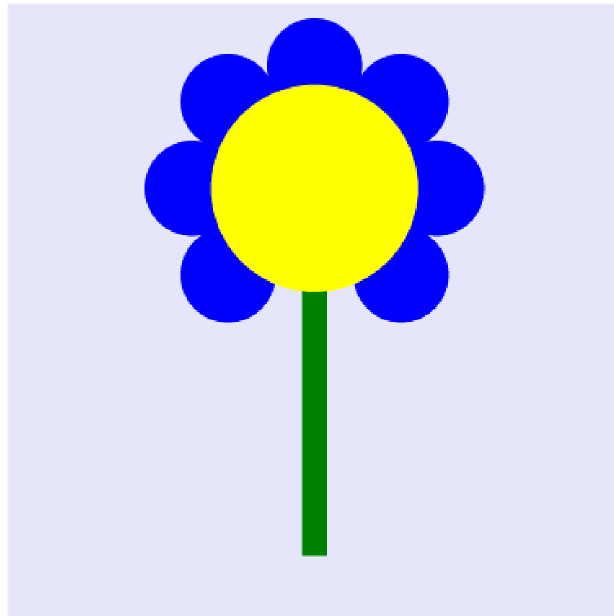
Pascal ORTIZ



Le jeu du Pendu

Version du 19 juin 2019

Licence CC-BY

Table des matières

Présentation du jeu 2
Prérequis et méthodologie 2
Une interface minimale 3
Connecter les lettres et la zone de texte 4
Une première version 7
Gérer la défaite et la victoire 10
Quelques améliorations 13
Faire apparaître le score 15
Un bouton pour rejouer 17
Placement des widgets dans la version définitive 20
Synchroniser le score et les images 24
La version finale 25

Présentation du jeu

Il s'agit de coder en Tkinter le classique jeu du Pendu sauf que chaque essai entraîne la perte d'un pétale dans une fleur jusqu'à ce que la fleur n'ait plus de pétales :

Toute lettre qui a été choisie devient grisée et donc inaccessible. On peut rejouer à tout instant de la partie en cliquant sur le bouton *Nouveau*. Lorsque la partie est terminée, la fleur produit un smiley (sourire ou grimace selon l'issue).

Malgré son apparente simplicité, le projet nécessite un certain investissement et une bonne coordination entre les différentes tâches à accomplir.

Les dessins qui traduisent le score sont des images au format png. Il est souvent moins laborieux d'utiliser des images que de dessiner soi-même avec Tkinter (bien qu'en fait les images ici aient été produites avec Tkinter).

Le code Python, les images et le fichier de mots sont dans ce fichier zip.

Prérequis et méthodologie

Pour coder le jeu du Pendu tel que je vous le présente là, il est absolument indispensable d'avoir pratiqué plusieurs dizaines d'heures les bases de la programmation impérative en Python et d'être très à l'aise avec les booléens, les listes, les instructions de contrôle (instructions if/elif/else, boucle for, imbrication de structures de contrôle), les fonctions et le découpage en tâches. Il faut avoir pratiqué la mise en code de petits algorithmes. Il est nécessaire d'avoir un minimum de familiarité avec les chaînes.

Concernant Tkinter, il faut aussi avoir les bases :

- principe de la programmation événementielle
- fenêtres,
- placement de widgets avec les méthodes pack et grid
- widgets bouton, label, canevas et placement d'images
- savoir modifier une option d'un widget
- les événements de la souris

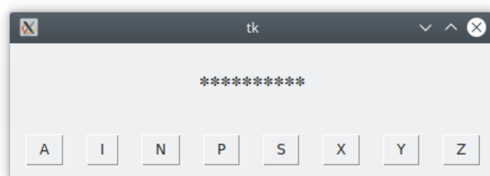
Cependant, ici, pas besoin de connaître la méthode `after` ni les variables de contrôle Tkinter.

La méthode que je vous conseille est celle que j'ai employée ci-dessous. Elle consiste à construire l'application

- entité par entité, de manière aussi indépendante que possible
- à dissocier autant que possible les élements visuels et les éléments du ressort de la logique du jeu
- en simplifiant au maximum et en ne gardant que les éléments indispensables au **fonctionnement** de l'application
- en ignorant au départ tous les aspects esthétiques
- en reliant les entités entre elles au fur et à mesure
- en testant systématiquemet étape par étape
- en n'incorporant qu'à la fin les fonctionnalités de confort et les améliorations esthétiques.

Une interface minimale

Un jeu du Pendu fonctionnel nécessite au minimum des touches pour choisir les lettres et une zone où le mot apparaît avec ses différentes lettres, étoilées si encore inconnues, découvertes sinon :



Pour simplifier, on va limiter le choix des lettres à

A, I, N, P, S, X, Y et Z.

Le mot à découvrir sera systématiquement SAPINS.

La zone où le mot en construction apparaît sera un widget de type Label. Les 8 lettres à choisir sont placées dans des boutons (classe Button). Dans un premier temps, je ne vais pas particulièrement soigner l'interface. Je vais utiliser la méthode `pack` de placement des widgets dans une fenêtre, elle est facile à utiliser.

Voici le code de l'interface ci-dessus :

lettres.py

```
1 from tkinter import *
2 root = Tk()
3
4 lbl = Label(root, text="**********", font="Times 15 bold")
5 lbl.pack(padx=20, pady=20)
6
7 ALPHA = "AINPSXYZ"
```

```
8
9 for c in ALPHA:
10     btn = Button(root, text=c)
11     btn.pack(side=LEFT, pady=10, padx=10)
12
13 root.mainloop()
```

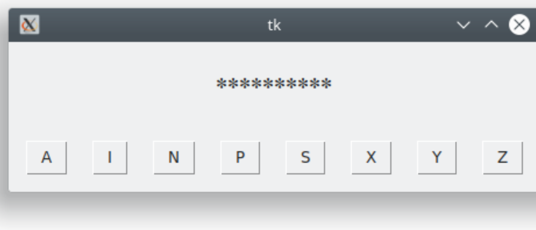
— Lignes 9-11 : on crée autant de boutons que contient notre alphabet, ici limité à 8 lettres. Bien sûr, on utilise une boucle `for` (imaginez qu'on ait 26 lettres !).
— On crée chaque widget (ligne 10) puis on les place avec la méthode pack (ligne 11).
— Ligne 4 : j'ai rempli la zone de texte avec un nombre arbitraire (10) d'étoiles.
— Ligne 4 : j'ai choisi la fonte et les paddings pour que l'interface soit à peu près visible.
— Ligne 11 : l'option `side=LEFT` de pack permet de placer chaque bouton à gauche du suivant plutôt que de les empiler les uns sur les autres.

