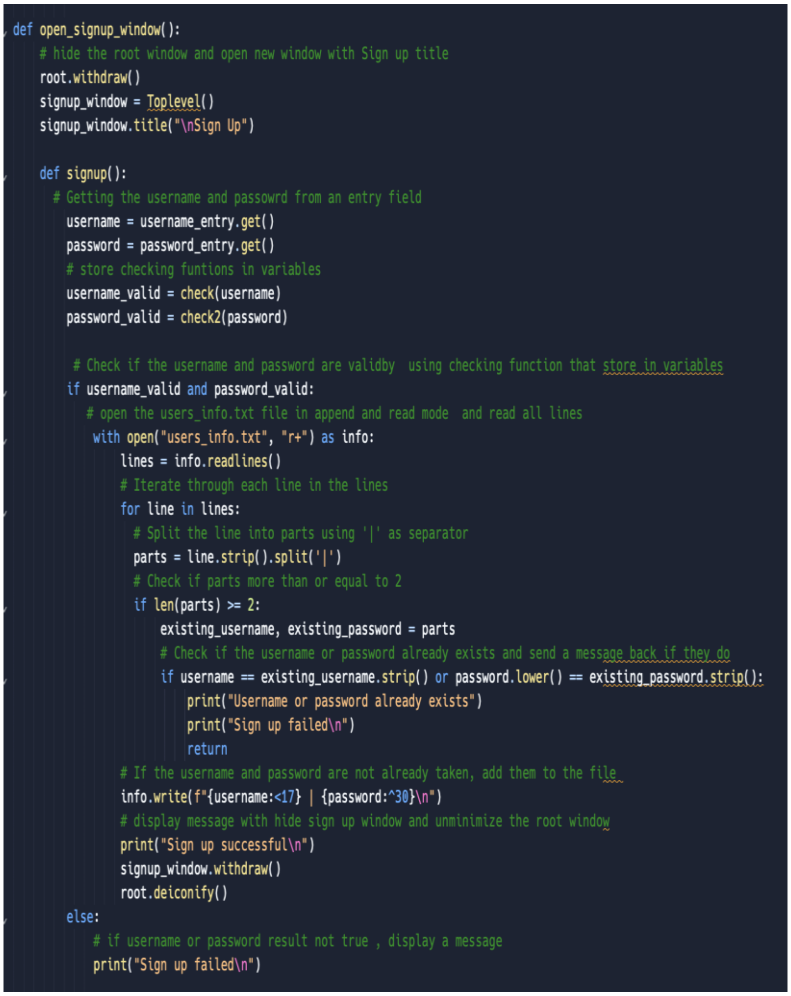
GUI linkage to functions of checking: After verifying username and password security functions, I faced challenges linking them to the GUI for file updates.

Probelm



```
def open_signup_window():
    # hide the root window and open new window with Sign up title
    root.withdraw()
    signup_window = Toplevel()
    signup_window.title("\nSign Up")

    def signup():
      # Getting the username and passowrd from an entry field
        username = username_entry.get()
        password = password_entry.get()
        # open the users_info.txt file in append and read mode  and read all lines
        with open("users_info.txt", "r+") as info:
            lines = info.readlines()
            # Iterate through each line in the lines
            for line in lines:
              # Split the line into parts using '|' as separator
              parts = line.strip().split('|')
              # Check if parts more than or equal to 2
              if len(parts) >= 2:
                  existing_username, existing_password = parts
                  # Check if the username or password already exists and send a message back if they do
                  if username == existing_username.strip() or password.lower() == existing_password.strip():
                      print("Username or password already exists")
                      print("Sign up failed\n")
                      return
            # If the username and password are not already taken, add them to the file
            info.write(f"{username:<17} | {password:^30}\n")
            # display message with hide sign up window and unminimize the root window
            print("Sign up successful\n")
            signup_window.withdraw()
            root.deiconify()
```

