ECOLE CENTRALE LYON : 2015-2016

INFO TC2 : CONCEPTION ET PROGRAMMATION OBJET

GROUPE TD : A2a

CHARGE DE TD : DELLANDREA EMMANUEL

JEU DU PENDU

RAPPORT DU TD4

KONAN Jordan N'guessan Ziahi

NDIAYE Serigne Fallou

Table des matières

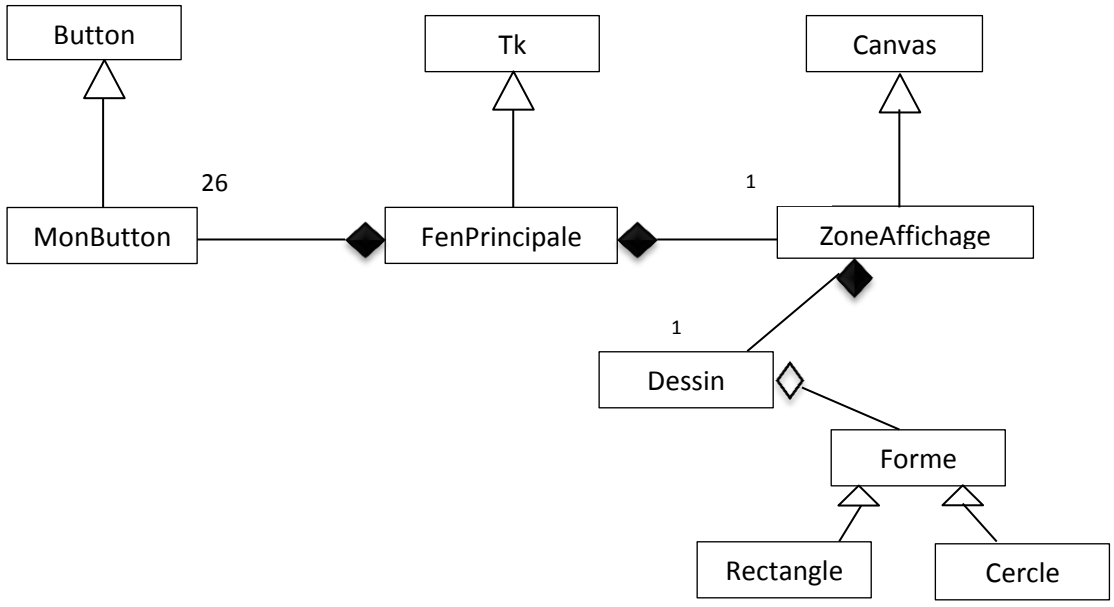
I. PRESENTATION DU PROBLEME

Dans ce TD, l'objectif est de créer un programme qui permet de jouer au ***jeu du pendu***. Il s'agit en effet, de retrouver les lettres contenus dans un mot qui est choisi de façon aléatoire dans une liste de mots. A chaque mauvaise réponse, les différentes parties du pendu prennent forme progressivement. Lorsque le nombre de tentatives autorisées est atteint, le pendu est complètement dessiné et le joueur perd ainsi la partie. Par contre s'il réussit à reconstituer le mot avant l'apparition du pendu, il gagne la partie.

II. PROGRAMME OBLIGATOIRE

1. DIAGRAMME DES CLASSES UML

Le diagramme suivant nous présente les différentes classes ainsi que les relations qui existent entre elles.



Les classes ***MonButton***, ***FenPrincipale*** et ***ZoneAffichage*** héritent respectivement des classes ***Button***, ***Tk*** et ***Canvas*** qui sont contenues dans le module Tkinter .

Les classes MonButton et ***ZoneAffichage*** sont des compositions de la classe FenPrincipale. En effet, on a dans ***FenPrincipale*** 26 intances de la classe ***MonButton*** et une instance de la classe ***ZoneAffichage.***

2. ATTRIBUTS ET METHODES DES CLASSES

a. La classe MonButton

MonButton
- fenPr : string - lettre : string
+ init (parent : string, fenPr: FenPrincipale, l: string, w :int) + cliquer()