C'est une interface complètement statique, une coquille vide. Cliquer sur un bouton n'a aucune action et la zone de texte est fixe.

Connecter les lettres et la zone de texte

Il faut maintenant que les boutons puissent réagir à la lettre qu'ils représentent. Dans un premier temps, écrivons un code qui réagit (mal) à l'appui sur un bouton : le clic sur un bouton affiche dans la console un message, ici juste **clic bouton** :

```
clic bouton
clic bouton
clic bouton
clic bouton
```



lettres_clic1.py

```
1 from tkinter import *
2 root = Tk()
3
4
5 def choisir_lettre(event):
6     print("clic bouton")
7
8
9 lbl = Label(root, text="**********", font="Times 15 bold")
10 lbl.pack(padx=20, pady=20)
11
12 ALPHA = "AINPSXYZ"
13
```

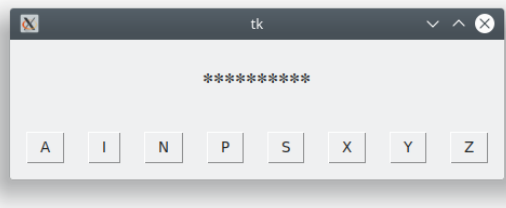
```
14 for c in ALPHA:
15     btn = Button(root, text=c)
16     btn.pack(side=LEFT, pady=10, padx=10)
17     btn.bind("<Button-1>", choisir_lettre)
18
19 root.mainloop()
```

— Ligne 17 : le clic (événement `<Button-1>`) sur un bouton entraîne l'exécution automatique de la fonction `choisir_lettre` qui affiche un message dans la console.

— Ligne 5 : la fonction `choisir_lettre` reçoit automatiquement en argument l'événement `event` représentant le clic sur le bouton.

Toutefois, la connexion de chaque bouton à la zone de texte est inexistante. La difficulté est de faire en sorte que la fonction `choisir_lettre` reconnaisse **le** bouton qui a été cliqué. Il se trouve que ce bouton est un widget détectable via l'attribut `widget` de l'événement `event` transmis à la fonction lors du clic. D'où le code suivant :

```
clic bouton : X
clic bouton : Y
clic bouton : X
clic bouton : N
clic bouton : A
```



lettres_clic2.py

```
1 from tkinter import *
2 root = Tk()
3
4
5 def choisir_lettre(event):
6     mon_btn = event.widget
7     texte = mon_btn["text"]
8     print("clic bouton : " + texte)
9
10
11 lbl = Label(root, text="**********", font="Times 15 bold")
12 lbl.pack(padx=20, pady=20)
13 ALPHA = "AINPSXYZ"
14
15 for c in ALPHA:
16     btn = Button(root, text=c)
17     btn.pack(side=LEFT, pady=10, padx=10)
18     btn.bind("<Button-1>", choisir_lettre)
19
20 root.mainloop()
```

— Ligne 6 : pour chaque clic sur un bouton, on capture le bouton à l'origine du clic.

— Ligne 7 : Tkinter permet toujours d'accéder au contenu d'une option d'un widget. Par exemple ici, avec ce bouton, on peut accéder au texte que porte le bouton grâce à l'option `text`
— Ligne 8 : le texte est ensuite affichée dans la console.

Maintenant, on voudrait que le bouton sur lequel on a cliqué et la zone de texte interagissent. Pour cela, on va choisir au hasard une position dans la zone de texte et on va y afficher la lettre. Mais avant cela, il faut avoir en tête qu'on a besoin d'une structure de données qui enregistre le mot en construction. Comme ce mot est en évolution, on va placer ses lettres dans une liste, car une liste est modifiable alors qu'une chaîne ne l'est pas. Initialement la liste sera une liste formée de 6 caractères `*` :

print_lettres.py

```
n=6
mot_en_progres=list("*"*n)

print(mot_en_progres)
```

```
['*', '*', '*', '*', '*', '*']
```

Par ailleurs, ce mot va devoir être placé dans le label. Pour cela, le mot doit être une chaîne de caractères et pas une liste. La façon usuelle en Python de transformer une liste de caractères en une chaîne est d'utiliser la méthode `join` illustrée ci-dessous :

methode_join.py

```
1 syllabes = ["PRO", "GRA", "MA", "TION"]
2 mot = "-".join(syllabes)
3 print(mot)
4
5 mot_en_progres = ['S', 'A', 'P', 'I', '*', 'S']
6 chaine = "".join(mot_en_progres)
7 print(chaine)
```

```
8 PRO-GRA-MA-TION
9 SAPI*S
```

— La méthode `join` est appliquée au séparateur, dans le premier cas, le trait d'union (ligne 2), dans le 2e cas (ligne 6), la chaîne vide puiqu'on veut juste rassembler les lettres.

Enfin, pour modifier le label, il suffit de changer son option `text`. D'où le code suivant :

lettres_visibles.py

```
1 from random import randrange
2 from tkinter import *
3 root = Tk()
4
5
6 def choisir_lettre(event):
7     mon_btn = event.widget
8     lettre = mon_btn["text"]
9     k = randrange(n)
10    mot_en_progres[k] = lettre
11    lbl["text"] = "".join(mot_en_progres)
```

```
12
13
14 n = 6
15 mot_en_progres = list("*" * n)
16 texte = "".join(mot_en_progres)
17
18 lbl = Label(root, text=texte, font="Times 15 bold")
19 lbl.pack(padx=20, pady=20)
20
21 ALPHA = "AINPSXYZ"
22
23 for c in ALPHA:
24     btn = Button(root, text=c)
25     btn.pack(side=LEFT, pady=10, padx=10)
26     btn.bind("<Button-1>", choisir_lettre)
27
28 root.mainloop()
```

— Ligne 14 : `n` désigne la longueur du mot
— Ligne 15 : le nombre d'étoiles est désormais `n`
— Ligne 9 : on choisit au hasard une position (l'indice `k`) dans la zone de texte.
— Ligne 10 : on écrit la lettre sur laquelle l'utilisateur a cliqué.

