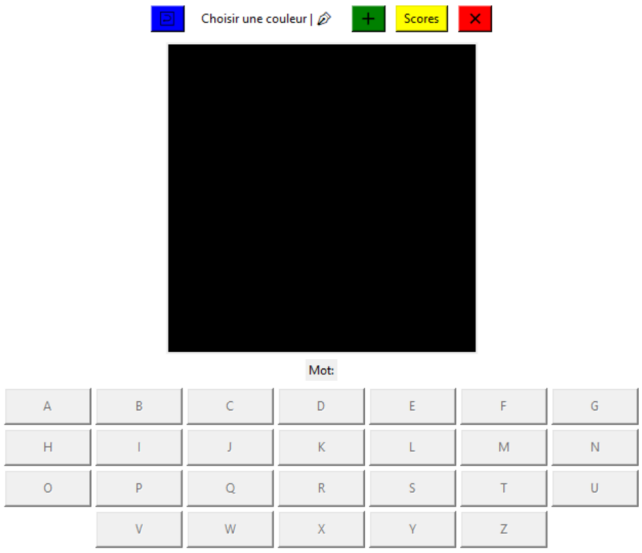
Le jeu du Pendu

> TRIBOULET Corentin – D1b – DELLANDREA

Partie: Amélioration du jeu



Fenêtre principale

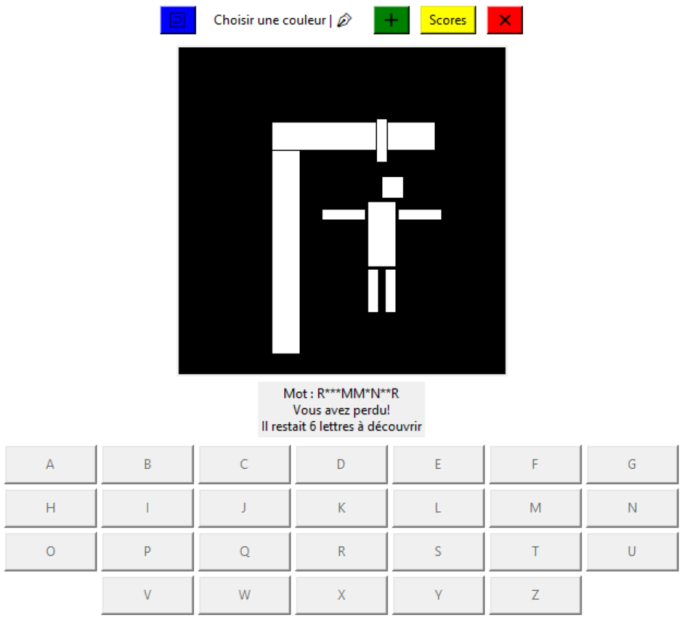
Avant propos

Indications générales:

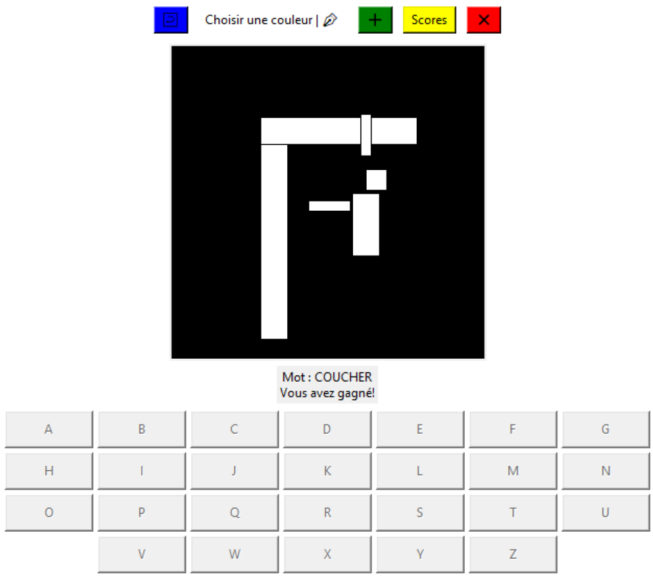
- le symbole " flèche " en bleu signifie Undo (ou retour en arrière)
- le symbole " + " en vert, débute une nouvelle partie
- le symbole " X " en rouge, quitte la page

Cas de fin de parties:

- Lorsque l'on perd, on reçoit en précision combien de lettres il nous restait à découvrir.

