ÉCOLE CENTRALE LYON

Informatique TC2 Rapport BE5

Groupe A2b

Réalisation du jeu du Pendu à l'aide de Python

Auteur :
M. Valentin VERDON

Encadrant :
M. Stéphane DERRODE

Version du
31 mars 2022

Table des matières

1 Introduction

Ce rapport présente la réalisation du jeu du Pendu à l'aide du langage *Python*. Nous ne développerons pas les détails de la réalisation de la base du jeu car elle a été réalisée en TD. Nous nous attarderons simplement sur la réalisation du diagramme UML du jeu ainsi que sur la réalisation de quelques options supplémentaires (personnalisation des couleurs du jeu, bouton "triche-retour").

2 Diagramme UML

Voici le diagramme UML du jeu :

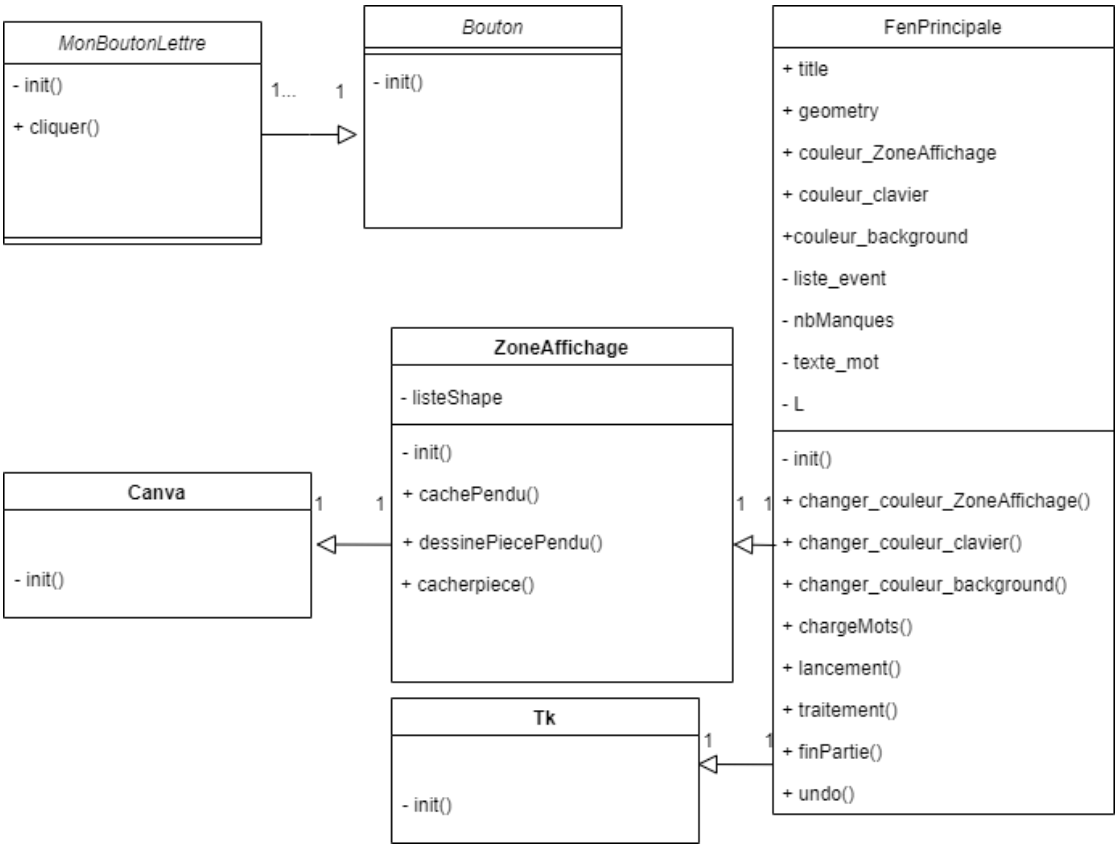

FIGURE 1 – Diagramme UML du jeu du Pendu

3 Amélioration du jeu

3.1 Première amélioration : personnalisation des couleurs

La première amélioration du jeu du Pendu consiste à pouvoir personnaliser les couleurs du jeu. Ici, je me suis penché sur la personnalisation de 3 espaces : le clavier, le background et la zone d'affichage du pendu.