Une première version

Maintenant qu'on sait comment reconnaître une touche et écrire dans la zone de texte, on va introduire le mot secret et on va permettre à l'utilisateur de le découvrir mais sans limite de coups.

Quelque part dans le code, on place le mot secret et on construit le mot caché qui l'accompagne, par exemple :

```
secret = "SAPINS"
mot_en_progres = list("*" * len(secret))
stars = "".join(mot_en_progres)
```

Dans la suite, on va supposer que le mot secret est `SAPINS`. On va être confronté au problème suivant : le mot est en construction, par exemple, on le connait sous la forme incomplète suivante

```
mot_en_progres=['*', 'A', P, '*', 'N', '*']
```

Si le joueur tape sur une lettre, pour mettre à jour, le programme doit parcourir par indices de la gauche vers la droite les lettres du mot `secret` et examiner si une de ces lettres est la lettre qui a été tapée. Si la lettre est rencontrée alors le programme doit remplacer, **au même indice**, dans `mot_en_progres`, la lettre présente par la lettre tapée par l'utilisateur. Regardons en détail 3 cas :

— le joueur tape `S` ; le programme scanne `secret = "SAPINS"` et trouve deux positions où se trouve `S`, aux indices 0 et 5 ; le programme se rend aux indices 0 et 5 de `mot_en_progres` et y écrit `S` en sorte que `mot_en_progres` devient `[S, 'A', P, '*', 'N', S]` ;

— le joueur tape X ; comme précédemment, le programme va parcourir le mot secret SAPINS lettre par lettre et, bien sûr, ne va pas trouver la lettre X et il ne fera rien de plus ;
— le joueur tape P ; comme dans les cas précédents, le programme va parcourir le mot secret SAPINS ; il va la trouver et comme la lettre se trouve déjà dans mot_en_progres, le programme va remplacer la lettre par elle-même et au total le contenu de mot_en_progres sera inchangé. Bien sûr on pourrait, de plusieurs façons, élimiminer ce cas, mais en première approximation, on va garder la procédure ainsi.

Le code qui va mettre à jour le mot en recherche va être une fonction appelée maj_mot_en_progres et voici son code :

```
def maj_mot_en_progres(mot_en_progres, lettre, secret):
    n = len(secret)
    for i in range(n):
        if secret[i] == lettre:
            mot_en_progres[i] = lettre
```

Ce code réalise exactement ce qui a été indiqué ci-dessus. Un point toutefois : cette fonction reçoit une liste en argument (mot_en_progres) et cette liste est donc modifiée (dernière ligne de la fonction). La fonction ne renvoie rien (pas besoin), elle agit, comme on dit, *par effet de bord* en modifiant la liste. Ultérieurement, une version plus conforme à l'usage académique de Python sera donnée et il y sera précisé comment traiter le cas d'une lettre déjà trouvée.

Maintenant, lorsque le joueur tape sur une lettre, la fonction choisir_lettre va appeler la fonction maj_mot_en_progres qui va changer (si nécessaire) la liste mot_en_progres et ce changement sera répercuté dans la label de la zone à écrire. On obtient donc une version incomplète mais qui a toutes les bases d'une version jouable :

Voici le code :

```
1  from tkinter import *
2  root = Tk()
3
4
5  def maj_mot_en_progres(mot_en_progres, lettre, secret):
6      n = len(secret)
7      for i in range(n):
8          if secret[i] == lettre:
9              mot_en_progres[i] = lettre
10
11
12 def choisir_lettre(event):
13     mon_btn = event.widget
```

```
14     lettre = mon_btn["text"]
15     maj_mot_en_progres(mot_en_progres, lettre, secret)
16     lbl["text"] = "".join(mot_en_progres)
17
18
19 secret = "SAPINS"
20 mot_en_progres = list("*" * len(secret))
21 stars = "".join(mot_en_progres)
22
23 lbl = Label(root, text=stars, font="Times 15 bold")
24 lbl.pack(padx=20, pady=20)
25
26 ALPHA = "AINPSXYZ"
27
28 for c in ALPHA:
29     btn = Button(root, text=c)
30     btn.pack(side=LEFT, pady=10, padx=10)
31     btn.bind("<Button-1>", choisir_lettre)
32
33 root.mainloop()
```

— Quand le joueur clique sur une lettre, le code passe dans la fonction lignes 12-16. La fonction maj_mot_en_progres est appelée (ligne 15).

— Lignes 5-9 : la fonction maj_mot_en_progres n'a pas de retour mais le résultat de l'appel **modifie** l'état du mot en cours de recherche, ici mot_en_progres ligne 9.

— Une fois le mot modifié dans le code, il ne reste plus qu'à le faire apparaître dans l'interface graphique. Pour ça, on construit avec la méthode join la nouvelle chaîne de caractères (ligne 16, à droite de l'affectation) et on modifie le texte du label (ligne 16, à gauche de l'affectation).