- Le code de la classe ***MonButton***

```
-------------------------------------------------------
Classe MonButton pour afficher et gérer le clavier virtuel  |
-------------------------------------------------------
"""
class MonButton(Button):
        def __init__(self,parent,fenPr,l,w):
                self.__fenPr=fenPr
                self.__lettre=l
                Button.__init__(self,master=parent,text=self.__lettre,width=w,bd=2, relief=RAISED, fg='re'\
                        'd', bg ='white', overrelief=RIDGE)

        # Méthode cliquer qui désactive le bouton cliqué et fait appel à la methode traitement de FenPrincipale
        def cliquer(self):
                self.config(state=DISABLED)
                self.__fenPr.traitement(self.__lettre)

########################## Fin de la classe#####################################
```

b. La classe ZoneAffichage

ZoneAffichage
+ init (parent: string, w: int, h: int, couleur :string) + afficherElement(nb :int)

- Le code de la classe ***ZoneAffichage***

```
------------------------------------------------------------
Classe ZoneAffichage pour afficher les formes géométriques  |
------------------------------------------------------------
"""
class ZoneAffichage(Canvas):
        def __init__(self,parent, w, h, c):
                Canvas.__init__(self, width = w, height = h, bg = c) # Héritage de Canvas
                self.__d=Dessin()                                    # objet qui va dessiner les formes
                # L'ensembles des formes nécessaires pour afficher le pendu
                self.__rec1=Rectangle(100,305,"black",50,2)
                self.__rec2=Rectangle(100,180,"black",2,250)
                self.__rec3=Rectangle(155,55,"black",105,2)
                self.__rec4=Rectangle(205,75,"black",2,35)
                self.__cer=Cercle(205,110,"black",40) # Tête
                self.__rec5=Rectangle(205,155,"black",20,55)
                self.__rec6=Rectangle(196,200,"black",2,55)
                self.__rec7=Rectangle(214,200,"black",2,55)
                self.__rec8=Rectangle(176,145,"black",40,2)
                self.__rec9=Rectangle(234,145,"black",40,2)
                # on ajoute les formes dans l'objet Dessin
                self.__d.addForme(self.__rec9)
                self.__d.addForme(self.__rec8)
                self.__d.addForme(self.__rec7)
                self.__d.addForme(self.__rec6)
                self.__d.addForme(self.__rec5)
                self.__d.addForme(self.__cer)
                self.__d.addForme(self.__rec4)
                self.__d.addForme(self.__rec3)
                self.__d.addForme(self.__rec2)
                self.__d.addForme(self.__rec1)
        # Méthode qui permet d'afficher un élément contenu dans l'objet dessin
        def afficherElement(self,nb):
                self.__d.afficherElement(self,nb)
############################ Fin de la classe ##################################
```

c. La classe FenPrincipale

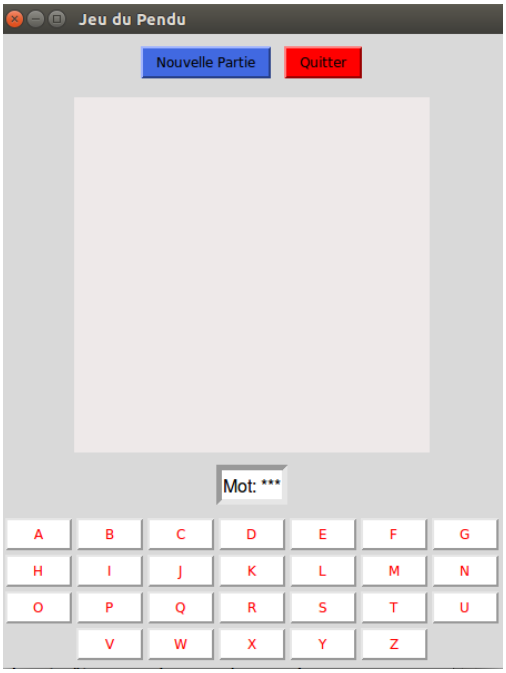
FenPrincipale
- motAffiche : string -nombre_manque : int -nombre_gagne : int
+ init () + newPartie() + chargerMots + nouveauMot () : motAffiche : string + traitement (l :string) + finPartie (b :boolean) +afficheElement(nb :int) + effacer ()

