Bing

Voici la suite du code que j'ai rédigé pour vous. 🙏

# Définir la méthode qui affiche l'écran du jeu pour le mode utilisateur

def show\_game\_screen(self):

# Effacer tous les widgets de la fenêtre principale

for widget in self.winfo\_children():

widget.destroy()

# Créer un cadre pour contenir les widgets de l'écran du jeu

frame = tk.Frame(self)

frame.pack(padx=20, pady=20)

# Créer un label pour afficher le titre de l'application

title = tk.Label(frame, text="Asseguim - Apprendre l'anglais de manière ludique et en équipe", font=("Arial", 24, "bold"))

title.grid(row=0, column=0, columnspan=4, pady=10)

# Créer un label pour afficher le nom de l'utilisateur

name\_label = tk.Label(frame, text="Bonjour " + self.name.get(), font=("Arial", 16))

name\_label.grid(row=1, column=0)

# Créer un label pour afficher le niveau choisi

level\_label = tk.Label(frame, text="Niveau: " + self.level.get(), font=("Arial", 16))

level\_label.grid(row=1, column=1)

# Créer un label pour afficher le score de l'utilisateur

score\_label = tk.Label(frame, text="Score: " + str(self.score.get()), font=("Arial", 16))

score\_label.grid(row=1, column=2)

# Créer un label pour afficher la division de l'utilisateur

division\_label = tk.Label(frame, text="Division: " + self.divisions[self.get\_division()], font=("Arial", 16), bg=self.colors[self.get\_division()])

division\_label.grid(row=1, column=3)

# Créer un label pour afficher la question courante

question\_label = tk.Label(frame, text=self.question.get(), font=("Arial", 16), wraplength=600)

question\_label.grid(row=2, column=0, columnspan=4, pady=10)

# Créer une liste vide pour stocker les boutons des choix de réponse

self.choices = []

# Créer une boucle pour créer quatre boutons des choix de réponse

for i in range(4):

# Créer un bouton avec un texte vide et une commande qui appelle la méthode qui vérifie la réponse

choice\_button = tk.Button(frame, text="", font=("Arial", 16), command=lambda i=i: self.check\_answer(i))

# Ajouter le bouton à la liste des choix

self.choices.append(choice\_button)

# Placer le bouton dans le cadre selon sa position

if i < 2:

choice\_button.grid(row=3, column=i\*2, padx=10, pady=10)

else:

choice\_button.grid(row=4, column=(i-2)\*2, padx=10, pady=10)

# Définir la méthode qui génère une nouvelle question pour le mode utilisateur

def new\_question(self):

# Récupérer le niveau choisi

level = self.level.get()

# Exécuter une requête SQL pour sélectionner une question aléatoire du niveau choisi

self.cur.execute("SELECT \* FROM questions WHERE level=? ORDER BY RANDOM() LIMIT 1", (level,))

# Récupérer le résultat de la requête sous forme de tuple

result = self.cur.fetchone()

# Vérifier que le résultat n'est pas vide

if result:

# Extraire les éléments du tuple et les affecter aux variables correspondantes

id, question, answer, choices, level = result

self.question.set(question)

self.answer.set(answer)

self.weight.set(int(level))

# Mélanger les choix de réponse de manière aléatoire

random.shuffle(choices)

# Affecter les choix de réponse aux boutons correspondants

for i in range(4):

self.choices[i].config(text=choices[i])

# Définir la méthode qui vérifie la réponse choisie par l'utilisateur et met à jour le score et la division

def check\_answer(self, i):

# Récupérer le texte du bouton choisi par l'utilisateur

choice = self.choices[i].cget("text")

# Récupérer la réponse correcte à la question courante

answer = self.answer.get()

# Récupérer la pondération de la question courante

weight = self.weight.get()

# Vérifier si le choix est égal à la réponse

if choice == answer:

# Afficher un message de félicitation

tk.messagebox.showinfo("Bravo", "Vous avez trouvé la bonne réponse!")

# Augmenter le score de l'utilisateur selon la pondération

self.score.set(self.score.get() + weight)

# Mettre à jour le classement des joueurs

self.update\_ranking()

# Mettre à jour la division du joueur

self.update\_division()

# Sinon

else:

# Afficher un message de déception

tk.messagebox.showerror("Dommage", "Vous avez choisi la mauvaise réponse. La bonne réponse était: " + answer)

# Générer une nouvelle question

self.new\_question()

# Définir la méthode qui met à jour le classement des joueurs selon leur score

def update\_ranking(self):

# Récupérer le nom de l'utilisateur

name = self.name.get()

# Récupérer l'identifiant de l'utilisateur

id = self.id.get()

# Récupérer le score de l'utilisateur

score = self.score.get()

# Vérifier si l'utilisateur existe déjà dans la table des joueurs

self.cur.execute("SELECT \* FROM players WHERE name=? AND id=?", (name, id))

result = self.cur.fetchone()

# Si l'utilisateur existe déjà, mettre à jour son score

if result:

self.cur.execute("UPDATE players SET score=? WHERE name=? AND id=?", (score, name, id))

# Sinon, insérer un nouveau joueur avec son nom, son identifiant et son score

else:

self.cur.execute("INSERT INTO players (name, id, score) VALUES (?, ?, ?)", (name, id, score))

# Valider les changements dans la base de données

self.conn.commit()

# Définir la méthode qui met à jour la division du joueur selon son score et son rang

def update\_division(self):

# Récupérer le nom de l'utilisateur

name = self.name.get()

# Récupérer le score de l'utilisateur

score = self.score.get()

# Exécuter une requête SQL pour sélectionner tous les joueurs triés par ordre décroissant de score

self.cur.execute("SELECT \* FROM players ORDER BY score DESC")

# Récupérer le résultat de la requête sous forme de liste de tuples

results = self.cur.fetchall()

# Créer une variable pour stocker le rang du joueur

rank = 0

# Créer une boucle pour parcourir la liste des joueurs

for i in range(len(results)):

# Extraire le nom et le score du joueur courant

player\_name, player\_id, player\_score = results[i]

# Si le nom et le score du joueur courant sont égaux à ceux de l'utilisateur, affecter i+1 au rang du joueur et sortir de la boucle

if player\_name == name and player\_score == score:

rank = i + 1

break

# Créer une variable pour stocker la division du joueur

division = 0

# Si le rang du joueur est inférieur ou égal à 3 et que son score est supérieur ou égal à 100, affecter 1 à la division du joueur (Mercure)

if rank <= 3 and score >= 100:

division = 1

# Sinon, si le rang du joueur est compris entre 4 et 15, affecter 2 à la division du joueur (Vénus)

elif 4 <= rank <= 15:

division = 2

# Sinon, si le rang du joueur est compris entre 16 et 27, affecter 3 à la division du joueur (Terre)

elif 16 <= rank <= 27:

division = 3