Version plus pythonnique

Ce qui suit sera réservée à une 3e ou 4e lecture ... Si vous écrivez le code suivant :

```
1 def maj_mot_en_progres(mot_en_progres, lettre, secret):
2     n = len(secret)
3     for i in range(n):
4         if secret[i] == lettre:
5             mot_en_progres[i] = lettre
```

vous risquez de déclencher l'ire de certains puristes. En effet, vous parcourez les éléments d'un itérable (ici secret) par un indice (ici i), en écrivant secret[i] alors qu'en Python, cela peut se faire sans indice, par la tournure expliquée ci-dessous :

```
secret = "SAPINS"

for c in secret:
    print(c)
```

Il est vrai que lorsqu'on n'a pas besoin de l'indice, cette tournure est plus lisible. Le problème est qu'ici, on a **aussi** besoin de l'indice où la lettre a été rencontrée (parce qu'on doit remplacer cette lettre dans la liste). La façon pythonnique de faire est de considérer la fonction standard

enumerate qui va parcourir la liste en générant un couple entier + élément de la liste comme on le voit ci-dessous :

```
secret = "SAPINS"

for i, c in enumerate(secret):
    print(i, c)
```

```
0 S
1 A
2 P
3 I
4 N
5 S
```

La différence est que vous n'accédez pas à l'élément en faisant l'opération secret[i]. Dans ces conditions, le code peut se récrire :

```
def maj_mot_en_progres(mot_en_progres, lettre, secret):
    for i,c in enumerate(secret):
        if c == lettre:
            mot_en_progres[i] = lettre
```

C'est l'occasion de placer l'optimisation indiquée plus haut évitant de remplacer une lettre par elle-même ; il suffit de ne faire le changement que si la lettre n'est pas présente dans mot_en_progres à l'indice courant :

```
def maj_mot_en_progres(mot_en_progres, lettre, secret):
    for i,c in enumerate(secret):
        if c == lettre and mot_en_progres[i] != lettre:
            mot_en_progres[i] = lettre
```

Les puristes trouveront peut-être que ce code n'est pas assez pythonnique ; en effet, vous parcourez à nouveau une liste avec accès par indice ; en fait, on parcourt en parallèle les deux itérables secret et mot_en_progres et donc, habituellement, en Python, cela se fait en utilisant la fonction standard zip qui va permettre une sorte de progression synchronisée dans secret et mot_en_progres. D'où le code suivant :

```
def maj_mot_en_progres(mot_en_progres, lettre, secret):
    for i,(a, b) in enumerate(zip(secret, mot_en_progres)):
        if a == lettre != b:
            mot_en_progres[i] = lettre
```

très pythonnique et dont je vous laisse juge de la lisibilité ...

Gérer la défaite et la victoire

Maintenant, on voudrait que le code implémente l'annonce de la victoire ou de la défaite. L'annonce va se faire dans un label qu'on va placer juste en-dessous de la zone de texte :

```
lbl = Label(root, text=stars, font="Times 15 bold")
lbl.pack(padx=20, pady=20)
```

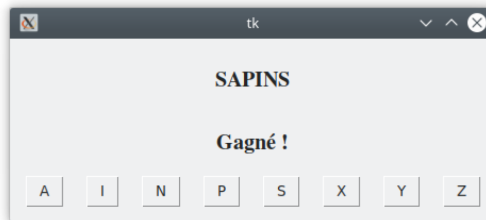
```
annonce = Label(root, width=8, font="Times 15 bold")
annonce.pack(padx=5, pady=5)
```

Cela agrandit légèrement la taille de la fenêtre.

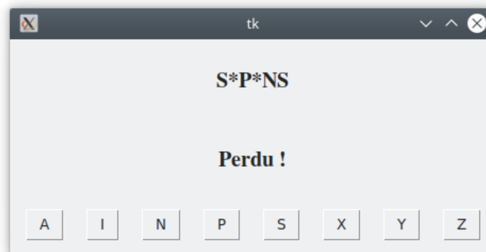
Pour annoncer juste la victoire, il suffit que le programme compare, chaque fois que le joueur a trouvé une nouvelle lettre, pour savoir s'il a obtenu le mot secret ou pas. De manière un peu simpliste mais acceptable, on peut effectuer directement la comparaison entre la liste `mot_en_progres` et le mot secret (transformé en liste), ce qui pourrait donner par exemple :

```
def maj_mot_en_progres(mot_en_progres, lettre, secret):
    n = len(secret)
    for i in range(n):
        if secret[i] == lettre:
            mot_en_progres[i] = lettre
            if mot_en_progres == list(secret):
                annonce["text"] = "Gagné !"
```

Si le joueur trouve la réponse alors une zone dans la fenêtre (le label `annonce`) affichera le message **Gagné** :



Pour la défaite, il faut créer un système de score. Pour l'exemple, on va supposer que le choix de **3** lettres non présentes entraîne la défaite. Pour gérer le score, on va créer une fonction `score`. On aura besoin d'un compteur `cpt` qui enregistre le nombre de choix incorrects. La fonction `score` surveillera la valeur de `cpt` et si cette valeur atteint `limite = 3`, le message de défaite sera écrit dans le label annonce :



Cela pourrait donner un code comme celui-ci :

```
1 def score(lettre):
2     global cpt
```

```
3     if lettre not in secret:
4         cpt += 1
5         if cpt >= limite:
6             annonce["text"] = "Perdu !"
7     elif mot_en_progres == list(secret):
8         annonce["text"] = "Gagné !"
```

— Si la lettre cliquée par le joueur n'est pas dans le mot inconnu (ligne 3), le compteur d'erreurs augmente de 1 (ligne 4) et s'il dépasse la limite (ligne 5), c'est que la partie est perdue (ligne 6).

— Sinon (ligne 7), c'est que le joueur a découvert une lettre ou choisi une lettre qui avait déjà été trouvée. Il se peut qu'il ait gagné et donc pour cela le joueur compare le mot en progrès et le mot inconnu.

— Ligne 2 : comme la fonction modifie le compteur `cpt` qui est défini en dehors de la fonction, `cpt` doit être déclaré en `global` (la suite va montrer qu'il n'est pas utile de placer `cpt` en paramètre comme on aurait pu croire).

La fonction `score` est appelée (ligne 6 ci-dessous) chaque fois qu'une lettre a été cliquée par le joueur :

```
def choisir_lettre(event):
    mon_btn = event.widget
    lettre = mon_btn["text"]
    maj_mot_en_progres(mot_en_progres, lettre, secret)
    lbl["text"] = "".join(mot_en_progres)
    score(lettre)
```

C'est là qu'on voit que si on donnait `cpt` en paramètre de `score`, il faudrait que `choisir_lettre` en dispose ce qui ne peut se faire (paramètres contraints ligne 1) qu'en utilisant `global` (non écrit).