solution

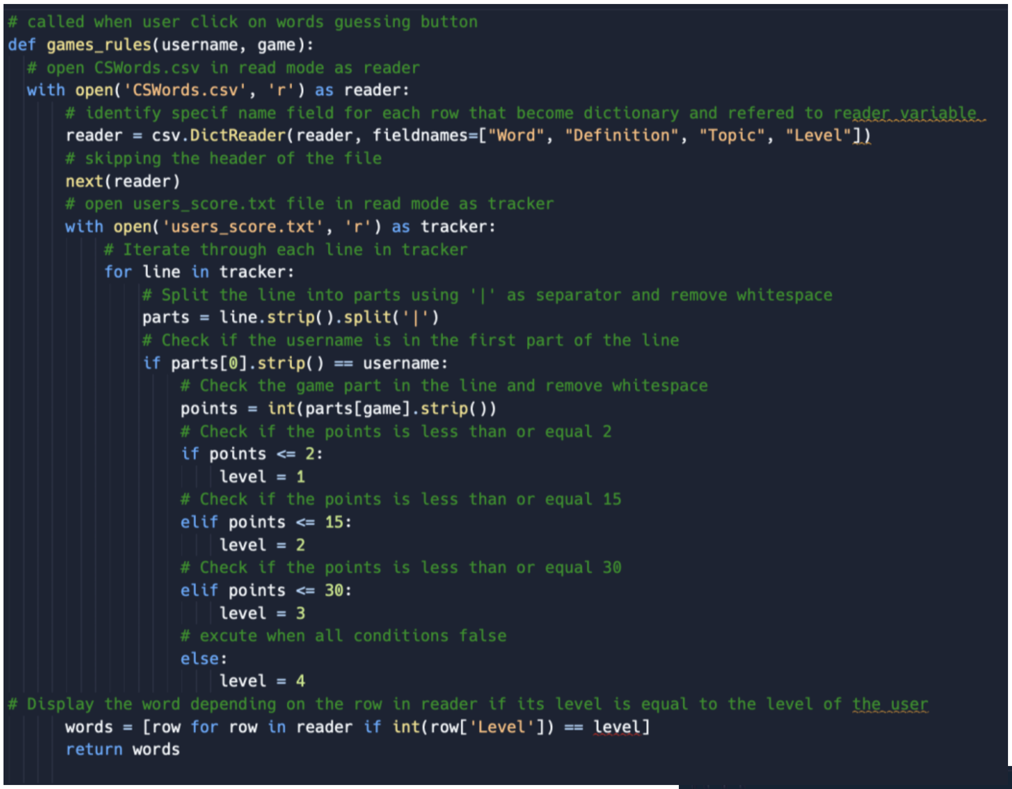


```
def open_signup_window():
    # hide the root window and open new window with Sign up title
    root.withdraw()
    signup_window = Toplevel()
    signup_window.title("\nSign Up")

    def signup():
      # Getting the username and passowrd from an entry field
        username = username_entry.get()
        password = password_entry.get()
        # store checking funtions in variables
        username_valid = check(username)
        password_valid = check2(password)

         # Check if the username and password are validby  using checking function that store in variables
        if username_valid and password_valid:
           # open the users_info.txt file in append and read mode  and read all lines
            with open("users_info.txt", "r+") as info:
                lines = info.readlines()
                # Iterate through each line in the lines
                for line in lines:
                  # Split the line into parts using '|' as separator
                  parts = line.strip().split('|')
                  # Check if parts more than or equal to 2
                  if len(parts) >= 2:
                      existing_username, existing_password = parts
                      # Check if the username or password already exists and send a message back if they do
                      if username == existing_username.strip() or password.lower() == existing_password.strip():
                          print("Username or password already exists")
                          print("Sign up failed\n")
                          return
                # If the username and password are not already taken, add them to the file
                info.write(f"{username:<17} | {password:^30}\n")
                # display message with hide sign up window and unminimize the root window
                print("Sign up successful\n")
                signup_window.withdraw()
                root.deiconify()
        else:
            # if username or password result not true , display a message
            print("Sign up failed\n")
```

I modified the code so that when the user enters the username and password, the check function is invoked. If the credentials are incorrect, no credentials add to the file.