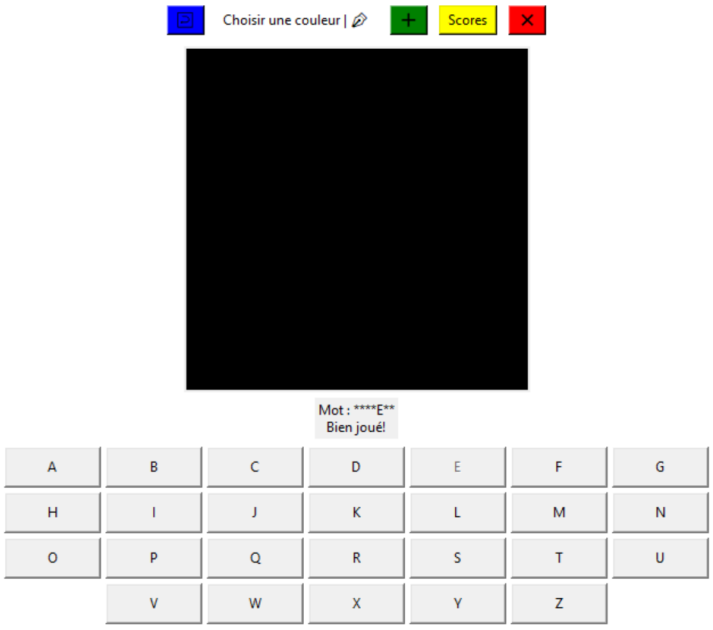


- Lorsque l'on gagne, le message de victoire est affiché



Cas d'évènements :

- On trouve une bonne lettre, un message de récompense est affiché



- On ne trouve pas de bonne lettre, une partie du pendu s'affiche ainsi qu'un message