Voici le code complet :

defaite.py

```
from tkinter import *
root = Tk()


def maj_mot_en_progres(mot_en_progres, lettre, secret):
    n = len(secret)
    for i in range(n):
        if secret[i] == lettre:
            mot_en_progres[i] = lettre


def score(lettre):
    global cpt
    if lettre not in secret:
        cpt += 1
        if cpt >= limite:
            annonce["text"] = "Perdu !"
```

```python
    elif mot_en_progres == list(secret):
        annonce["text"] = "Gagné !"


def choisir_lettre(event):
    mon_btn = event.widget
    lettre = mon_btn["text"]
    maj_mot_en_progres(mot_en_progres, lettre, secret)
    lbl["text"] = "".join(mot_en_progres)
    score(lettre)


secret = "SAPINS"
mot_en_progres = list("*" * len(secret))
stars = "".join(mot_en_progres)

lbl = Label(root, text=stars, font="Times 15 bold")
lbl.pack(padx=20, pady=20)

annonce = Label(root, width=8, font="Times 15 bold")
annonce.pack(padx=20, pady=20)

cpt = 0
limite = 3
ALPHA = "AINPSXYZ"

for c in ALPHA:
    btn = Button(root, text=c)
    btn.pack(side=LEFT, pady=10, padx=10)
    btn.bind("<Button-1>", choisir_lettre)
root.mainloop()
```

Quelques améliorations

Lorsque la partie est terminée, le jeu peut encore réagir à des clics sur les boutons alors qu'il devrait être bloqué. Pour supprimer ce comportement, il suffit de placer un drapeau end qui vaudra `False` quand on sera en fin de partie. Ce drapeau va empêcher la fonction `choisir_lettre` de réagir. Voici un code partiel :

```python
def score(lettre):
    global cpt, end
    if lettre not in secret:
        cpt += 1
        if cpt >= limite:
            annonce["text"] = "Perdu !"
            end = True
    elif mot_en_progres == list(secret):
        annonce["text"] = "Gagné !"
```

```
10          end = True
11
12
13 def choisir_lettre(event):
14     if end:
15         return
16     mon_btn = event.widget
17     lettre = mon_btn["text"]
18     maj_mot_en_progres(mot_en_progres, lettre, secret)
19     lbl["text"] = "".join(mot_en_progres)
20     score(lettre)
21
22
23 cpt = 0
24 limite = 3
25 end = False
```

- Ligne 25 : définition du drapeau
- La partie est terminée (victoire ligne 9 et défaite ligne 6) donc le drapeau est placé à `True` (lignes 7 et 10)
- Lignes 14-15 : lorsque la partie est terminée (`end = True`), la fonction (ligne 13) déclenchée par le clic sur une touche est désactivée (ligne 15).

Rappelons au passage qu'il existe une autre façon, plus appropriée, de désactiver le déclenchement d'une fonction suite à un événement, consistant à appeler la méthode unbind.

Enfin, on voudrait aussi qu'une touche de lettre qui a été choisie par le joueur soit désactivée pour les coups ultérieurs. Beaucoup de widgets de Tkinter peuvent être désactivés par le changement d'une option `state` que l'on place à `DISABLED`. Pour un bouton, cela a pour effet de griser la partie active du widget :

Pour cela, il suffit de le faire la première fois que la touche est cliquée :

```
def choisir_lettre(event):
    if end:
        return
    mon_btn = event.widget
    lettre = mon_btn["text"]

    # Le bouton est désactivé
```

```
    mon_btn["state"]=DISABLED

    maj_mot_en_progres(mot_en_progres, lettre, secret)
    lbl["text"] = "".join(mot_en_progres)
    score(lettre)
```

Faire apparaître le score

Au jeu du Pendu, le joueur est averti de combien d'étapes il lui reste avant d'avoir perdu. Il serait donc souhaitable de lui montrer par exemple le nombre d'erreurs commises :

Pour cela, on va faire évoluer le label qui annonce la victoire ou la défaite pour qu'il donne aussi le nombre de lettres manquées.
Il suffit de faire modifier la fonction de score. Voici un code partiel :

```
1 def score(lettre):
2     global cpt, end
3     if lettre not in secret:
4         cpt += 1
5         if cpt >= limite:
6             annonce["text"] = "Perdu !"
7             end = True
8         else:
9             annonce["text"] = cpt
10    elif mot_en_progres == list(secret):
11        annonce["text"] = "Gagné !"
12        end = True
13
14
15 annonce = Label(root, width=8, font="Times 15 bold", text=0)
```

— Le score n'évolue que si le joueur ne devine pas une lettre (ligne 3) et s'il n'a pas perdu (ligne 8), il suffit d'afficher le score (ligne 9).

— Ligne 15 : au départ, le score est à 0 (option `text=0`).

Un code complet est le suivant :

```python
from tkinter import *
root = Tk()


def maj_mot_en_progres(mot_en_progres, lettre, secret):
    n = len(secret)
    for i in range(n):
        if secret[i] == lettre:
            mot_en_progres[i] = lettre


def score(lettre):
    global cpt, end
    if lettre not in secret:
        cpt += 1
        if cpt >= limite:
            annonce["text"] = "Perdu !"
            end = True
        else:
            annonce["text"] = cpt

    elif mot_en_progres == list(secret):
        annonce["text"] = "Gagné !"
        end = True


def choisir_lettre(event):
    if end:
        return
    mon_btn = event.widget
    lettre = mon_btn["text"]

    # Le bouton est désactivé
    mon_btn["state"] = DISABLED

    maj_mot_en_progres(mot_en_progres, lettre, secret)
    lbl["text"] = "".join(mot_en_progres)
    score(lettre)


secret = "SAPINS"
mot_en_progres = list("*" * len(secret))
stars = "".join(mot_en_progres)

lbl = Label(root, text=stars, font="Times 15 bold")
lbl.pack(padx=20, pady=20)

annonce = Label(root, width=8, font="Times 15 bold", text=0)
```

