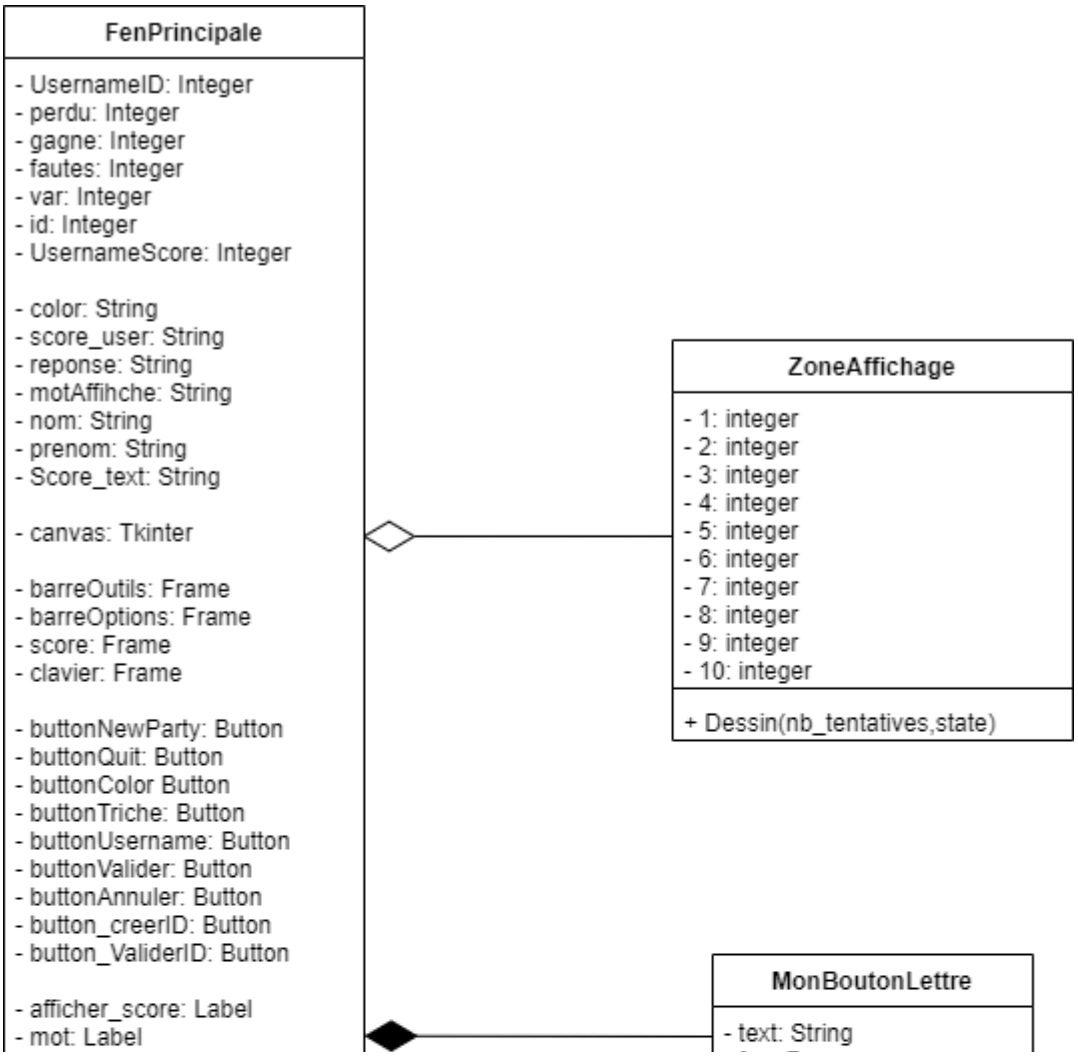
Gonçalves
Taïga
Groupe D1b
Encadrant: Vuillemot Romain