Tout d'abord, nous utiliserons une fenêtre extérieure, déjà existante pour choisir notre couleur :

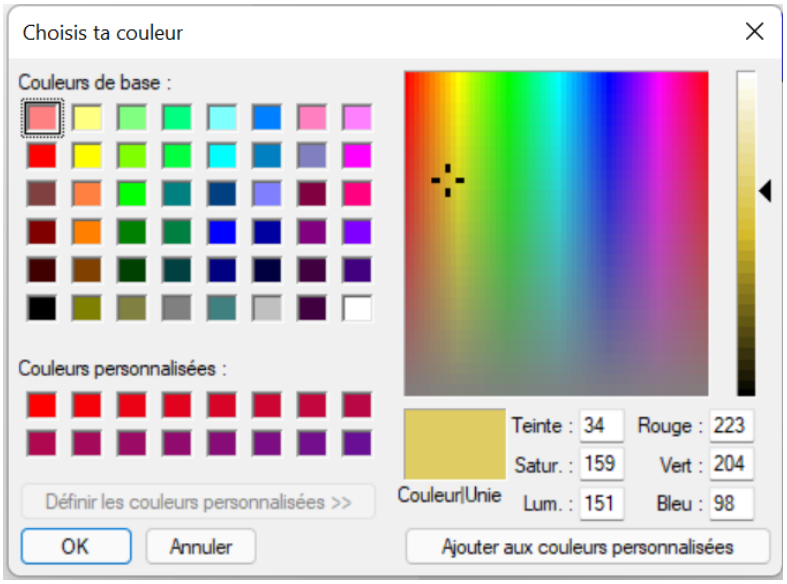

FIGURE 2 – fenêtre personnalisation couleur

Pour cela, nous penserons à importer le module suivant en haut du programme : *from tkinter.colorchooser import askcolor*

Puis nous créons dans la fonction *init* de la **FenPrincipale** pour chacune des zones concernées un paramètre contenant la couleur :

```
self.couleur_ZoneAffichage = "green"
self.couleur_clavier='white'
self.couleur_background='white'
```

FIGURE 3 – création des paramètres

On crée les fonctions associées aux boutons ainsi que d'autre fonctions pour récupérer les paramètres et changer la couleur de la zone demandée :

```python
#Menu pour changer la couleur des formes du pendu
def changer_couleur_ZoneAffichage(self): #change la couleur des formes qui vont être
    C = askcolor(title="Choisis ta couleur")
    self.set_couleur_ZoneAffichage(C[1])

def changer_couleur_clavier(self): #change la couleur d'arrière plan du clavier
    C = askcolor(title="Choisis ta couleur")
    self.set_couleur_clavier(C[1])

def changer_couleur_background(self): #change la couleur d'arrière plan du clavier
    C = askcolor(title="Choisis ta couleur")
    self.set_couleur_background(C[1])
```

(a) fonction associée aux boutons

```python
#fonction pour gérer la couleur des éléments
def set_couleur_ZoneAffichage(self,couleur):
    self.couleur_ZoneAffichage  = couleur
    self.__canevas.config(bg=self.couleur_ZoneAffichage)

def set_couleur_clavier(self,couleur):
    self.couleur_clavier =couleur
    for i in self.__L:
        i.config(bg=self.couleur_clavier)

def set_couleur_background(self,couleur):
    self.couleur_background =couleur
    self.configure(bg=self.couleur_background)
```

(b) fonctions récupératrice des paramètres

FIGURE 4

Finalement, on crée le bouton avec le menu associé et on associé les fonctions créées précédement :

```python
#création de l'interface pour changer les couleurs de l'interface
boutonCouleur=Menubutton(barreOutils,text='Choisir une couleur | ➪ ',bg='white')
boutonCouleur.pack(side=RIGHT, padx=5, pady=5)
menuDeroulantCouleur=Menu(boutonCouleur, tearoff = 0)
menuDeroulantCouleur.add_command(label="Couleur de la zone d'affichage", command=self.changer_couleur_ZoneAffichage)
menuDeroulantCouleur.add_command(label='Couleur du clavier', command=self.changer_couleur_clavier)
menuDeroulantCouleur.add_command(label='Couleur de l'arrière plan', command=self.changer_couleur_background)
```

FIGURE 5 – création des paramètres

Nous obtenons ceci :