Difficulty Level: match the displayed word for user in the game with its corresponding difficulty level . The problem lies in linking the database field to display the word assigned to a specific level, like level 1

solution

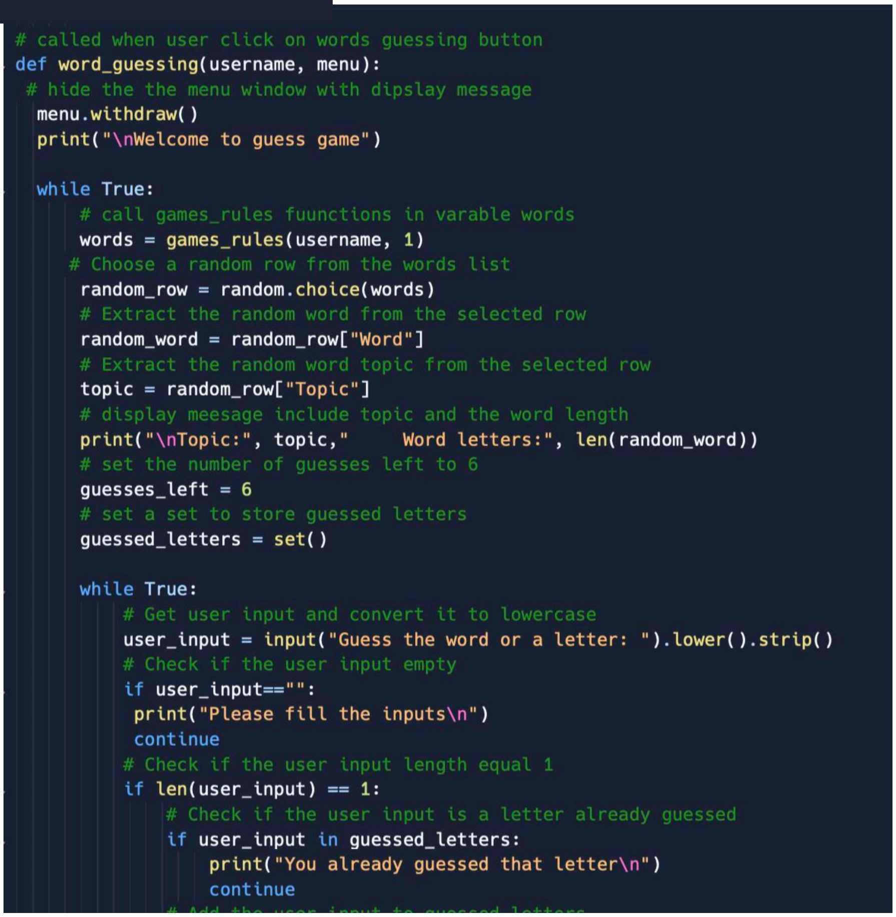


```
# called when user click on words guessing button
def games_rules(username, game):
 # open CSWords.csv in read mode as reader
  with open('CSWords.csv', 'r') as reader:
      # identify specif name field for each row that become dictionary and refered to reader variable
      reader = csv.DictReader(reader, fieldnames=["Word", "Definition", "Topic", "Level"])
      # skipping the header of the file
      next(reader)
      # open users_score.txt file in read mode as tracker
      with open('users_score.txt', 'r') as tracker:
          # Iterate through each line in tracker
          for line in tracker:
              # Split the line into parts using '|' as separator and remove whitespace
              parts = line.strip().split('|')
              # Check if the username is in the first part of the line
              if parts[0].strip() == username:
                  # Check the game part in the line and remove whitespace
                  points = int(parts[game].strip())
                  # Check if the points is less than or equal 2
                  if points <= 2:
                      level = 1
                  # Check if the points is less than or equal 15
                  elif points <= 15:
                      level = 2
                  # Check if the points is less than or equal 30
                  elif points <= 30:
                      level = 3
                  # excute when all conditions false
                  else:
                      level = 4
# Display the word depending on the row in reader if its level is equal to the level of the user
      words = [row for row in reader if int(row['Level']) == level]
      return words
```

I solved the problem by creating a separate function to open the score storage file.

This function checks the game score points, and assigns levels based on these points, linking them to the CSV file to corresponding words for the level.