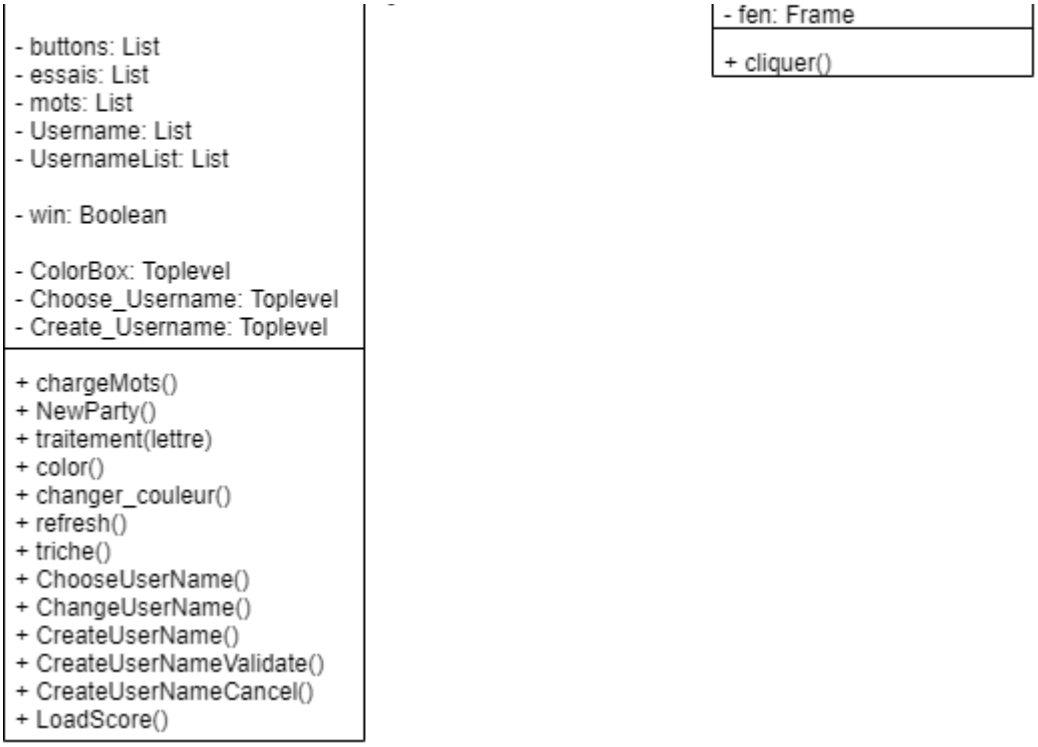
BE #5 : Le jeu du Pendu

Sommaire:

1. Diagramme de Classe UML

L'objectif de ce BE est de réaliser le `jeu du Pendu`. Voici donc une représentation de la structure du programme sous forme d'un diagramme de Classe UML:





Nous ne détaillerons pas le codage de la partie fonctionnelle du jeu, celle-ci étant déjà détaillée dans l'énoncé. Nous allons donc nous intéresser sur les trois fonctionnalités optionnelles qui ont été ajoutées dans le programme.

2. Apparence

Dans cette partie, nous allons améliorer la classe FenPrincipale afin de pouvoir modifier les couleurs de tous les objets présents dans l'interface du jeu. Pour cela, nous allons d'abord créer des attributs dans lesquels nous stockerons la couleur actuelle de chaque objet:

```
class FenPrincipale(Tk):
    def __init__(self):
        Tk.__init__(self)

        self.__color_bg='dodgerblue'
        self.__color_canvas='darkorange'
        self.__end_text='red'
        self.__color_contour_bouttons='white'
        self.__color_buttonNewParty_bg='SystemButtonFace'
        self.__color_buttonNewParty_fg='black'
        self.__color_buttonNewParty_active='SystemButtonFace'
        self.__color_buttonQuit_bg='SystemButtonFace'
        self.__color_buttonQuit_fg='black'
        self.__color_buttonQuit_active='SystemButtonFace'
        self.__color_buttonColor_bg='SystemButtonFace'
        self.__color_buttonColor_fg='black'
        self.__color_buttonColor_active='SystemButtonFace'
```

```
        self.__color_buttonTriche_bg='SystemButtonFace'
        self.__color_buttonTriche_fg='black'
        self.__color_buttonTriche_active='SystemButtonFace'
```

Créons ensuite les attributs `self.__barreOptions` et `self.__buttonColor` qui représentent respectivement un objet de type `tk.Frame` et `tk.Button`.

