13. Application : le jeu du pendu

Pour terminer le module In121, nous allons réinvestir les notions de Python acquises cette année en les appliquant à la réalisation d'un jeu du *pendu*. Il s'agit du célèbre jeu dans lequel un joueur doit deviner un mot dont il ne connaît initialement que la longueur. Il doit alors proposer des lettres les unes après les autres. À chaque proposition, si la lettre fait partie du mot à deviner alors on dévoile tous les emplacements où elle apparaît dans le mot, sinon on inflige une pénalité au joueur. La partie se termine lorsque le mot caché a été trouvé, ou lorsque trop de pénalités ont été accumulées.

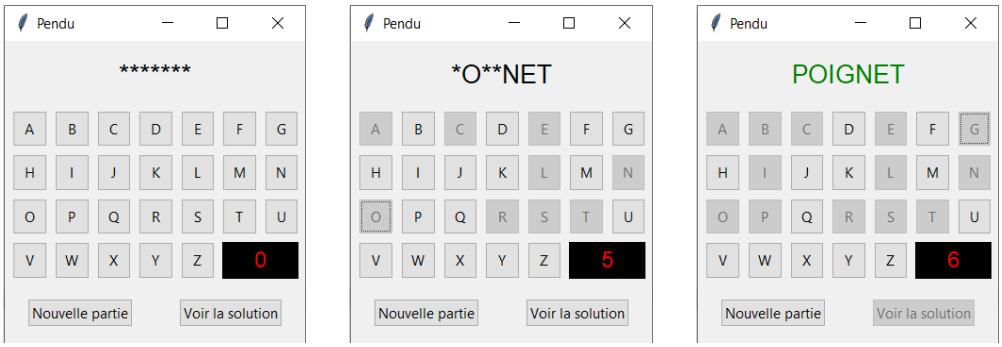


FIGURE 13.1. – Exemple d'interface graphique proposant une partie de *pendu*.

Ce projet sera découpé en deux parties :

1. d'abord, vous allez créer un module `Pendu` qui contiendra toutes les fonctions nécessaires au déroulement d'une partie de *pendu*,
2. ensuite, vous créerez une interface graphique qui utilisera les fonctions de ce module pour proposer à l'utilisateur une partie interactive de *pendu*.

13.1. Écriture du module

13.1.1. Cahier des charges

Votre module `Pendu.py` devra implémenter les fonctions suivantes.

— **`get_random_word`** – Tire au hasard un mot dans un « dictionnaire », c'est-à-dire un fichier texte contenant plusieurs mots (un mot par ligne). Vous utiliserez l'algorithme décrit dans la sous-section suivante.

Argument :

— `filename` (str) : le chemin vers le « dictionnaire ».

Valeur de retour : (str) un mot du dictionnaire.

— **`get_masked_word`** – Renvoie une version « masquée » du mot à trouver, c'est-à-dire dans laquelle les lettres qui n'ont pas encore été trouvées sont remplacées par des astérisques.

Arguments :

— `secret_word` (str) : le mot à trouver,

— `found_letters` (set) : un ensemble contenant les lettres qui ont déjà été trouvées par le joueur.

Valeur de retour : (str) le mot masqué.

Exemple : `get_masked_word('POIGNET', {'E', 'O', 'N', 'T'})` doit renvoyer `'*O**NET'`.

— **`check_letter`** – Teste si une lettre donnée appartient au mot ou pas, et met à jour les lettres trouvées par le joueur le cas échéant.

Arguments :

— `c` (str) : la lettre testée,

— `secret_word` (str) : le mot à trouver,

— `found_letters` (set) : un ensemble contenant les lettres qui ont déjà été trouvées par le joueur.

Si la lettre testée `c` appartient au mot à trouver, alors la fonction `check_letter` doit l'ajouter à l'ensemble `found_letters`.

Valeur de retour : (boolean) `True` si la lettre testée appartient au mot à trouver, `False` sinon.

— **`is_word_found`** – Teste si le mot secret a été totalement trouvé par le joueur ou non.

Arguments :

— `secret_word` (str) : le mot à trouver,

— `found_letters` (set) : un ensemble contenant les lettres qui ont déjà été trouvées par le joueur.

Valeur de retour : (boolean) `True` si le joueur a trouvé toutes les lettres du mot à trouver, `False` s'il lui reste encore des lettres à trouver.