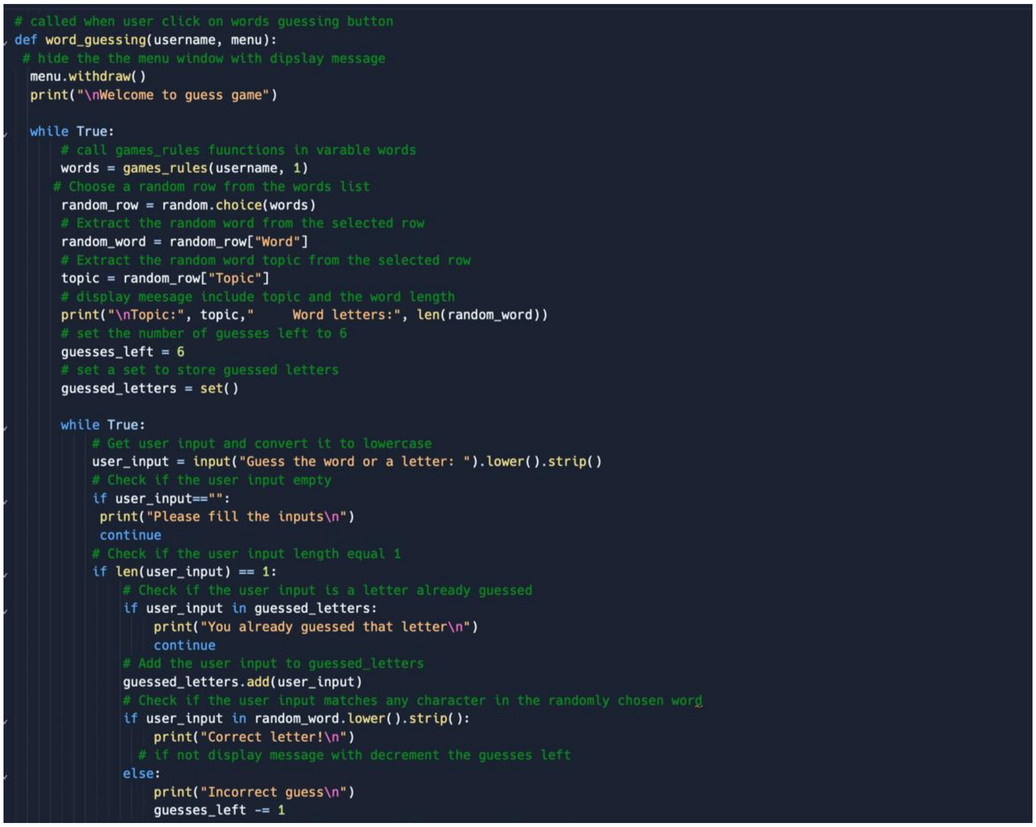
Afterwards, this function is called within the game, as shown in the image.