Le bouton `self.__buttonColor` est relié à la méthode `self.color` :

```
    def color(self):
        self.__buttonColor.config(state="disabled")
        self.__ColorBox=Toplevel(self)
        self.__ColorBox.title("Choix des couleurs")
        Choix=[('Arrière plan',1),('Fenêtre principale',2),('texte de fin',3),
('contour des boutons',4),('Bouton "Nouvelle partie"',5),
               ('Texte "Nouvelle Partie"',6),('Bouton "Quitter"',7),('Texte
"Quitter"',8),('Bouton "Couleur"',9),
               ('Texte "Couleur"',10),('Bouton "Triche"',11),('Texte "Triche"',12)]
        self.__var=IntVar()
        for text,choix in Choix:
            b =
Radiobutton(self.__ColorBox,text=text,value=choix,variable=self.__var,command=self.chang

            b.deselect()
            b.pack(anchor=W)
        self.__Button_Valider=Button(self.__ColorBox,text="Ok",width=10)
        self.__Button_Valider.pack(anchor=S)
        self.__Button_Valider.config(command=self.__ColorBox.destroy)
        self.__ColorBox.bind('<Destroy>', self.refresh)
```

Cette méthode ouvre une nouvelle fenêtre avec plusieurs boutons. L'attribut `self.__var` mémorise le numéro du dernier bouton appuyé, et le clic de n'importe quel bouton appelle la méthode `self.changer_couleur` :

```
    def changer_couleur(self):
        choix=self.__var.get()
        color=colorchooser.askcolor()[1]
        if choix==1:
            self.__color_bg=color
        elif choix==2:
            self.__color_canvas=color
        elif choix==3:
            self.__end_text=color
        elif choix==4:
            self.__color_contour_bouttons=color
        elif choix==5:
            self.__color_buttonNewParty_bg=color
            self.__color_buttonNewParty_active=color
        elif choix==6:
            self.__color_buttonNewParty_fg=color
        elif choix==7:
            self.__color_buttonQuit_bg=color
            self.__color_buttonQuit_active=color
```

```
        elif choix==8:
            self.__color_buttonQuit_fg=color
        elif choix==9:
            self.__color_buttonColor_bg=color
            self.__color_buttonColor_active=color
        elif choix==10:
            self.__color_buttonColor_fg=color
        elif choix==11:
            self.__color_buttonTriche_bg=color
            self.__color_buttonTriche_active=color
        elif choix==12:
            self.__color_buttonTriche_fg=color
```

Cette méthode permet de modifier les attributs mémorisant la couleur de chaque objet de l'interface. Enfin, en appuyant sur le bouton `self.__Button_Valider`, la méthode `self.refresh` est appelée:

```
    def refresh(self,event):
        self.__buttonColor.config(state="normal")
        self.__ColorBox.destroy()
        self.configure(bg=self.__color_bg)
        self.__canvas.configure(bg=self.__color_canvas)
        self.__canvas.itemconfig(self.__perdu,fill=self.__end_text)
        self.__canvas.itemconfig(self.__gagne,fill=self.__end_text)
        self.__barreOutils.configure(bg=self.__color_contour_bouttons)
        self.__clavier.configure(bg=self.__color_contour_bouttons)

self.__buttonNewParty.config(bg=self.__color_buttonNewParty_bg,fg=self.__color_buttonNew


self.__buttonQuit.config(bg=self.__color_buttonQuit_bg,fg=self.__color_buttonQuit_fg,act


self.__buttonColor.config(bg=self.__color_buttonColor_bg,fg=self.__color_buttonColor_fg,


self.__buttonTriche.config(bg=self.__color_buttonTriche_bg,fg=self.__color_buttonTriche_
```

Cette méthode permet d'actualiser les couleurs de tous les objets de l'interface.

