TP : Réalisez un bon vieux pendu

C'est le moment de mettre en pratique ce que vous avez appris. Vous n'aurez pas besoin de tout, bien entendu, mais je vais essayer de vous faire travailler un maximum de choses.

Nous allons donc faire un jeu de pendu plutôt classique. Ce n'est pas bien original mais on va pimenter un peu l'exercice, vous allez voir.

Votre mission

Nous y voilà. Je vais vous préciser un peu la mission, sans quoi on va avoir du mal à s'entendre sur la correction.

Un jeu du pendu

Le premier point de la mission est de réaliser un jeu du pendu. Je rappelle brièvement les règles, au cas où : l'ordinateur choisit un mot au hasard dans une liste, un mot de huit lettres maximum. Le joueur tente de trouver les lettres composant le mot. À chaque coup, il saisit une lettre. Si la lettre figure dans le mot, l'ordinateur affiche le mot avec les lettres déjà trouvées. Celles qui ne le sont pas encore sont remplacées par des étoiles (*). Le joueur a 8 chances. Au delà, il a perdu.

On va compliquer un peu les règles en demandant au joueur de donner son nom, au début de la partie. Cela permettra au programme d'enregistrer son score.

Le score du joueur sera simple à calculer : on prend le score courant (0 si le joueur n'a aucun score déjà enregistré) et, à chaque partie, on lui ajoute le nombre de coups restants comme points de partie. Si, par exemple, il me reste trois coups au moment où je trouve le mot, je gagne trois points.

Par la suite, vous pourrez vous amuser à faire un décompte plus poussé du score, pour l'instant cela suffira bien.

Le côté technique du problème

Le jeu du pendu en lui-même, vous ne devriez avoir aucun problème à le mettre en place. Rappelez-vous que le joueur ne doit donner qu'une seule lettre à la fois et que le programme doit bien vérifier que c'est le cas avant de continuer. Nous allons découper notre programme en trois fichiers :

- Le fichier `donnees.py` qui contiendra les variables nécessaires à notre application (la liste des mots, le nombre de chances autorisées…).
- Le fichier `fonctions.py` qui contiendra les fonctions utiles à notre application. Là, je ne vous fais aucune liste claire, je vous conseille de bien y réfléchir, avec une feuille et un stylo si cela vous aide (Quelles sont les actions de mon programme ? Que puis-je mettre dans des fonctions ?).
- Enfin, notre fichier `pendu.py` qui contiendra notre jeu du pendu.

Gérer les scores

Vous avez, j'espère, une petite idée de comment faire cela… mais je vais quand même clarifier : on va enregistrer dans un fichier de données, que l'on va appeler `scores` (sans aucune extension) les scores du jeu. Ces scores seront sous la forme d'un dictionnaire : en clés, nous aurons les noms des joueurs et en valeurs les scores, sous la forme d'entiers.

Il faut gérer les cas suivants :

- Le fichier n'existe pas. Là, on crée un dictionnaire vide, aucun score n'a été trouvé.
- Le joueur n'est pas dans le dictionnaire. Dans ce cas, on l'ajoute avec un score de `0`.

À vous de jouer

Vous avez l'essentiel. Peut-être pas tout ce dont vous avez besoin, cela dépend de comment vous vous organisez, mais le but est aussi de chercher ! Encore une fois, c'est un exercice pratique, ne sautez pas à la correction tout de suite, cela ne vous apprendra pas grand chose.

Bonne chance !

Correction proposée

Voici la correction que je vous propose. J'espère que vous êtes arrivés à un résultat satisfaisant, même si vous n'avez pas forcément réussi à tout faire. Si votre jeu marche, c'est parfait !

Télécharger les fichiers

Voici le code des trois fichiers.

`donnees.py`

```
"""Ce fichier définit quelques données, sous la forme de variables,

utiles au programme pendu"""

# Nombre de coups par partie

nb_coups = 8

# Nom du fichier stockant les scores

nom_fichier_scores = "scores"

# Liste des mots du pendu

liste_mots = [

    "armoire",

    "boucle",
```

```
    "buisson",
    "bureau",
    "chaise",
    "carton",
    "couteau",
    "fichier",
    "garage",
    "glace",
    "journal",
    "kiwi",
    "lampe",
    "liste",
    "montagne",
    "remise",
    "sandale",
    "taxi",
    "vampire",
    "volant",
]
```