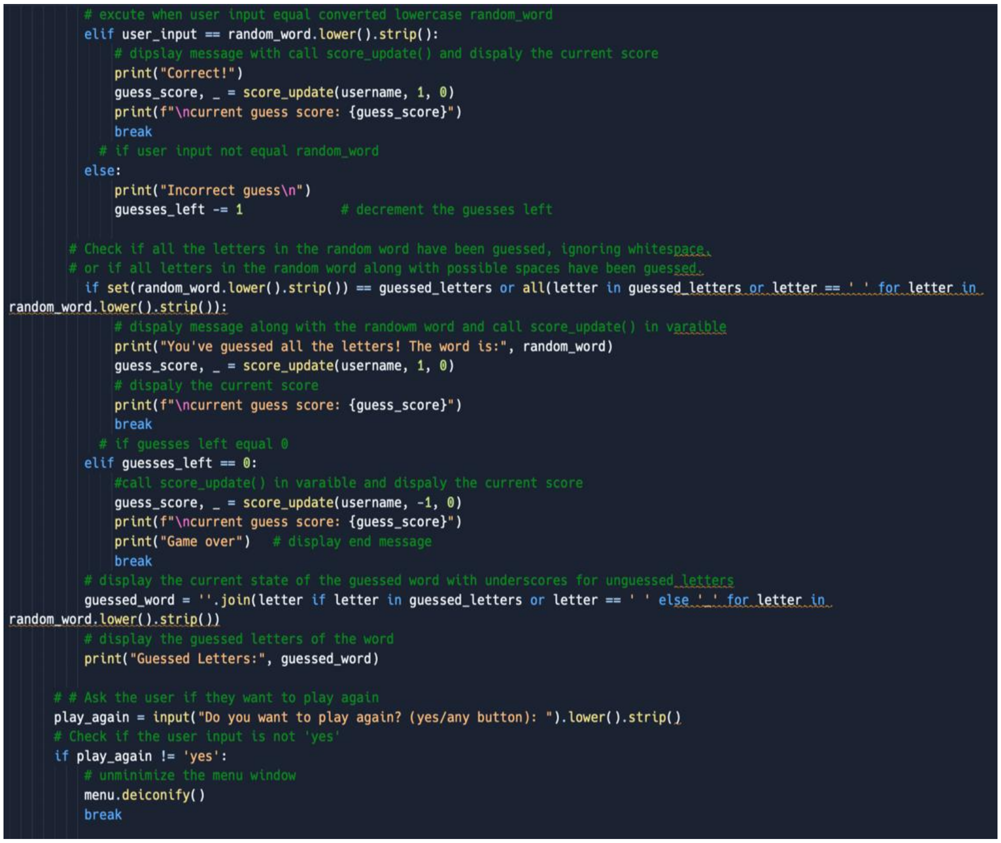
```
# called when user click on words guessing button
def word_guessing(username, menu):
 # hide the the menu window with dipslay message
  menu.withdraw()
  print("\nWelcome to guess game")

  while True:
      # call games_rules fuunctions in varable words
      words = games_rules(username, 1)
     # Choose a random row from the words list
      random_row = random.choice(words)
      # Extract the random word from the selected row
      random_word = random_row["Word"]
      # Extract the random word topic from the selected row
      topic = random_row["Topic"]
      # display meesage include topic and the word length
      print("\nTopic:", topic,"     Word letters:", len(random_word))
      # set the number of guesses left to 6
      guesses_left = 6
      # set a set to store guessed letters
      guessed_letters = set()

      while True:
          # Get user input and convert it to lowercase
          user_input = input("Guess the word or a letter: ").lower().strip()
          # Check if the user input empty
          if user_input=="":
           print("Please fill the inputs\n")
           continue
          # Check if the user input length equal 1
          if len(user_input) == 1:
              # Check if the user input is a letter already guessed
              if user_input in guessed_letters:
                  print("You already guessed that letter\n")
                  continue
```

Visualize progress : Display the guessed letters in the word for user to be aware of his progress and to skip guess same letter

solution



```
# called when user click on words guessing button
def word_guessing(username, menu):
 # hide the the menu window with dipslay message
  menu.withdraw()
  print("\nWelcome to guess game")

  while True:
      # call games_rules fuunctions in varable words
      words = games_rules(username, 1)
     # Choose a random row from the words list
      random_row = random.choice(words)
      # Extract the random word from the selected row
      random_word = random_row["Word"]
      # Extract the random word topic from the selected row
      topic = random_row["Topic"]
      # display meesage include topic and the word length
      print("\nTopic:", topic,"     Word letters:", len(random_word))
      # set the number of guesses left to 6
      guesses_left = 6
      # set a set to store guessed letters
      guessed_letters = set()

      while True:
          # Get user input and convert it to lowercase
          user_input = input("Guess the word or a letter: ").lower().strip()
          # Check if the user input empty
          if user_input=="":
           print("Please fill the inputs\n")
           continue
          # Check if the user input length equal 1
          if len(user_input) == 1:
              # Check if the user input is a letter already guessed
              if user_input in guessed_letters:
                  print("You already guessed that letter\n")
                  continue
              # Add the user input to guessed_letters
              guessed_letters.add(user_input)
              # Check if the user input matches any character in the randomly chosen word
              if user_input in random_word.lower().strip():
                  print("Correct letter!\n")
                # if not display message with decrement the guesses left
              else:
                  print("Incorrect guess\n")
                  guesses_left -= 1
```