2. Bouton triche

Nous allons maintenant implémenter une nouvelle fonctionnalité, permettant de réaliser un retour en arrière durant une partie. Pour cela nous allons créer le bouton `self.__buttonTriche` placé dans la fenêtre `self.__barreOptions`. Ce bouton est relié à la méthode `self.triche`:

```
    def triche(self):
        if len(self.__essais)>0:

            if self.__win==True or self.__fautes==10:
                for i in range(26):
```

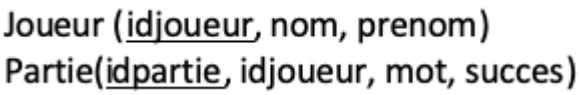
```
                    self.__buttons[i].config(state=NORMAL)
                    if self.__win==True:
                        self.__canvas.itemconfig(self.__gagne,state='hidden')
                    else:
                        self.__canvas.itemconfig(self.__perdu,state='hidden')
                conn = sqlite3.connect('pendu.db')
                curseur = conn.cursor()
                idpartie = curseur.execute("SELECT COUNT(idpartie) FROM
Partie").fetchall()[0][0]-1
                curseur.execute("DELETE FROM Partie WHERE idpartie=
{}".format(idpartie))
                conn.commit()
                conn.close()
                self.LoadScore()

            if self.__motAffiche==self.__reponse:
                self.__win=FALSE

            lettre=self.__essais.pop()
            trouve=False
            for i in range(len(self.__reponse)):
                if lettre==self.__reponse[i]:
                    motAffiche=self.__motAffiche[:i]+"*"+self.__motAffiche[i+1:]
                    self.__motAffiche=motAffiche
                    self.__mot.config(text='Mot: '+self.__motAffiche)
                    trouve=True

            if not trouve:
                self.__canvas.Dessin(self.__fautes,"hidden")
                self.__fautes+=-1

            ###Réactiver la dernière lettre (attention: toutes les lettres se
retrouvent désactivé si la partie était terminée)
            for i in range(26):
                lettre=chr(ord('A')+i)
                if lettre in self.__essais:
                    self.__buttons[i].config(state="disabled")
                else:
                    self.__buttons[i].config(state="normal")
```

Le codage de cette méthode est simple: il suffit d'écrire le contraire de la méthode `self.traitement`. La seule précaution à prendre est la réactivation du clavier. Pour ce faire, nous créons l'attribut `self.__essais` qui stocke toutes les lettres qui ont été cliquées dans l'ordre sous forme de liste. L'appel de la méthode `self.triche` supprime le dernier élément de cette liste.

3. Score joueur

Pour finir, nous allons ajouter la possibilité de se créer un identifiant et d'afficher son score. Ces données seront stockées dans la base de données SQL appelée

`pendu.db` dont sa structure est la suivante:



Créons d'abord les objets suivants:

```
        self.__score = Frame(self)
        self.__score.pack(side=TOP,pady=4,padx=10)
        self.__afficher_score = Label(self.__score)
        self.__afficher_score.grid(row=0,column=0,padx=5)
        self.__buttonUsername = Button(self.__score, text="Changer
d'utilisateur",borderwidth=2,width=18)
        self.__buttonUsername.grid(row=0,column=1,pady=1)

        self.__UsernameID=0 #Utilisateur par défaut
        self.__score_user=self.LoadScore()

        self.__buttonUsername.config(command=self.ChooseUsername)
```

Le choix d'un identifiant étant facultatif, nous créons l'identifiant `0` qui est l'identifiant par défaut. Le score affiché au lancement du programme correspond donc à celui de l'usilisateur par défaut. Ce score s'affiche grâce à la méthode `self.__LoadScore` :

```
    def LoadScore(self):
        conn = sqlite3.connect('pendu.db')
        curseur = conn.cursor()
        self.__Username= curseur.execute("SELECT nom,prenom FROM Joueur WHERE
idjoueur={}".format(self.__UsernameID)).fetchall()[0]
        self.__UsernameScore = curseur.execute("SELECT COUNT(succes) FROM Partie WHERE
idjoueur={} AND succes=1".format(self.__UsernameID)).fetchall()[0]
        self.__score_text = "Score de "+self.__Username[0]+" "+self.__Username[1]+":
"+str(self.__UsernameScore[0])
        self.__afficher_score.config(text=self.__score_text)
        conn.close()
```