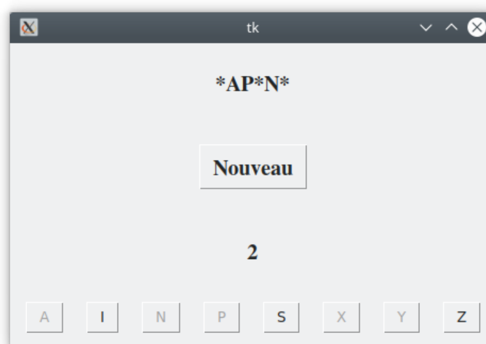
```
annonce.pack(padx=20, pady=20)

cpt = 0
limite = 3
end = False
ALPHA = "AINPSXYZ"

for c in ALPHA:
    btn = Button(root, text=c)
    btn.pack(side=LEFT, pady=10, padx=10)
    btn.bind("<Button-1>", choisir_lettre)
root.mainloop()
```

Un bouton pour rejouer

Le dernier changement d'importance à réaliser est de donner la possibilité au joueur de recommencer une partie à l'aide d'un bouton :



Pour cela, on va écrire une fonction `init` qui sera exécutée lors d'un clic du bouton. Cette fonction sera aussi exécutée au lancement du jeu. Ecrire ce genre de fonction oblige à bien réfléchir à la construction du jeu.
On va placer le bouton juste en-dessous de la zone de texte :

```
reset=Button(root, text="Nouveau", font="Times 15 bold", command=init)
reset.pack(padx=20, pady=20)
```

Il faut maintenant analyser quels sont les éléments communs à toutes les parties et les éléments qui changent à chaque nouvelle partie. La présence des widgets est invariable. L'initialisation des boutons de lettres est invariable aussi. De même les lettres de l'alphabet utilisées ainsi que la limite de défaite. Ce qui est variable :

— le mot à découvrir et donc l'initialisation de la zone de texte
— la zone de score
— le drapeau end
— la zone de texte

— l'état des boutons (désactivés)

Tous ces éléments vont être définis dans la fonction `init`. Et devront être déclarés `global` pour qu'ils soient accessibles depuis d'autres fonctions.

Dans l'exemple ci-dessous, le mot à découvrir va être conservé, ce qui permettra de tester le programme sans surprise. Il va falloir alléger l'initialisation de la zone de texte :

```
lbl = Label(root, font="Times 15 bold")
```

L'option `text` a disparu et sera construite dans la fonction `init`.

Pour réactiver tous les boutons, il faut y accéder. Or, on a gardé aucune trace des boutons dans les codes précédents. Il va donc falloir les placer dans une liste, d'où le nouveau code de définition de ces boutons :

```
btns = []

for c in ALPHA:
    btn = Button(root, text=c)
    btn.pack(side=LEFT, pady=10, padx=10)
    btn.bind("<Button-1>", choisir_lettre)
    btns.append(btn)
```

Dans la fonction `init`, la réinitilisation des boutons se fera comme suit :

```
for btn in btns:
    btn["state"] = NORMAL
```

Voici le code au complet :

rejouer.py

```
from tkinter import *
root = Tk()


def maj_mot_en_progres(mot_en_progres, lettre, secret):
    n = len(secret)
    for i in range(n):
        if secret[i] == lettre:
            mot_en_progres[i] = lettre


def score(lettre):
    global cpt, end
    if lettre not in secret:
        cpt += 1
        if cpt >= limite:
            annonce["text"] = "Perdu !"
            end = True
        else:
            annonce["text"] = cpt
```

```python
    elif mot_en_progres == list(secret):
        annonce["text"] = "Gagné !"
        end = True


def choisir_lettre(event):
    if end:
        return
    mon_btn = event.widget
    lettre = mon_btn["text"]

    # Le bouton est désactivé
    mon_btn["state"] = DISABLED

    maj_mot_en_progres(mot_en_progres, lettre, secret)
    lbl["text"] = "".join(mot_en_progres)
    score(lettre)


def init():
    global end, mot_en_progres, secret, cpt

    secret = "SAPINS"
    mot_en_progres = list("*" * len(secret))
    stars = "".join(mot_en_progres)
    lbl["text"] = stars
    for btn in btns:
        btn["state"] = NORMAL

    cpt = 0
    end = False


lbl = Label(root, font="Times 15 bold")
lbl.pack(padx=20, pady=20)

reset = Button(root, text="Nouveau", font="Times 15 bold", command=init)
reset.pack(padx=20, pady=20)

annonce = Label(root, width=8, font="Times 15 bold", text=0)
annonce.pack(padx=20, pady=20)

limite = 3
ALPHA = "AINPSXYZ"
btns = []

for c in ALPHA:
    btn = Button(root, text=c)
```

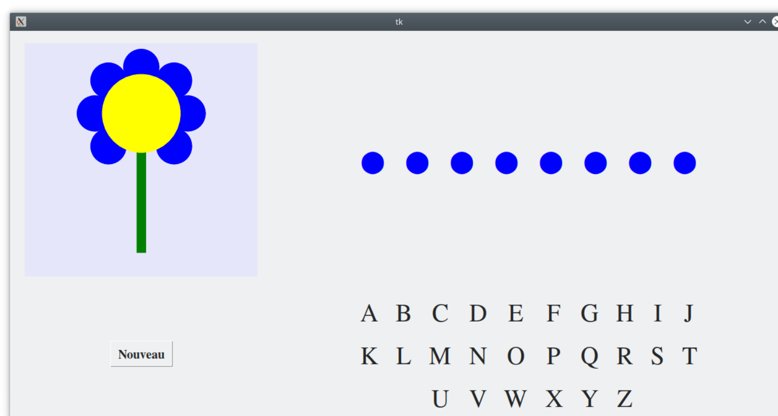
```
    btn.pack(side=LEFT, pady=10, padx=10)
    btn.bind("<Button-1>", choisir_lettre)
    btns.append(btn)

init()

root.mainloop()
```

Placement des widgets dans la version définitive

Tout ce qui suit utilise le gestionnaire `grid` de placement des widgets. Voici une copie d'écran de la version définitive :



Il y a quatre zones :

— les lettres de l'alphabet
— la zone de texte placée dans un label
— le dessin indiquant les coups manqués par le joueur et qui sera placé dans un canevas
— le bouton *Nouveau.*

Ces 4 zones s'organisent suivant une grille 2x2. On va donc utiliser le gestionnaire de placement `grid`.