Algorithme de sélection aléatoire

Le fichier `dico.txt` mis à votre disposition sur Moodle contient plus de 800 mots de la langue française. Tant que vous ne savez pas combien de mots contient ce fichier *exactement*, une façon « naïve » de choisir un mot au hasard dans cette liste à l'aide de Python serait la suivante :

— on parcourt toutes les lignes du fichiers une première fois afin de connaître le nombre N exact de mots dans le fichier,

— on choisit un entier k au hasard entre 1 et N (par exemple à l'aide de la bibliothèque `random`),

— on parcourt à nouveau les k premières lignes du fichiers de sorte à retrouver le k-ième mot du fichier.

L'inconvénient de cet algorithme est que l'on doit parcourir deux fois le fichier : une première fois pour savoir combien de mots il contient, et une seconde fois pour aller récupérer le mot choisi. Pour pallier ce problème, on pourrait envisager de stocker les mots dans une liste `L` lors du premier passage, afin de simplement récupérer l'élément `L[j]` ensuite sans avoir besoin de re-parcourir le fichier. Mais ce nouvel algorithme serait alors bien trop gourmand en mémoire : on stocke une liste de N mots alors qu'un seul d'entre eux nous intéresse. Cela sera d'autant moins pertinent que N sera grand.

Voici plutôt un algorithme qui permet de choisir de manière équiprobable un mot dans un fichier en un seul parcours, et ce sans avoir besoin de connaître à l'avance le nombre N de mots dans le fichier ni de stocker temporairement plusieurs mots.

1. On se donne une variable `word` qui ne contiendra tout du long de l'algorithme qu'une seule chaîne de caractères.

2. On parcourt le fichier ligne par ligne : quand on lit le i-ième mot, on tire au hasard (et de manière équiprobable) un nombre entre 1 et i (inclus). Si ce nombre vaut 1, on remplace le contenu de la variable `word` par le mot en cours. Sinon on poursuit l'algorithme sans rien faire de particulier.
3. À la fin de l'algorithme, on regarde quel est le mot qui reste dans la variable `word`.

Exemple. Voici un exemple de réalisation de l'algorithme.

1. Le 1er mot du fichier est ANGLE. On tire un entier au hasard entre 1 et 1. On trouve 1, on stocke donc ANGLE dans la variable `word`.
2. Le 2ème mot du fichier est ARMOIRE. On tire un entier au hasard entre 1 et 2. On trouve 1. On stocke donc ARMOIRE dans la variable `word` (à la place de ANGLE).
3. Le 3ème mot du fichier est BANC. On tire un entier au hasard entre 1 et 3. On trouve 3. On laisse la variable `word` telle quelle (elle contient donc toujours ARMOIRE).
4. Le 4ème mot du fichier est BUREAU. On tire un entier au hasard entre 1 et 4. On trouve 2. On laisse la variable `word` telle quelle (elle contient donc toujours ARMOIRE).
5. Le 5ème mot du fichier est CABINET. On tire un entier au hasard entre 1 et 5. On trouve 1. On stocke donc CABINET dans la variable `word` (à la place de ARMOIRE).
6. etc.

On poursuit ainsi l'algorithme et on regarde quel mot est stocké dans la variable `word` une fois qu'on a parcouru tous les mots du fichier.

Remarque. Il s'agit d'un cas particulier d'un algorithme appelé *reservoir sampling*, qui permet plus généralement de sélectionner k mots dans une liste ou un fichier formé de N mots (dont on ne connaît pas la taille N à l'avance).

Ce n'est pas forcément évident au premier abord, mais cet algorithme offre à tous les mots la même chance d'être obtenu à la fin. Le lecteur curieux trouvera ci-dessous une démonstration de ce fait à l'aide des outils de MA121.

Démonstration. Notons N le nombre de mots dans le fichier, et pour tout i compris entre 1 et N, notons X_i le nombre entre 1 et i qui a été tiré au hasard lors de la lecture du i-ème mot. On a :

$$\mathbb{P}(X_i = 1) = \frac{1}{i} \quad \text{et} \quad \mathbb{P}(X_i \neq 1) = \frac{i-1}{i}.$$

Pour que le k-ième mot du fichier soit retenu, il faut avoir obtenu 1 lors du k-ième tirage, et ne pas avoir obtenu 1 sur tous les tirages suivants (sinon le mot aurait été remplacé). La probabilité que cet événement se produise est égale à

$$p = \mathbb{P}([X_k = 1] \cap [X_{k+1} \neq 1] \cap [X_{k+2} \neq 1] \cap \cdots \cap [X_N \neq 1]).$$

En supposant que les tirages sont mutuellement indépendants, cette probabilité est alors égale à

$$p = \mathbb{P}(X_k = 1) \times \prod_{i=k+1}^{N} \mathbb{P}(X_i \neq 1) = \frac{1}{k} \times \prod_{i=k+1}^{N} \frac{i-1}{i}.$$

En reconnaissant un produit télescopique, on trouve

$$p = \frac{1}{k} \times \frac{k}{N} = \frac{1}{N}.$$

Ainsi le k-ième mot a une chance sur N d'être retenu à la fin de l'algorithme.