- Le code de la classe ***FenPrincipale***

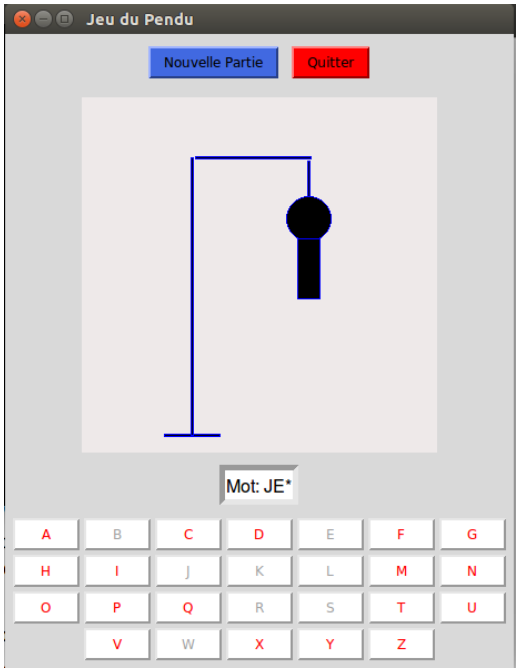
```
------------------------------------------------------------------------
Classe FenPrincipale heritant de TK et qui utilise les classes précédentes |
------------------------------------------------------------------------
"""
class FenPrincipale(Tk):
        def __init__(self):
                Tk.__init__(self)
                self.title('Jeu du Pendu')
                # Frame qui contient le menu
                f1=Frame(self)
                f1.pack(side=TOP,padx=5,pady=5)
                boutonNew = Button(f1, text ='Nouvelle Partie',bd=2, relief=RAISED,  bg ='royal' \
                        'blue', overrelief=RIDGE, command =self.newPartie).pack(side=LEFT,padx=5,pady=5)
                boutonQuit = Button(f1, text ='Quitter',bd=2, relief=RAISED,  bg ='re'\
                        'd', overrelief=RIDGE, command =self.destroy).pack(side=LEFT,padx=5,pady=5)
                #Canevas d'affichage des formes (le pendu)
                self.__zoneAffichage = ZoneAffichage(self,320,320,'snow2')
                self.__zoneAffichage.pack(padx=5, pady=5)
                #Label qui va servir à afficher le mot à deviner
                self.__labelMot=Label(self,bd=5, relief=SUNKEN, bg='white',font=('Helvetica', 12))
                self.__labelMot.pack(padx=5,pady=5)
                # Frame qui contient les buttons du clavier virtuel
                f2=Frame(self)
                f2.pack(side=TOP,padx=5,pady=5)
                #Création des boutons du clavier
                self.__buttons = []
                for i in range(26):
                        l=chr(ord('A')+i)
                        self.__buttons.append(MonButton(f2,self,l,4))
                        self.__buttons[i].config(command=self.__buttons[i].cliquer)

                # Affichage du clavier virtuel
                for i in range(3):
                        for j in range(7):
                                self.__buttons[j+7*i].grid(padx=1,pady=1,row=i,column=j)
                for i in range(1):
                        for  j in range(5):
                                self.__buttons[21+j].grid(padx=1,pady=1,row=4,column=j+1)

                #on fait appel aux méthodes chargerMots et newPartie dans le constructeur de la classe
                # pour pouvoir gérer l'initialisation des variables nécessaires pour jouer une partie
                self.chargerMots()
                self.newPartie()

        """
        ------------------------------------------------------------
        Méthode newPartie pour initialiser les paramètres du jeu    |
        ------------------------------------------------------------
        """
    def newPartie(self):
            self.effacer() # on efface le canvas
            self.__mot=self.nouveauMot() # on récupère le mot tiré au hasard
            print(self.__mot)       # on va afficher le mot dans le console (pour pouvoir tester le fonctionnement de code)
            self.__motAffiche=len(self.__mot)*'*' #On affiche les asterix pour cacher le mot à deviner
            self.__labelMot.config(text='Mot: '+self.__motAffiche,bg='white')
            self.__nbManques=0 # va servir à afficher le bon élément du pendu et à determiner la fin de partie
            self.__nbGagne=0   # permet de determiner la fin de partie
            self.__lesmots=len(self.__mot)*['*']
            for b in self.__buttons:   #activation de tous les boutons du clavier virtuel
                    b.config(state=NORMAL)
            """
            ---------------------------------------------------------------------------
            Méthode chargerMots pour récuperer les mots contenus dans le fichier mots.txt  |
            ---------------------------------------------------------------------------
            """
    def chargerMots(self):
            fich=open('mots.txt','r')
            contenu=fich.read()
            self.__mots=contenu.split('\n')
            fich.close()
```

```
        ------------------------------------------------------------------
        Méthode nouveauMot pour séléctionner au hasard un mot du fichier txt |
        ------------------------------------------------------------------
        """
    def nouveauMot(self):
        return self.__mots[randint(0,len(self.__mots))]
        """
        ------------------------------------------------------------------------------------
        Méthode traitement qui constitue le coeur du programme en prenant en entrée une lettre |
        pour ensuite décider de l'affichage du pendu ou non en fonction du résultat du clic    |
        ------------------------------------------------------------------------------------
        """
    def traitement(self,letr):
        cpt=0                                   # un compteur pour determiner le nombre de fois où la lettre apparait
        for i in range(0,len(self.__mot)):      # on parcours le mot et teste si la lettre s'y trouve ou pas
            if self.__mot[i]==letr:
                self.__lesmots[i]=letr
                cpt+=1
                self.__nbGagne+=1
        if cpt==0:                              # si la lettre n'est pas dans le mot on affiche un élément du pendu
            self.__nbManques+=1
            self.afficheElement(self.__nbManques)
            #tester si fin partie
            if self.__nbManques==10:
                self.finPartie(False)
        else:                                   # sinon on affiche une partie du mot à deviner
            self.__labelMot.config(text='Mot: '+''.join(self.__lesmots))
            # on teste si fin partie
            if self.__nbGagne==len(self.__mot):
                self.finPartie(True)
        """

          ------------------------------------------------------------------------------------
          Méthode finPartie qui prend un booléen et affiche le résultat final de la partie |
          ------------------------------------------------------------------------------------
          """
     def finPartie(self,booleen):
          if booleen:
              message=self.__mot+" - Bravo! Vous avez gagné."
              couleur='green'
          else:
              message="Vous avez perdu! Le mot était: "+self.__mot
              couleur='red'
          #on change le texte qui va s'afficher à la place du mot et la couleur du label
          self.__labelMot.config(text=message,bg=couleur)
          #on desactive le clavier
          for but in self.__buttons:
              but.config(state=DISABLED)
          """
          ------------------------------------------------------------------------------------
          Méthode afficherElement qui fait appelle à la méthode afficherElement de ZoneAffichage|
          ------------------------------------------------------------------------------------
          """
     def afficheElement(self,nb):
          self.__zoneAffichage.afficherElement(nb)
          """
          ------------------------------------------------------------
          Méthode qui sert juste à effacer le canevas                 |
          ------------------------------------------------------------
          """
     def effacer(self):
          """ Efface la zone graphique """
          self.__zoneAffichage.delete(ALL)

########################## Fin de la classe ######################
```