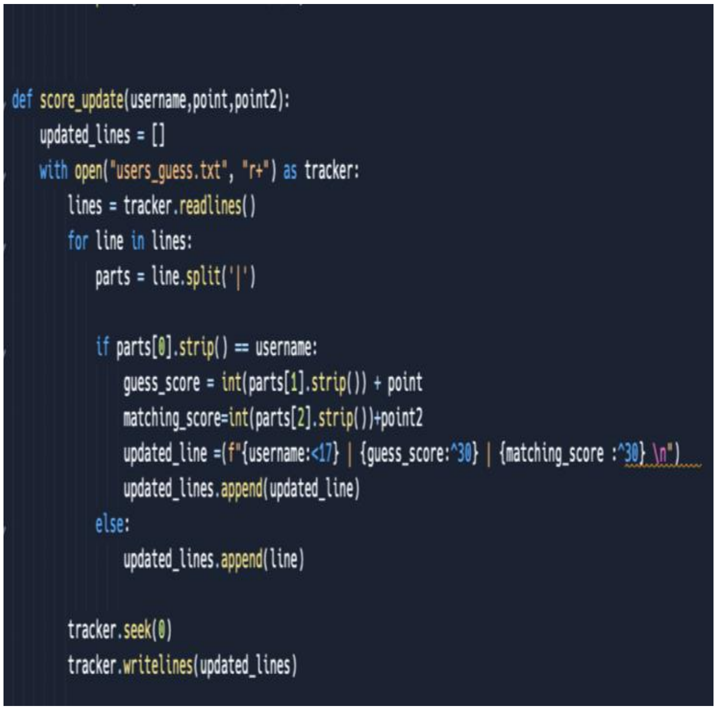
```
          # excute when user input equal converted lowercase random_word
          elif user_input == random_word.lower().strip():
              # dipslay message with call score_update() and dispaly the current score
              print("Correct!")
              guess_score, _ = score_update(username, 1, 0)
              print(f"\ncurrent guess score: {guess_score}")
              break
            # if user input not equal random_word
          else:
              print("Incorrect guess\n")
              guesses_left -= 1              # decrement the guesses left

        # Check if all the letters in the random word have been guessed, ignoring whitespace.
        # or if all letters in the random word along with possible spaces have been guessed.
          if set(random_word.lower().strip()) == guessed_letters or all(letter in guessed_letters or letter == ' ' for letter in
random_word.lower().strip()):
              # dispaly message along with the randowm word and call score_update() in varaible
              print("You've guessed all the letters! The word is:", random_word)
              guess_score, _ = score_update(username, 1, 0)
              # dispaly the current score
              print(f"\ncurrent guess score: {guess_score}")
              break
            # if guesses left equal 0
          elif guesses_left == 0:
              #call score_update() in varaible and dispaly the current score
              guess_score, _ = score_update(username, -1, 0)
              print(f"\ncurrent guess score: {guess_score}")
              print("Game over")   # display end message
              break
          # display the current state of the guessed word with underscores for unguessed letters
          guessed_word = ''.join(letter if letter in guessed_letters or letter == ' ' else '_' for letter in
random_word.lower().strip())
          # display the guessed letters of the word
          print("Guessed Letters:", guessed_word)

      # # Ask the user if they want to play again
      play_again = input("Do you want to play again? (yes/any button): ").lower().strip()
      # Check if the user input is not 'yes'
      if play_again != 'yes':
          # unminimize the menu window
          menu.deiconify()
          break
```

I solved by use join function to merge guessed letters with their positions in the word for display of correct guesses and remaining unknown letters disappear.

This enhances user to track the progress

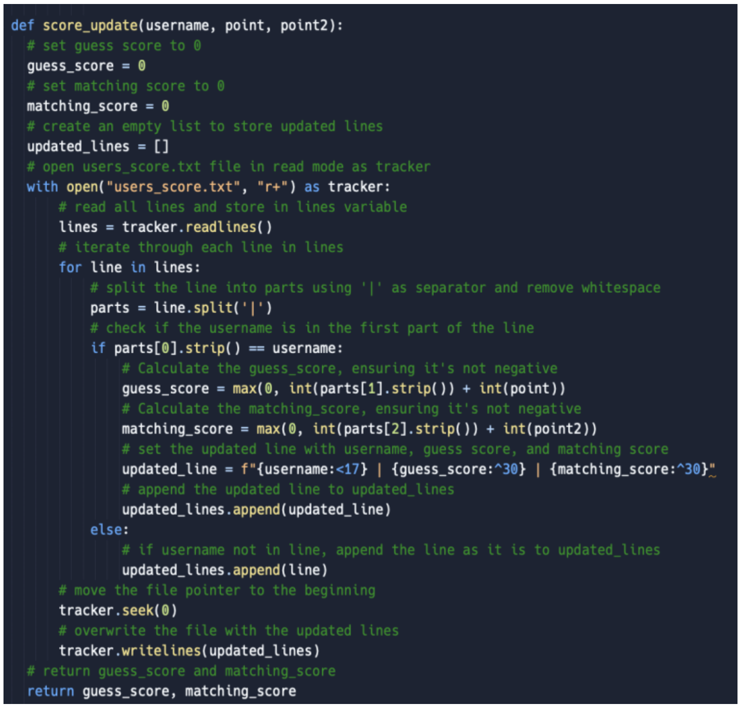
Game scores: The task challenge presenting the current score of a particular game using a function named score_update(). I have successfully addressed this challenge by following the steps in the shown image:

Problem



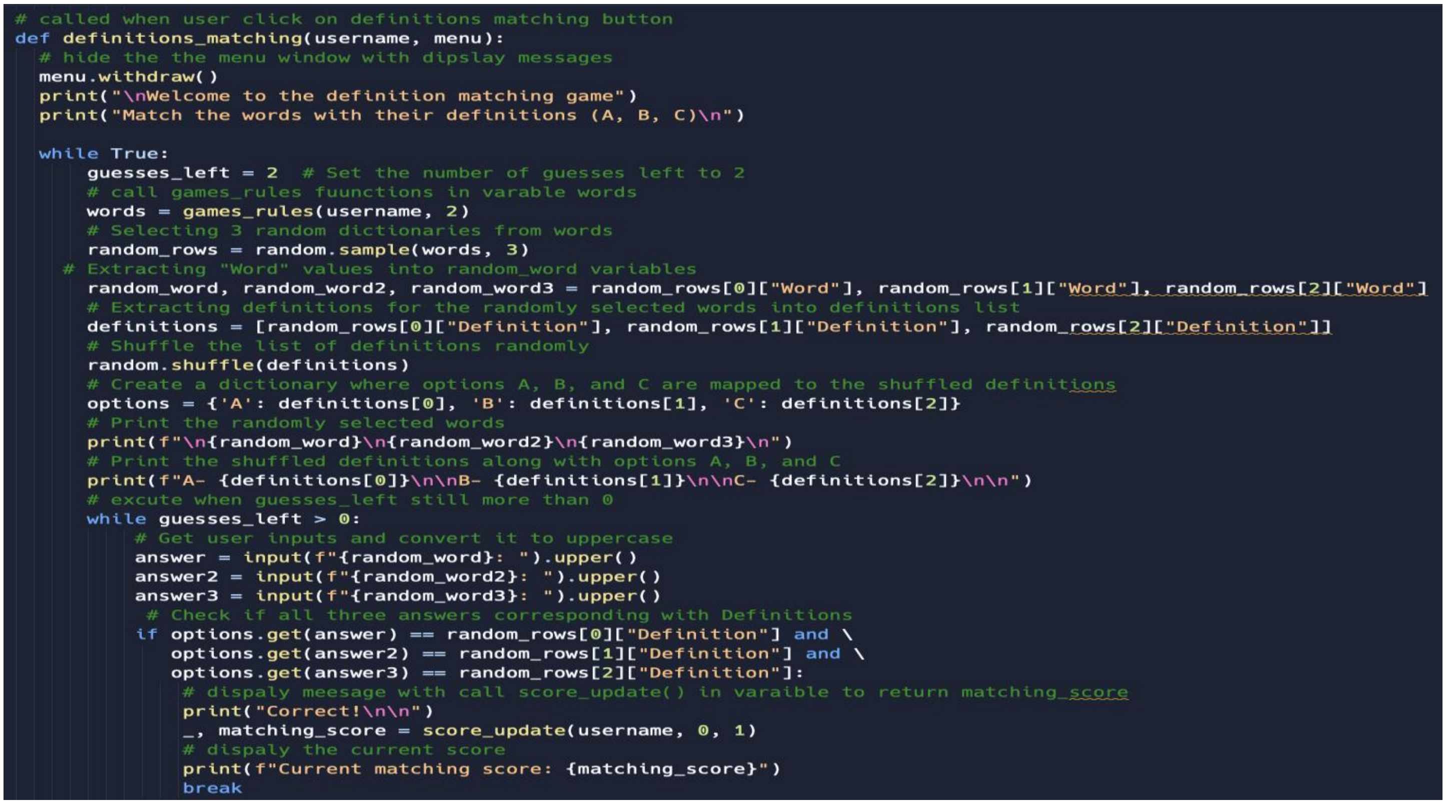
```
def score_update(username,point,point2):
    updated_lines = []
    with open("users_guess.txt", "r+") as tracker:
        lines = tracker.readlines()
        for line in lines:
            parts = line.split('|')

            if parts[0].strip() == username:
                guess_score = int(parts[1].strip()) + point
                matching_score=int(parts[2].strip())+point2
                updated_line =(f"{username:<17} | {guess_score:^30} | {matching_score :^30} \n")
                updated_lines.append(updated_line)
            else:
                updated_lines.append(line)

        tracker.seek(0)
        tracker.writelines(updated_lines)
```