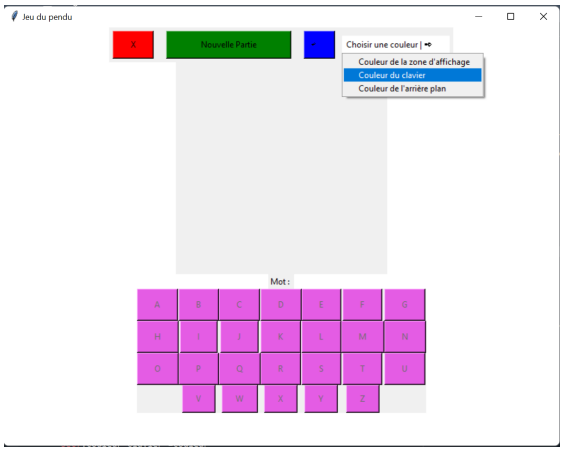

FIGURE 6 – bouton retour

3.2 Seconde amélioration : création de l'action retour

Cette seconde amélioration a pour objectif de créer un bouton qui permet de retourner au coup d'avant. Il faut donc dégriser la lettre qui avait été jouée, modifier le mot à découvrir si une lettre avait été découverte ainsi que cacher le dernier membre du pendu si on n'avait pas trouver de lettre.

Pour cela, nous commencons par créer le bouton :

```
#bouton retour en arrière
boutonUndo=Button(barreOutils,text=' ↩ ',bg='blue',width=3)
boutonUndo.pack(side=RIGHT, padx=5, pady=5,ipady=7,ipadx=7)
```

FIGURE 7 – bouton retour

Ensuite, nous créons une liste qui contiendra l'historique des coups joués avec le résultat du coup (vrai si la lettre fait partie du mot chercher et faux dans le cas contraire).

Cela nous donne ceci :

```
if cpt==0:
    self.__nbManques += 1
    self.__liste_event.append([lettre,False])
    if self.__nbManques >=10:
        self.finPartie(False)

    self.__canevas.dessinePiecePendu(self.__nbManques)

else:
    self.__texte_mot.set("Mot : " + self.__motAffiche)
    self.__liste_event.append([lettre,True])
    if self.__mot == self.__motAffiche:
        self.finPartie(True)
```

FIGURE 8 – incrémentation de la liste

Ensuite, nous implémentons la fonction associée au bouton de triche. Elle va permettre de dégriser la case qui avait été jouée et en fonction du résultat de celle-ci rétablir la situation précédente (cela fait appel à la fonction cacher de la classe **Zone Affichage** pour cacher les membres qui avaient été affichés ou bien cela cache les lettres qui avaient été trouvées ...) :

```python
#fonction pour avoir l'historique des coups
def undo(self):
    try :
        num_bouton=ord(self.__liste_event[-1][0])-ord('A')
        self.__L[num_bouton].config(state=NORMAL) #changer l'état de la lettre
        if self.__liste_event[-1][1]==False: #on a raté le dernier coup
            self.__canevas.cacherpiece(self.__nbManques) #retirer la forme du coup manque
            self.__nbManques-=1 #reset des coups manqués

        else: #on a trouver une lettre au dernier coup
            newmotcache=''
            for i in range(len(self.__mot)):
                T=np.array(self.__liste_event) #convertion
                if self.__mot[i] in list(T[:-1,0]):#si c'est la bonne lettre (on vérifie dans les anciens coups)
                    newmotcache+=self.__mot[i]
                else:
                    newmotcache+='*'
            self.__motAffiche = newmotcache
        self.__texte_mot.set('Mot : '+self.__motAffiche+'\n'+"Vous avez annulé le coup précédent")
        self.__liste_event.pop() #on retire le dernier coup de la liste

    #cas où le premier coup n'a pas encore été joué
    except IndexError:
        tkinter.messagebox.showwarning ( title = "Erreur" , message = "Attention, tu dois jouer ton premier coup!" )
```

FIGURE 9 – fonction undo

On pourra remarquer la gestion d'exeption opérée par les fonction *try* et *except* qui ici, prennent en compte le cas où le joueur voudrait revenir en arrière alors qu'aucun coup avait été joué (on aurait pu tout simplement griser la case).

Finalement, on associe la fonction au bouton en écrivant ceci :

$$boutonUndo.config(command = self.undo)$$

On obtient :

FIGURE 10 – Exemple d'erreur

4 Conclusion

Finalement, nous avons pu voir comment était structuré notre jeu du Pendu à l'aide de son diagramme UML puis nous avons vu en détail l'implémentation de deux améliorations du jeu : la personnalisation des couleurs de certaines parties ainsi que la création d'une fonction triche.