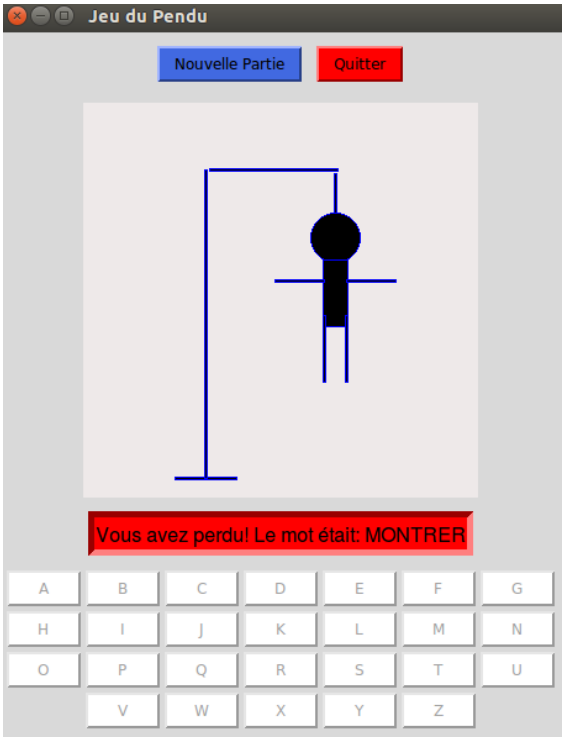
3. LES RESULTATS D'EXECUTION DU PROGRAMME