Solution



```
def score_update(username, point, point2):
  # set guess score to 0
  guess_score = 0
  # set matching score to 0
  matching_score = 0
  # create an empty list to store updated lines
  updated_lines = []
  # open users_score.txt file in read mode as tracker
  with open("users_score.txt", "r+") as tracker:
      # read all lines and store in lines variable
      lines = tracker.readlines()
      # iterate through each line in lines
      for line in lines:
          # split the line into parts using '|' as separator and remove whitespace
          parts = line.split('|')
          # check if the username is in the first part of the line
          if parts[0].strip() == username:
              # Calculate the guess_score, ensuring it's not negative
              guess_score = max(0, int(parts[1].strip()) + int(point))
              # Calculate the matching_score, ensuring it's not negative
              matching_score = max(0, int(parts[2].strip()) + int(point2))
              # set the updated line with username, guess score, and matching score
              updated_line = f"{username:<17} | {guess_score:^30} | {matching_score:^30}"
              # append the updated line to updated_lines
              updated_lines.append(updated_line)
          else:
              # if username not in line, append the line as it is to updated_lines
              updated_lines.append(line)
      # move the file pointer to the beginning
      tracker.seek(0)
      # overwrite the file with the updated lines
      tracker.writelines(updated_lines)
  # return guess_score and matching_score
  return guess_score, matching_score
```

I adjusted the score_update() function to return both scores needed for the game. When used in the game, it retrieves the selected score from both sets of scores, as shown in the image:



```
# called when user click on definitions matching button
def definitions_matching(username, menu):
  # hide the the menu window with dipslay messages
  menu.withdraw()
  print("\nWelcome to the definition matching game")
  print("Match the words with their definitions (A, B, C)\n")

  while True:
      guesses_left = 2  # Set the number of guesses left to 2
      # call games_rules fuunctions in varable words
      words = games_rules(username, 2)
      # Selecting 3 random dictionaries from words
      random_rows = random.sample(words, 3)
    # Extracting "Word" values into random_word variables
      random_word, random_word2, random_word3 = random_rows[0]["Word"], random_rows[1]["Word"], random_rows[2]["Word"]
      # Extracting definitions for the randomly selected words into definitions list
      definitions = [random_rows[0]["Definition"], random_rows[1]["Definition"], random_rows[2]["Definition"]]
      # Shuffle the list of definitions randomly
      random.shuffle(definitions)
      # Create a dictionary where options A, B, and C are mapped to the shuffled definitions
      options = {'A': definitions[0], 'B': definitions[1], 'C': definitions[2]}
      # Print the randomly selected words
      print(f"\n{random_word}\n{random_word2}\n{random_word3}\n")
      # Print the shuffled definitions along with options A, B, and C
      print(f"A- {definitions[0]}\n\nB- {definitions[1]}\n\nC- {definitions[2]}\n\n")
      # excute when guesses_left still more than 0
      while guesses_left > 0:
          # Get user inputs and convert it to uppercase
          answer = input(f"{random_word}: ").upper()
          answer2 = input(f"{random_word2}: ").upper()
          answer3 = input(f"{random_word3}: ").upper()
           # Check if all three answers corresponding with Definitions
          if options.get(answer) == random_rows[0]["Definition"] and \
             options.get(answer2) == random_rows[1]["Definition"] and \
             options.get(answer3) == random_rows[2]["Definition"]:
              # dispaly meesage with call score_update() in varaible to return matching_score
              print("Correct!\n\n")
              _, matching_score = score_update(username, 0, 1)
              # dispaly the current score
              print(f"Current matching score: {matching_score}")
              break
```