Les boutons

Les lettres s'organisent en trois lignes de largeur 10, 10 et 6 (le total fait donc les 26 lettres de l'alphabet) et s'organisent suivant une grille 3 x 10. On va placer tous les boutons dans un frame appelé `lettres` qui lui même contiendra les 26 boutons organisés en grid.
Faisons un essai dans un fichier à part juste pour les 26 boutons :

Voici le code :

boutons_seuls.py

```
from tkinter import *

root = Tk()

lettres = Frame(root)
lettres.pack()

ALPHA = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

for i in range(2):
    for j in range(10):
        btn = Button(
            lettres,
            text=ALPHA[10 * i + j],
            relief=FLAT,
            font='times 30')
        btn.grid(row=i, column=j)

for j in range(6):
    btn = Button(
        lettres, text=ALPHA[20 + j], relief=FLAT, font='times 30')
    btn.grid(row=2, column=j + 2)

root.mainloop()
```

- Ligne 10 : le traitement de chacune des deux lignes de lettres.
- Ligne 11 : le traitement de chacun des 10 boutons de chaque ligne.
- Ligne 13 : chaque bouton est placé dans le frame `lettres`
- ligne 14 : on écrit la lettre sur le bouton
- ligne 15 : on rend le bouton discret
- Ligne 17 : le placement de chaque bouton dans les cases de la grille (indices commençant à 0)
- lignes 19-22 : le traitement des 6 lettres de la dernière ligne.

La zone de texte

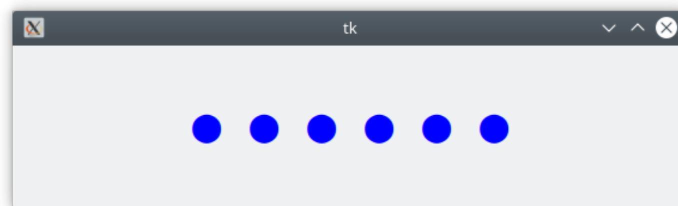
Le caractère qui va dissimuler les lettres ne sera pas une astérisque mais un caractère UTF-8 représentant un petit disque. Pour que les lettres ne soient pas trop serrées dans la zone de texte, on va placer un espace entre deux lettres ce qui s'obtient en utilisant la méthode `join` :

```
secret="SAPINS"
text=’ ’.join(""*len(secret))

print(text)
```

```
5
```

La zone de texte se présente ainsi :



Voici le code :

```
from tkinter import *

root = Tk()

lbl = Label(
    root, font=(’Deja Vu Sans Mono’, 30, ’bold’), width=23, fg="blue")
lbl.pack(pady=40)

secret = "SAPINS"
word_in_progress = list(’ ’.join("" * len(secret)))
lbl["text"] = ’’.join(word_in_progress)

root.mainloop()
```

La police a été choisie `Deja Vu Sans Mono` car c'est une police monospace et cela évite au label de gondoler en fonction de la taille des caractères présents. Par ailleurs, pour éviter un élargissement automatique du label, la largeur est fixée à 23 caractères donc des mots d'au plus 12 lettres (23=12+11).

Les images

Le score est représenté par le nombre de pétales d'une fleur. Au départ, il y a 7 pétales, pour 7 essais. Chaque fois que le joueur fait un essai erroné, la fleur perd un pétale. Les fleurs sont en fait des images au format png. Elle seront placées dans un canevas en haut à gauche de l'interface.

En regroupant tous les widgets précédents, on obtient le code d'une interface (non fonctionnelle) :

interface.py

```
from tkinter import *


def init():
    secret = "SAPINS"
    mot_en_progres = list('_'.join("" * len(secret)))
    lbl["text"] = ''.join(mot_en_progres)
    cnv.create_image((width_img / 2, height_img / 2), image=img)


root = Tk()

# Les images

img = PhotoImage(file="7.png")

width_img = img.width()
height_img = img.height()

cnv = Canvas(
    root, width=width_img, height=height_img, highlightthickness=0)
cnv.grid(row=0, column=0, padx=20, pady=20)

# La zone de texte

lbl = Label(
    root, font=('Deja Vu Sans Mono', 45, 'bold'), width=23, fg="blue")
lbl.grid(row=0, column=1)

# Rejouer

reset = Button(root, text="Nouveau", font="Times 15 bold", command=init)
reset.grid(row=1, column=0)

# Les boutons pour les lettres

lettres = Frame(root)
lettres.grid(row=1, column=1)

ALPHA = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

for i in range(2):
    for j in range(10):
        btn = Button(
            lettres,
            text=ALPHA[10 * i + j],
            relief=FLAT,
```

```
48             font='times 30')
49         btn.grid(row=i, column=j)
50
51 for j in range(6):
52     btn = Button(
53         lettres, text=ALPHA[20 + j], relief=FLAT, font='times 30')
54     btn.grid(row=2, column=j + 2)
55
56 init()
57
58 root.mainloop()
```

— Pour positionner exactement l'image dans le canevas (ligne 20), on détermine les dimensions de l'image (ligne 17-18) et elles deviennent les dimensions du canevas (ligne 20).

Synchroniser le score et les images

La seule partie du jeu qui n'ait pas été codée est l'interfaçage entre le score dans le jeu et le score représenté par les images.

Chaque fois que le joueur propose une lettre absente dans le mot, la fleur perd un pétale. Il y a 8 fichiers de fleurs, nommés `0.png`, `1.png`, etc, `7.png` où le nombre indique le nombre de pétales présents dans l'image. A la fin du jeu, on plaque un smiley sur l'image du score, le smiley étant généré par le fichier `win.png` ou `fail.png`. Les 10 images sont toutes placées à côté du fichier source Python du jeu.