Le bouton `self.__buttonUsername` appelle la méthode `self.ChooseUsername` qui permet de choisir son identifiant, ou d'en créer un nouveau:

```
    def ChooseUsername(self):
        self.__Choose_Username = Toplevel(self)
        self.__Choose_Username.title("Choix de l'utilisateur")
        conn = sqlite3.connect('pendu.db')
        curseur = conn.cursor()
        self.__UsernameList = curseur.execute("SELECT * FROM Joueur").fetchall()
        conn.close()
        self.__id=IntVar()
        for idjoueur,nom,prenom in self.__UsernameList:
            u = Radiobutton(self.__Choose_Username,text=nom+"
"+prenom,value=idjoueur,variable=self.__id,command=self.ChangeUsername,indicatoron=0)
            u.deselect()
            u.grid(sticky=W,padx=2)
```

```
        row=len(self.__UsernameList)+2
        Label(self.__Choose_Username).grid()
        self.__Button_annuler = Button(self.__Choose_Username,
text="Annuler",command=self.__Choose_Username.destroy)
        self.__Button_annuler.grid(row=row,column=1,padx=2)
        self.__Button_creerID = Button(self.__Choose_Username, text="Créer un nouvel
identifiant",command=self.CreateUsername)
        self.__Button_creerID.grid(row=row,column=0,padx=2)
```

Dans cette méthode, si on clique sur un identifiant, la méthode `self.ChangeUsername` est appelée:

```
    def ChangeUsername(self):
        self.__UsernameID = self.__id.get()
        self.LoadScore()
        self.__Choose_Username.destroy()
```

Cette méthode permet de changer d'identifiant en modifiant l'attribut `self.__UsernameID` (qui contenait 0 par défaut) et en appelant la méthode `self.LoadScore`. Maintenant, si on veut créer un nouvel identifiant, il suffit de cliquer sur le bouton `self.__Button_creerID` qui appelle la méthode suivante:

```
    def CreateUsername(self):
        self.__Choose_Username.destroy()
        self.__Create_Username = Toplevel(self)
        self.__Create_Username.title("Création d'un nouvel identifiant")
        Label(self.__Create_Username,text="Nom: ").grid(row=0)
        Label(self.__Create_Username,text="Prenom: ").grid(row=1)
        self.__nom = StringVar()

Entry(self.__Create_Username,textvariable=self.__nom).grid(row=0,column=1,pady=2)
        self.__prenom = StringVar()

Entry(self.__Create_Username,textvariable=self.__prenom).grid(row=1,column=1,pady=2)
        self.__Button_ValiderID = Button(self.__Create_Username,
text="Valider",command=self.CreateUsernameValidate)
        self.__Button_ValiderID.grid(row=2)
        self.__Button_AnnulerID = Button(self.__Create_Username,
text="Annuler",command=self.CreateUsernameCancel)
        self.__Button_AnnulerID.grid(row=2,column=1,sticky=W,pady=4)
```