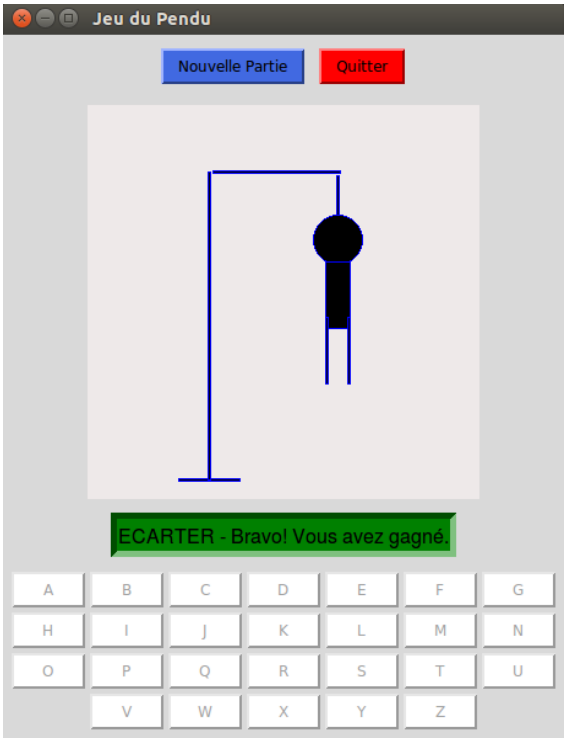
Lancement du jeu



Jeu en cours



Partie Perdue



Partie Gagnée

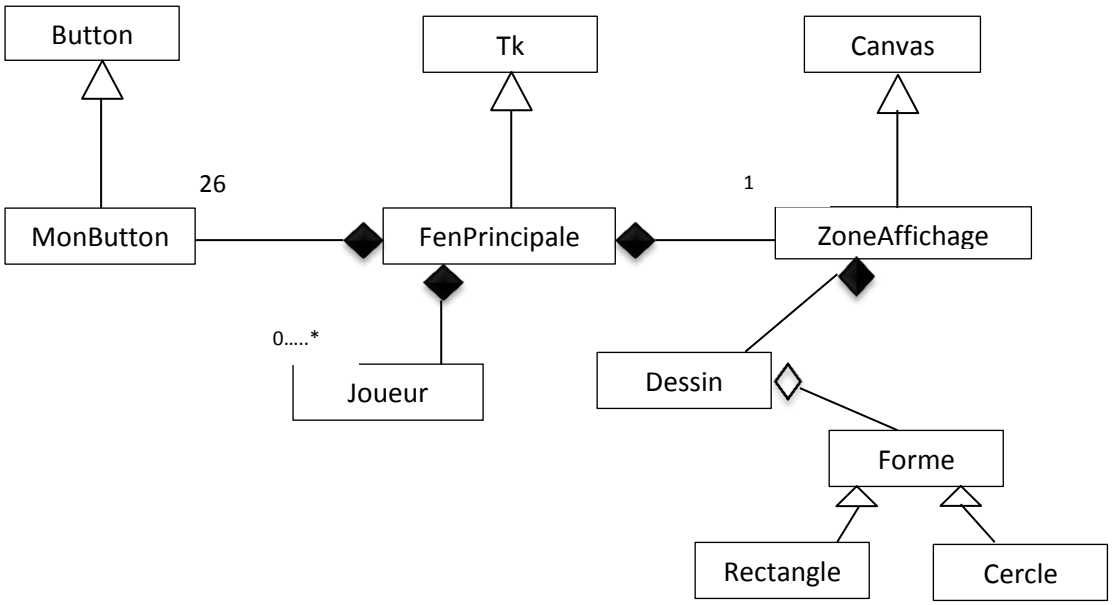
III. PARTIE OPTIONNELLE

Cette partie ajoute des fonctionnalités dans le jeu initial. Elle va gérer les scores des joueurs ainsi que la durée d'une partie. Cette durée (estimée en seconde) est choisie par le joueur lui-même au début d'une nouvelle partie permettra à ce dernier d'évaluer son niveau de rapidité pour deviner un mot.

Par ailleurs l'ergonomie du jeu a été améliorée avec des options supplémentaires. En effet, le joueur a accès à un bouton « help » qui affiche la première et la dernière lettre du mot. Il est même possible de se faire applaudir à la fin d'une partie (en cas de victoire).

NB : Le fichier Pendu3.py diffère du fichier Pendu2.py par une animation musicale. En fait, la bibliothèque « simpleaudio » qui a permis de faire l'animation musicale n'est connue que pour les versions de python3 antérieures à python3.5.