Dès le début du jeu, on charge les images au format `PhotoImage` et on place dans une liste les images avec les pétales, l'image `0.png` étant à l'indice 0, l'image `1.png` étant à l'indice 1, et ainsi de suite. Le score est enregistré dans une variable globale `nro` valant 7 au départ. Chaque fois que le joueur fait une réponse incorrecte, l'image est effacée et remplacée par l'image dans la liste et d'indice `nro - 1`. Quand une partie commence, toutes les images du canevas sont effacées.

L'image de départ est `7.png`. Il faudra penser à la charger dans la fonction `init`. Le changement d'image est géré par la fonction `score` dont voici le code :

```
1  def score(lettre):
2      global nro, end, img
3      if lettre not in secret:
4          cnv.delete(images[nro])
5          nro -= 1
6          cnv.create_image((width_img / 2, height_img / 2),
7                           image=images[nro])
8          if nro == 0:
9              cnv.create_image((width_img / 2, height_img / 2),
10                              image=fail)
11             lbl["text"] = " ".join(secret)
12
13             end = True
14     elif mot_en_progres == list(" ".join(secret)):
15         cnv.create_image((width_img / 2, height_img / 2), image=win)
16         end = True
```

— Chaque fois que le joueur se trompe (ligne 3), l'image courante est effacée (ligne 4) et remplacée par celle ayant un pétale de moins (ligne 7).
— Lorsque la partie est terminée, un smiley de victoire (ligne 15) ou de défaite (ligne 10) est superposé à l'image courante.

La version finale

Dans la version qui suit, le mot secret est extrait au hasard d'un fichier `arbres.txt` contenant 160 noms d'arbres. Voici le code complet :

pendu.py

```
from tkinter import *
from random import randrange


def maj_mot_en_progres(mot_en_progres, lettre, secret):
    n = len(secret)
    for i in range(n):
        if secret[i] == lettre:
            mot_en_progres[2 * i] = lettre


def score(lettre):
    global nro, end, img
    if lettre not in secret:
        cnv.delete(images[nro])
        nro -= 1
        cnv.create_image((width_img / 2, height_img / 2),
                         image=images[nro])
        if nro == 0:
            cnv.create_image((width_img / 2, height_img / 2),
                             image=fail)
            lbl["text"] = " ".join(secret)

            end = True
    elif mot_en_progres == list(" ".join(secret)):
        cnv.create_image((width_img / 2, height_img / 2), image=win)
        end = True


def choisir_lettre(event):
    if end:
        return
    mon_btn = event.widget
    lettre = mon_btn["text"]
    mon_btn["state"] = DISABLED
    maj_mot_en_progres(mot_en_progres, lettre, secret)
    lbl["text"] = "".join(mot_en_progres)
```

```
38      score(lettre)
39
40
41  def init():
42      global end, mot_en_progres, secret, nro, img
43      secret = arbres[randrange(len(arbres))]
44      mot_en_progres = list(' '.join("" * len(secret)))
45      lbl["text"] = ''.join(mot_en_progres)
46      cnv.delete(ALL)
47      cnv.create_image((width_img / 2, height_img / 2), image=images[-1])
48
49      for btn in btns:
50          btn["state"] = NORMAL
51
52      nro = limite
53      end = False
54
55
56  root = Tk()
57
58  limite = 7
59
60  # Les images
61  images = [PhotoImage(file="%s.png" % j) for j in range(limite + 1)]
62
63  fail = PhotoImage(file="fail.png")
64  win = PhotoImage(file="win.png")
65
66  width_img = win.width()
67  height_img = win.height()
68
69  cnv = Canvas(
70      root, width=width_img, height=height_img, highlightthickness=0)
71  cnv.grid(row=0, column=0, padx=20, pady=20)
72
73  # La zone de texte
74
75  lbl = Label(
76      root, font=('Deja Vu Sans Mono', 45, 'bold'), width=23, fg="blue")
77  lbl.grid(row=0, column=1)
78
79  # Rejouer
80
81  reset = Button(root, text="Nouveau", font="Times 15 bold", command=init)
82  reset.grid(row=1, column=0)
83
84  # Les boutons pour les lettres
85
```

```
lettres = Frame(root)
lettres.grid(row=1, column=1)

ALPHA = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
btns = []

for i in range(2):
    for j in range(10):
        btn = Button(
            lettres,
            text=ALPHA[10 * i + j],
            relief=FLAT,
            font='times 30')
        btn.grid(row=i, column=j)
        btn.bind("<Button-1>", choisir_lettre)
        btns.append(btn)

for j in range(6):
    btn = Button(
        lettres, text=ALPHA[20 + j], relief=FLAT, font='times 30')
    btn.grid(row=2, column=j + 2)
    btn.bind("<Button-1>", choisir_lettre)
    btns.append(btn)

with open("arbres.txt") as f:
    arbres = f.read().split("\n")

init()

root.mainloop()
```

- Ligne 52 : `nro` représente en permanence le nombre de pétales de la fleur.
- Ligne 47 : en début de partie, la dernière image de la liste (d'indice -1 en python) est affichée, elle contient 7 pétales. Elle est placée au centre du canevas (les dimensions sont divisées par 2 dans le code ligne 47).
- Auparavant (ligne 46), les deux images de la partie précédente ont été effacées.
- Ligne 2 : on a besoin du module `random` pour tirer au hasard un des mots de la liste d'arbres (ligne 43)
- Comme on a placé des espaces entre deux lettres du mots à trouver (cf. ligne 44), il faut adapter le code pour le remplacement d'une lettre trouvée (ligne 9) : les lettres sont aux indices pairs (aux indices impairs, il y a des espaces).
- Si le joueur perd la partie (ligne 19), il faut penser à lui afficher la réponse attendue (ligne 22).
- En Python, il est d'usage d'utiliser une instruction `with` (ligne 110) pour ouvrir un fichier car la fermeture automatique du fichier est assurée, en particulier en cas d'exception. Il ne serait toutefois pas très gênant **ici** de remplacer les lignes 110-111 par

```
arbres = open("arbres.txt").read().split("\n")
```