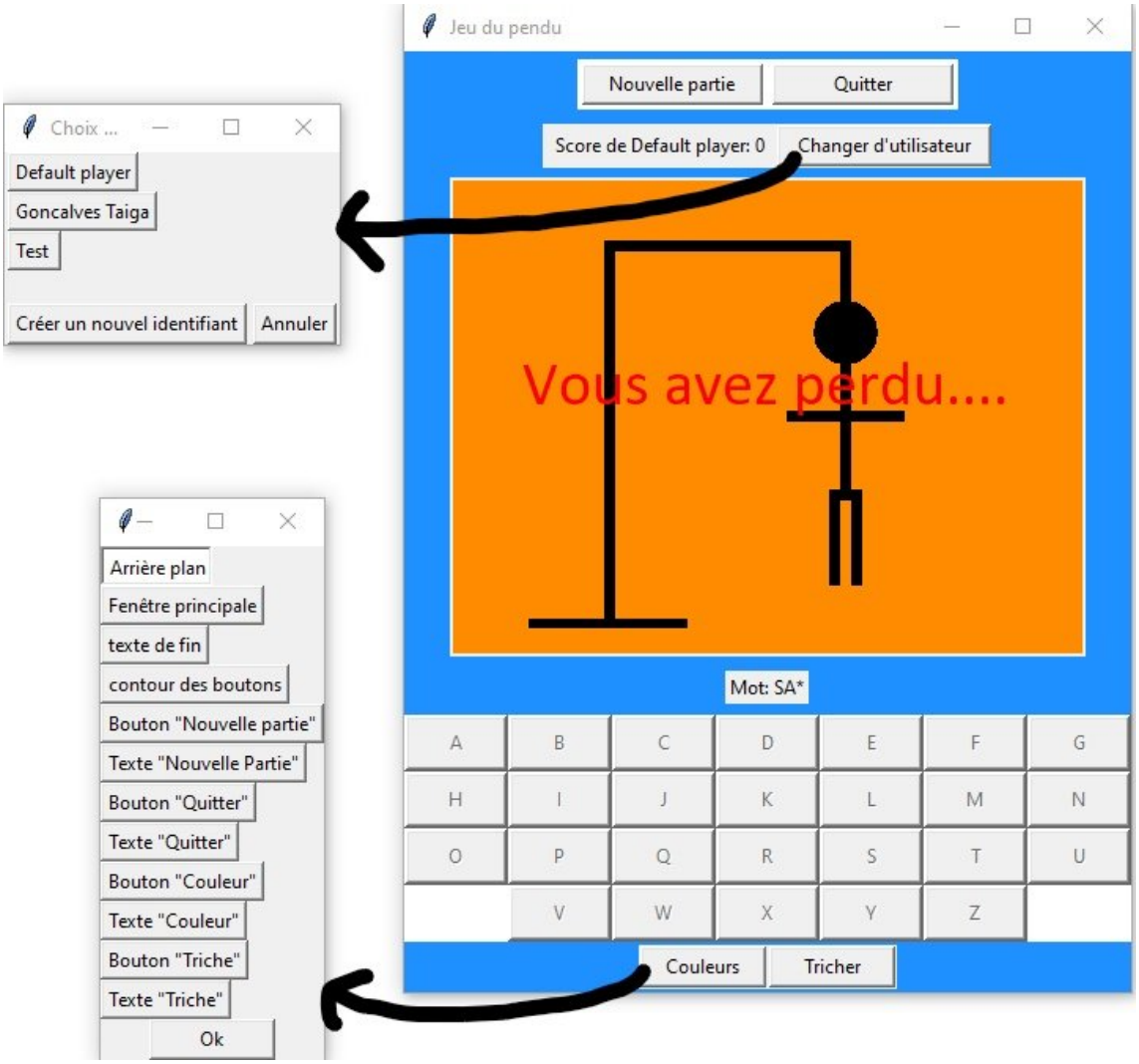
Cette méthode permet d'entrer un nom et un prenom pour son identifiant. Le numéro de l'identifiant (clé privé) est créee en comptant le nombre d'identifiants stockés dans la base de données. Les boutons `self.__Button_ValiderID` et `self.__Button_AnnulerID` permettent de valider ou d'annuler la création de l'identifiant, en faisant respectivement appel aux méthodes suivantes:

```
    def CreateUsernameValidate(self):
        self.__Create_Username.destroy()
        conn = sqlite3.connect('pendu.db')
        curseur = conn.cursor()
        idjoueur = curseur.execute("SELECT COUNT(idjoueur) FROM Joueur").fetchall()[0]
[0]
```

```
        nom = self.__nom.get()
        prenom = self.__prenom.get()
        curseur.execute("INSERT INTO Joueur
VALUES({},'{}','{}')".format(idjoueur,nom,prenom))
        conn.commit()
        conn.close()
        self.ChangeUsername()
        self.ChooseUsername()

    def CreateUsernameCancel(self):
        self.__Create_Username.destroy()
        self.ChooseUsername()
```

5. Résumé

Nous avons donc réalisé un jeu du pendu avec trois fonctionnalités facultatives: la première permet de changer les couleurs de l'interface. La deuxième permet de tricher lors d'une partie, en effectuant un retour en arrière. La dernière permet d'afficher le score d'un identifiant au choix.


Jeu du pendu
Nouvelle partie
Quitter
Score de Default player: 0
Changer d'utilisateur
Vous avez perdu....
Mot: SA*
A B C D E F G
H I J K L M N
O P Q R S T U
V W X Y Z
Couleurs
Tricher
Choix ...
Default player
Goncalves Taiga
Test
Créer un nouvel identifiant
Annuler
Arrière plan
Fenêtre principale
texte de fin
contour des boutons
Bouton "Nouvelle partie"
Texte "Nouvelle Partie"
Bouton "Quitter"
Texte "Quitter"
Bouton "Couleur"
Texte "Couleur"
Bouton "Triche"
Texte "Triche"
Ok