13.2. Interface graphique

Dans un fichier nommé `main.py` (et situé dans le même répertoire que votre fichier `Pendu.py`), vous importerez votre module Pendu et le module Tkinter afin de créer une interface graphique permettant de jouer des parties de *pendu*.

Vous pouvez vous inspirer de l'interface présentée dans la figure 13.1 si besoin, mais rien ne vous oblige à respecter la même disposition ou le même style. La seule contrainte est que votre interface propose les fonctionnalités suivantes :

— une façon simple et intuitive pour l'utilisateur de proposer des lettres une par une,

— un affichage du mot masqué et des pénalités accumulées, tous les deux mis à jour au fur et à mesure des propositions de l'utilisateur,

— la possibilité d'afficher immédiatement la solution (par exemple via un bouton),

— la possibilité de démarrer une nouvelle partie avec un nouveau mot tiré au hasard (par exemple via un bouton).

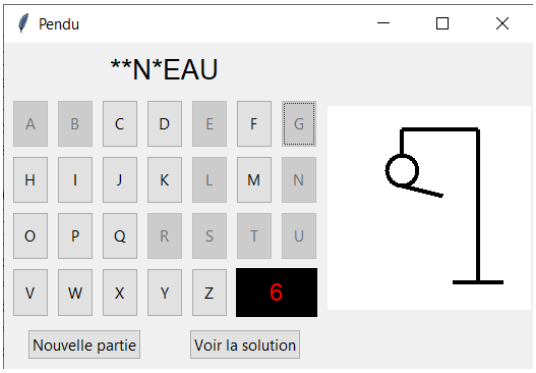
Astuce. Si vous créez un grand nombre de boutons qui doivent appeler des *callbacks* différents mais variant peu d'un bouton à l'autre, vous pouvez éviter de définir chacun des ces *callbacks* à la main en créant plutôt une unique fonction qui s'occupera de créer et renvoyer ces *callbacks* (voir section 8.2, paragraphe *Une fonction peut renvoyer une fonction*).

```
import tkinter as tk
import tkinter.ttk as ttk

def make_callback(S):
    ''' Cette fonction crée et renvoie une nouvelle fonction (callback)
    dépendant de S '''

    def callback():
        ''' Le callback dépendant de S qui sera renvoyé '''
        label["text"] = "Vous avez cliqué sur " + S

    return callback

window = tk.Tk()

label = ttk.Label(window, text="Cliquez quelque part")
label.pack()

for name in "UN", "DEUX", "TROIS", "QUATRE", "CINQ":
    btn = ttk.Button(window, text=name, command=make_callback(name))
    btn.pack()

window.mainloop()
```

Exercices supplémentaires

Pour les plus rapides, voici quelques pistes que vous pouvez explorer pour pousser votre projet plus loin (libre à vous d'inventer les vôtres au gré de votre imagination).

▷ **Affichage visuel des pénalités.**
Utilisez le widget `Canvas` (documentation) pour représenter visuellement le nombre de pénalités en dessinant pas à pas le célèbre personnage « pendu » ayant donné son nom au jeu.



▷ **Choix du dictionnaire.**
Laissez le choix à l'utilisateur du dictionnaire de mots utilisés pour la partie (par exemple via un menu déroulant). Vous pourriez par exemple proposer plusieurs langues, ou encore des dictionnaires à thème.

▷ **Sauvegarde des meilleurs scores.**
Lorsque l'utilisateur trouve le mot, proposez-lui de saisir son nom et enregistrer son résultat dans le tableau des scores. Le tableau des scores pourra par exemple être un fichier CSV stocké avec les autres fichiers de ce projet, que votre programme ira lire et modifier pour afficher les meilleurs scores ou ajouter des nouveaux joueurs.

▷ **D'autres jeu.**
Inspirez vous de l'organisation de votre programme imposée dans ce projet pour vous attaquer à la création d'autres jeux de lettres, comme par exemple les jeux télévisés Motus ou encore Des chiffres et des lettres.