fonctions.py

```
"""Ce fichier définit des fonctions utiles pour le programme pendu.
On utilise les données du programme contenues dans donnees.py"""
import os
import pickle
from random import choice
from donnees import *
# Gestion des scores
```

```
def recup_scores():
    """Cette fonction récupère les scores enregistrés si le fichier existe.
    Dans tous les cas, on renvoie un dictionnaire,
    soit l'objet dépicklé,
    soit un dictionnaire vide.
    On s'appuie sur nom_fichier_scores défini dans donnees.py"""
    if os.path.exists(nom_fichier_scores): # Le fichier existe
        # On le récupère
        fichier_scores = open(nom_fichier_scores, "rb")
        mon_depickler = pickle.Unpickler(fichier_scores)
        scores = mon_depickler.load()
        fichier_scores.close()
    else: # Le fichier n'existe pas
        scores = {}
    return scores
def enregistrer_scores(scores):
    """Cette fonction se charge d'enregistrer les scores dans le fichier
    nom_fichier_scores. Elle reçoit en paramètre le dictionnaire des scores
    à enregistrer"""
    fichier_scores = open(nom_fichier_scores, "wb") # On écrase les anciens
scores
    mon_pickler = pickle.Pickler(fichier_scores)
    mon_pickler.dump(scores)
    fichier_scores.close()


# Fonctions gérant les éléments saisis par l'utilisateur
def recup_nom_utilisateur():
    """Fonction chargée de récupérer le nom de l'utilisateur.
    Le nom de l'utilisateur doit être composé de 4 caractères minimum,
```

```
    chiffres et lettres exclusivement.
    Si ce nom n'est pas valide, on appelle récursivement la fonction
    pour en obtenir un nouveau"""
    nom_utilisateur = input("Tapez votre nom: ")
    # On met la première lettre en majuscule et les autres en minuscules
    nom_utilisateur = nom_utilisateur.capitalize()
    if not nom_utilisateur.isalnum() or len(nom_utilisateur)<4:
        print("Ce nom est invalide.")
        # On appelle de nouveau la fonction pour avoir un autre nom
        return recup_nom_utilisateur()
    else:
        return nom_utilisateur
def recup_lettre():
"""Cette fonction récupère une lettre saisie par
    l'utilisateur. Si la chaîne récupérée n'est pas une lettre,
    on appelle récursivement la fonction jusqu'à obtenir une lettre"""
    lettre = input("Tapez une lettre: ")
    lettre = lettre.lower()
    if len(lettre)>1 or not lettre.isalpha():
        print("Vous n'avez pas saisi une lettre valide.")
        return recup_lettre()
    else:
        return lettre
# Fonctions du jeu de pendu
def choisir_mot():
    """Cette fonction renvoie le mot choisi dans la liste des mots
    liste_mots.
    On utilise la fonction choice du module random (voir l'aide)."""
    return choice(liste_mots)
```

```
def recup_mot_masque(mot_complet, lettres_trouvees):
    """Cette fonction renvoie un mot masqué tout ou en partie, en fonction :
    - du mot d'origine (type str)
    - des lettres déjà trouvées (type list)
    On renvoie le mot d'origine avec des * remplaçant les lettres que l'on
    n'a pas encore trouvées."""
    mot_masque = ""
    for lettre in mot_complet:
        if lettre in lettres_trouvees:
            mot_masque += lettre
        else:
            mot_masque += "*"
    return mot_masque
```

pendu.py

```
"""Ce fichier contient le jeu du pendu.
Il s'appuie sur les fichiers :
- donnees.py
- fonctions.py"""
from donnees import *
from fonctions import *
# On récupère les scores de la partie
scores = recup_scores()
# On récupère un nom d'utilisateur
utilisateur = recup_nom_utilisateur()
# Si l'utilisateur n'a pas encore de score, on l'ajoute
if utilisateur not in scores.keys():
    scores[utilisateur] = 0 # 0 point pour commencer
# Notre variable pour savoir quand arrêter la partie
```

```
continuer_partie = 'o'

while continuer_partie != 'n':

    print("Joueur {0}: {1} point(s)".format(utilisateur, scores[utilisateur]))

    mot_a_trouver = choisir_mot()

    lettres_trouvees = []

    mot_trouve = recup_mot_masque(mot_a_trouver, lettres_trouvees)

    nb_chances = nb_coups

    while mot_a_trouver!=mot_trouve and nb_chances>0:

        print("Mot à trouver {0} (encore {1} chances)".format(mot_trouve,
nb_chances))

        lettre = recup_lettre()

        if lettre in lettres_trouvees: # La lettre a déjà été choisie

            print("Vous avez déjà choisi cette lettre.")

        elif lettre in mot_a_trouver: # La lettre est dans le mot à trouver

            lettres_trouvees.append(lettre)

            print("Bien joué.")

        else:

            nb_chances -= 1

            print("... non, cette lettre ne se trouve pas dans le mot...")

        mot_trouve = recup_mot_masque(mot_a_trouver, lettres_trouvees)

    # A-t-on trouvé le mot ou nos chances sont-elles épuisées ?

    if mot_a_trouver==mot_trouve:

    print("Félicitations ! Vous avez trouvé le mot {0}.".format(mot_a_trouver))

    else:

        print("PENDU !!! Vous avez perdu.")

    # On met à jour le score de l'utilisateur

    scores[utilisateur] += nb_chances

    continuer_partie = input("Souhaitez-vous continuer la partie (O/N) ?")

    continuer_partie = continuer_partie.lower()
```

```
# La partie est finie, on enregistre les scores

enregistrer_scores(scores)

# On affiche les scores de l'utilisateur

print("Vous finissez la partie avec {0} points.".format(scores[utilisateur]))
```

Résumé

Dans l'ensemble, je ne pense pas que le code soit très délicat à comprendre. Vous pouvez vous rendre compte à quel point le code du jeu est facile à lire grâce à nos fonctions. On délègue une partie de l'application à nos fonctions qui s'assurent que les choses sont « bien faites ». Si un bug survient, il est plus facile de modifier une fonction que tout un code sans aucune structure.

Par cet exemple, j'espère que vous prendrez bien l'habitude de documenter un maximum vos fichiers et fonctions. C'est réellement un bon réflexe à avoir.

N'oubliez pas la spécification de l'encodage en tête de chaque fichier, ni la mise en pause du programme sous Windows.