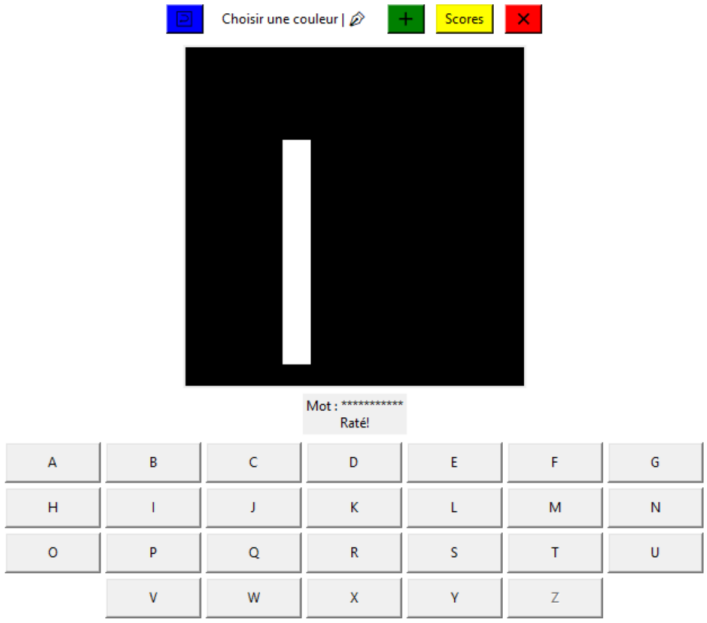
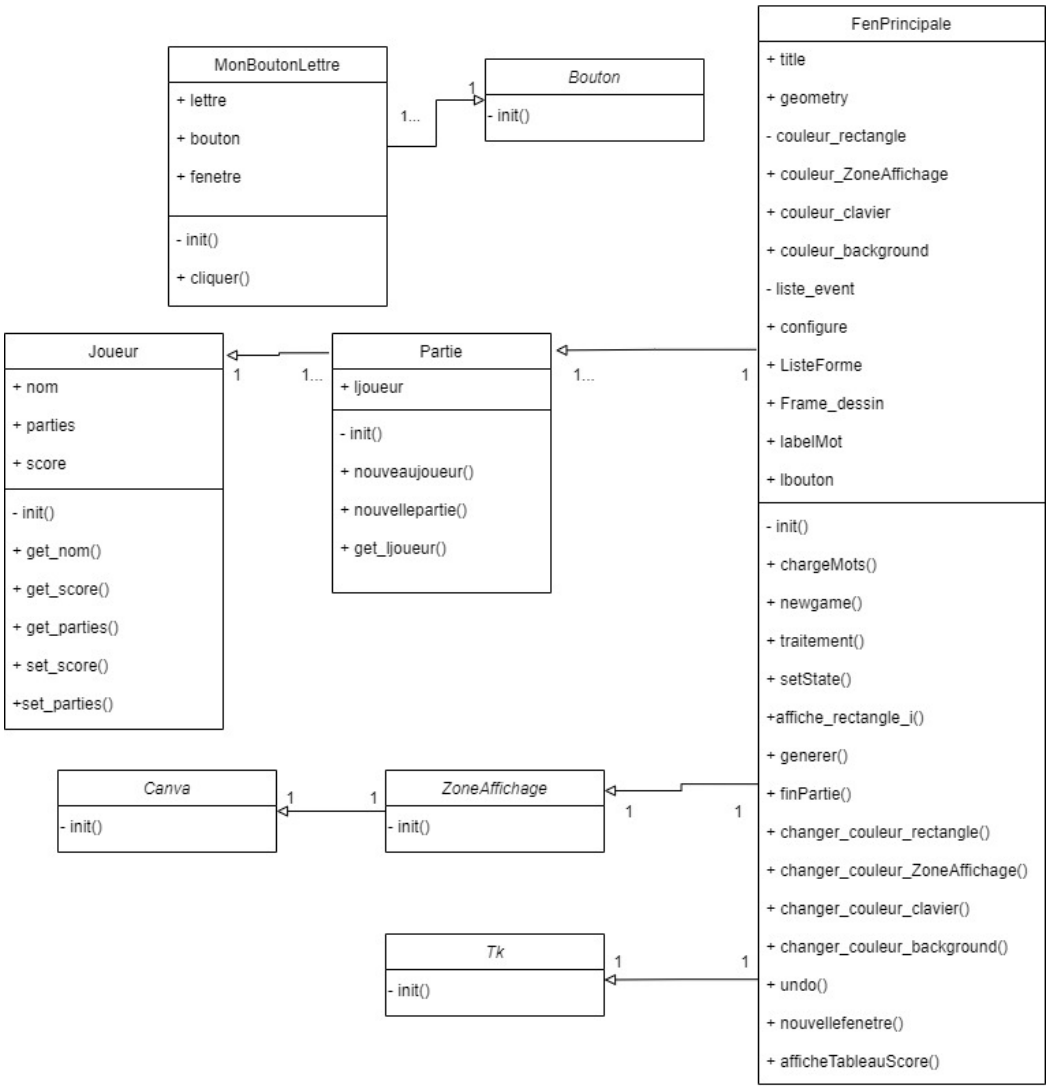


Diagramme UML:

Diagramme UML: Pendu



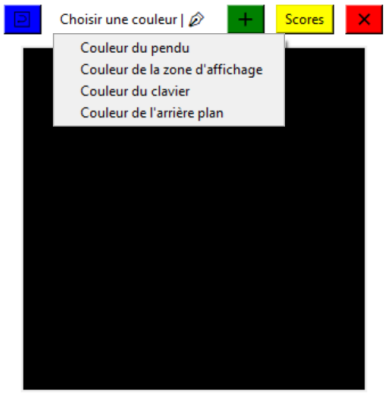
1. Apparence

Choix des couleurs

L'utilisateur aura le choix des couleurs pour 4 différentes zones de la fenêtre:

- Le pendu
- La zone d'affichage
- Le clavier
- L'arrière plan

Ce choix est possible grâce au **menu** déroulant suivant:



Code à rajouter pour obtenir le menu

```
#Menu pour changer la couleur des formes du pendu
boutonCouleur=Menubutton(Barre_doutils,text='Choisir une couleur | ➸ ',bg='white')
boutonCouleur.pack(side=RIGHT, padx=5, pady=5)
menuDeroulantCouleur=Menu(boutonCouleur, tearoff = 0)
menuDeroulantCouleur.add_command(label='Couleur du pendu', command=self.changer_couleur_rectangle)
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)

#configuration des boutons
boutonCouleur.configure(menu=menuDeroulantCouleur)

#setters
def set_couleur_rectangle(self,couleur):
        self.__couleur_rectancle=couleur

def set_couleur_ZoneAffichage(self,couleur):
        self.couleur_ZoneAffichage  = couleur

def set_couleur_clavier(self,couleur):
        self.couleur_clavier =couleur

def set_couleur_background(self,couleur):
        self.couleur_background =couleur

#fonctions
def changer_couleur_rectangle(self): #change la couleur des formes qui vont être affichées à l'avenir
        C = askcolor(title="Choisis ta couleur")
        self.set_couleur_rectangle(C[1])

def changer_couleur_ZoneAffichage(self): #change la couleur des formes qui vont être affichées à l'avenir
        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])
```

2. Triche!

La fonction triche est représenté par la fonction Undo (ou retour en arrière). La clef pour pouvoir effectuer le retour en arrière c'est de savoir ce qu'on a fait juste avant. C'est pour cela que l'on stocke chaque évènement dans la liste *self.levent*.

Chaque évènement y est stocké sous forme d'un doublet dont la première composante est:

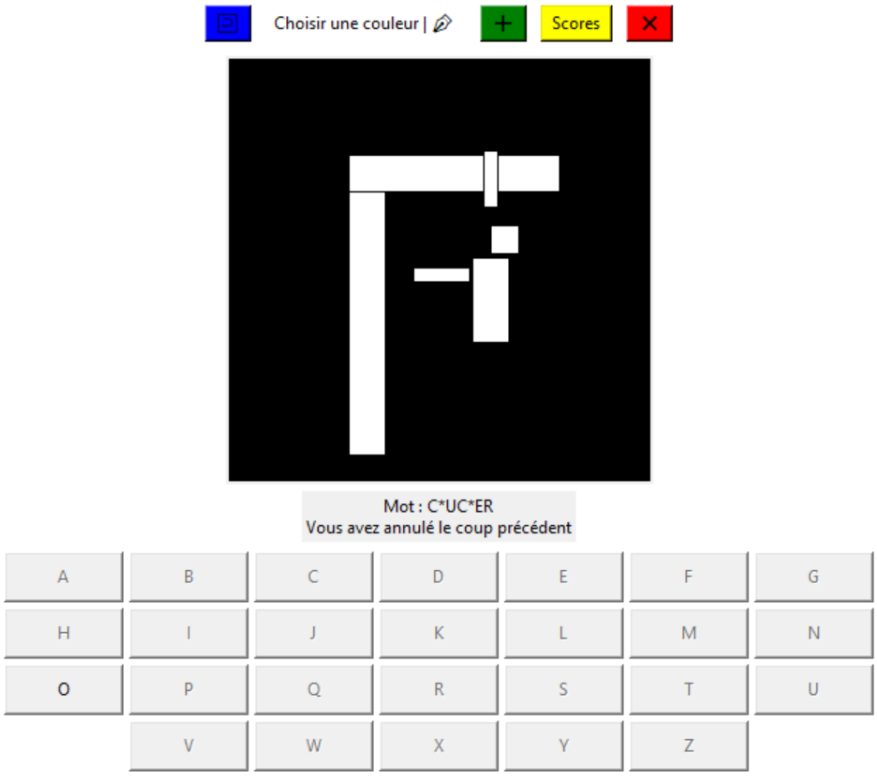
- la lettre voulue

et la deuxième composante est un booléen :

- True si la lettre est dans le mot
- False si la lettre n'est pas dans le mot

Exemple : [['E', True], ['D', False], ['Q', False], ['A', True], ['L', False], ['B', False], ['M', False]]

```
def undo(self):  #qu'a ton fait au tour precedent ?
        num_bouton=ord(self.__liste_event[-1][0])-ord('A')
        self.lbouton[num_bouton].bouton.config(state=NORMAL) #changer l'état de la lettre
        if self.__liste_event[-1][1]==False: #on a raté le dernier coup
            self.Frame_dessin.delete(self.nbcoupsmanques) #retirer la forme du coup manque
            self.nbcoupsmanques-=1 #reset des coups manqués

        else: #on a trouver une lettre au dernier coup
            newmotcache=''
            for i in range(len(self.__mot)):
                #print(newmotcache)
                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.mot_cache = newmotcache
        self.labelMot.set('Mot : '+self.mot_cache+'\n'+"Vous avez annulé le coup précédent")
        self.__liste_event.pop() #on retire le dernier coup de la liste
```



Les potentiels problèmes

- le bouton Undo peut être cliquer avant que l'on décide de jouer (j'ai essayé en vain de supprimer ce problème)

3. Score joueur

On se propose de calculer le score d'un joueur par la manière suivante:

- s'il a perdu, on mentionne "perdu"
- s'il a gagné on mentionne le nombre de lettres sélectionnées qui n'était pas dans le mot à découvrir

Pour afficher les scores, le bouton "Scores" en jaune ouvrira une nouvelle fenêtre où sera affiché le tableau récapitulatif des parties précédentes de tous les joueurs.

Pour identifier chaque joueur, au début de chaque partie, une fenêtre s'ouvre de demande le nom du joueur. Après avoir saisie le nom, le joueur ferme la page et peut commencer à jouer.

L'attribut score est définie par un doublet de la manière suivante:

- si on a gagné la partie, la première composante correspond au nombre de lettres sélectionnées qui n'était pas dans le mot à découvrir et ma seconde au booléen True
- si on perd, la première composante est le texte 'perdu' et la seconde le booléen False

Exemple : Joueur(nom='Joueur 1', score=[['perdu',False] , [4,True] , [8,True]])

Définition des nouvelles classes

```
class Partie(Joueur):
    def __init__(self):
        self.ljoueur=[] #liste des joueurs

    def nouveaujoueur(self,nom,score):
        j=Joueur(nom,[score])
        self.ljoueur.append(j)
        j.set_parties(1) #nouvelle partie

    def nouvellepartie(self,nom,score):#on sait que le joueur a deja joué
        for j in self.ljoueur:
            if j.get_nom()==nom:
                j.score.append(score) #ajout du score
                nb_ancienne_parties=j.get_parties()
                j.set_parties(1+nb_ancienne_parties) #ajout de 1 du nombre de partie

    def get_ljoueur(self):
        return(self.ljoueur)

class Joueur: #creation de la classe joueur
    def __init__(self,nom,score):
        self.nom=nom
        self.score=score
        self.parties=0

    #DESCRIPTION Joueur(Nom du joueur,[(1),(2)])
    #(1)= 'perdu' si la partie est perdue
    #   = nombre d'essais pour réussir
    #(2)= True si la partie est gagnée
    #   = False si la partie est perdue

    def get_nom(self):
        return(self.nom)

    def get_score(self):
        return(self.score)

    def get_parties(self):
        return(self.parties)

    def set_score(self,score):
        self.score = score

    def set_parties(self, parties):
        self.parties = parties
```

Définition des fonctions

```
def nouvellefenetre(self): #créé la nouvelle fenetre
        self.__nom = StringVar()
        nouvellefen = Toplevel(fen)
        EmployeID=Label(nouvellefen, text="Nom :").pack(side=LEFT,ipadx=20,ipady=20,padx=2, pady=2)
        nomparticipant = Entry(nouvellefen,textvariable=self.__nom)
        nomparticipant.pack(side=LEFT,ipady=5,padx=20, pady=2)
        consigne=Label(nouvellefen,text="(Fermer la fenêtre après avoir renseigné votre nom)")
        consigne.pack(side=LEFT,ipady=5,padx=20, pady=2)


    def afficheTableauScore(self):
        fenScores=Toplevel(fen) #ouverture d'une fenêtre où l'on affiche les scores
        self.Tableau=[]
        nombre_parties_max=0
        nombre_joueur=len(self.ljoueur)

        for j in self.ljoueur:#determination du nombre maximal de partie de tous les joueurs
            if nombre_parties_max<len(j.get_score()):
```

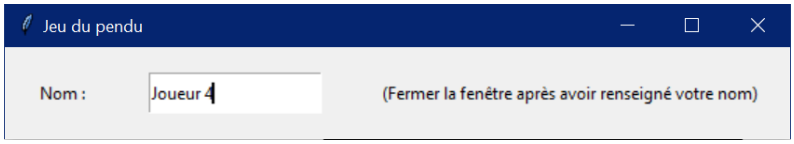
```
                nombre_parties_max=len(j.get_score())

        #creation du Tableau en type tableau-liste
        for j in self.ljoueur:
            self.Tableau.append([j.get_nom()]+j.get_score()+[None]*(nombre_parties_max-len(j.get_score())))
                        #les cases None correspondent au fait que le joueur n'a pas encore joué cette partie

        #creation de chaque case du tableau
        Tab=np.array(self.Tableau)
        for i in range(nombre_joueur):
            for j in range(nombre_parties_max + 1):#on affiche le nom + les parties

                self.e = Label(fenScores, width=20, fg='grey',justify=CENTER,text=str(Tab[i,j]))#insertion du score dans le tableau
                self.e.grid(row=i+1, column=j+1,ipadx=5,ipady=5,padx=5,pady=5) #decalage à cause de l'indication


        #commentaires à ajouter en haut du tableau
        indication=Label(fenScores,text="Scores: nombres d'essais pour gagner OU mention perdu")
        indication.grid(row=0, column=0,ipady=5,padx=5, pady=2)
        nom_joueur=Label(fenScores,text="Nom des joueurs")
        nom_joueur.grid(row=0, column=1,ipady=5,padx=5, pady=2)

        for i in range(nombre_parties_max):#affichage des parties
                self.e = Label(fenScores, text='Partie ' + str(i+1))
                self.e.grid(row=0, column=i+2,ipadx=5,ipady=5,padx=5,pady=5) #decalage à cause de l'indication
```

Le bouton Scores

```
#bouton scores
        boutonScores=Button(Barre_doutils,text=" Scores ",bg='yellow' )
        boutonScores.pack(side=RIGHT, padx=5, pady=5)
#configuration
        boutonScores.config(command=self.afficheTableauScore)
```

Rendu de la console

- Lorsque que l'on commence la partie:



- Si on demande les scores:

Entre temps, le joueur 4 perd sa 2ème partie, la case None est remplacée par 'perdu'

Les potentiels problèmes

- le joueur commence sans donner de nom à la page popup
- le joueur écrit son nom en minuscule et l'autre fois en majuscule
- comment déterminer le meilleur joueur