1. DIAGRAMME DES CLASSES UML



2. ATTRIBUTS ET METHODES DES CLASSES

Seules les nouvelles classes et celles qui ont été modifiées seront détaillées dans cette section.

a. La classe Joueur

Joueur
-nom : string -nombre_Partie: int -nombre _Partie_Gagne :int -scores :list<int>
+ init (nom :string) + set_nom (n : string) + get_nom() : n : string + set_scores(b : boolean) + get_scores() : l : list +set_nbPartie()

```
------------------------------------------------------------
Classe Joueur pour gérer les scores des joueurs             |
------------------------------------------------------------
"""
class Joueur:
        def __init__(self,nom):
                self.__nom=nom # nom du Joueur
                self.__nbPartie=0 # on initialise le nombre de partie jouée à 0
                self.__nbPartieGagne=0 # on initialise le nombre de partie gagnée à 0
                # liste contenant le nombre de parties jouées,parties gagnées,parties perdues
                self.__scores=[0,0,0]

        ####### getters et setters ########
        def getNom(self):
                return self.__nom

        def getScores(self):
                        return self.__scores

        def setNom(self,new_nom):
                self.__nom=new_nom

        def setNbPartie(self): # permet d'incrémenter le nombre de partie a chaque nouvelle partie
                self.__nbPartie+=1

        # setScores permet de mettre à jour la liste __scores à la fin de chaque partie
        # elle prend en entrée un booleen pour savoir si la partie est gagnée ou perdue
        def setScores(self,booleen):
                self.setNbPartie()
                if booleen:
                        self.__nbPartieGagne+=1 #on incrémente le nombre de partie gagné
                self.__scores[0]=self.__nbPartie
                self.__scores[1]=self.__nbPartieGagne
                self.__scores[2]=self.__nbPartie-self.__nbPartieGagne
```

b. La classe ZoneAffichage

ZoneAffichage
+ init (parent: string, w: int, h: int, couleur :string) + afficherElement(nb :int) +afficherCoupe()

- Code de la classe ***ZoneAffichage***

```
"""
-------------------------------------------------------------
Classe ZoneAffichage pour afficher les formes géométriques  |
-------------------------------------------------------------
"""

class ZoneAffichage(Canvas):
        def __init__(self,parent, w, h, c):
                Canvas.__init__(self, width = w, height = h, bg = c)

        # méthode qui affiche une image du pendu
        def afficherElement(self,nb):
                self.im='ImagesPendu/pendu'+str(nb)+'.gif'
                self.image = PhotoImage(file = self.im)
                self.create_image(0, 0, anchor=NW, image =  self.image)
                self.config(height=self.image.height(),width=self.image.width())

        # méthode qui affiche  la coupe quand la partie est gagnée
        def afficherCoupe(self):
                self.im='ImagesPendu/coupe.gif'
                self.image = PhotoImage(file = self.im)
                self.create_image(0, 0, anchor=NW, image =  self.image)
                self.config(height=self.image.height(),width=self.image.width())

######################### Fin de la classe ##########################################
```

c. La classe FenPrincipale

FenPrincipale
- motAffiche : string -nombre_manque : int -nombre_gagne : int - joueurs :list<string> -nomJoueur :string
+ init () + newPartie() +debutJeu() +update_clock() + chargerMots + nouveauMot () : motAffiche : string + traitement (l :string) + finPartie (b :boolean) +afficheElement(nb :int) +afficheCoupe() +playAudio(s :string) + effacer () +help() +chercherJoueur(nom:string) +infoScore()

- Code de la classe ***FenPrincipale***

```
"""
-------------------------------------------------------------------------
Classe FenPrincipale heritant de TK et qui utilise les classes précédentes |
-------------------------------------------------------------------------
"""
class FenPrincipale(Tk):
        def __init__(self):
                #liste des joueurs pendant toute l'execution du programme
                self.__joueurs=[]
                self.__nomJoueu=''
                Tk.__init__(self)
                self.title('Jeu du Pendu')
                #Création de la première Frame pour le menu
                f1=Frame(self,bg='white')
                f1.pack(side=TOP,padx=5,pady=5)
                boutonNew = Button(f1, text ='Nouvelle Partie',bd=2, relief=RAISED,  bg ='royal blu'\
                        'e', overrelief=RIDGE, command =self.debutJeu).pack(side=LEFT,padx=5,pady=5)
                boutonHelp = Button(f1, text ='Help',bd=2, relief=RAISED,  bg ='gree'\
                        'n', overrelief=RIDGE, command =self.help).pack(side=LEFT,padx=5,pady=5)
                boutonInfo = Button(f1, text ='Infos Score',bd=2, relief=RAISED,  bg ='yello'\
                        'w', overrelief=RIDGE, command =self.infosScore).pack(side=LEFT,padx=5,pady=5)
                boutonQuit = Button(f1, text ='Quitter',bd=2, relief=RAISED,  bg ='re'\
                        'd', overrelief=RIDGE, command =self.destroy).pack(side=LEFT,padx=5,pady=5)
                #Création de la Frame  pour afficher le score et le temps
                fscore=Frame(self,bg='white')
                fscore.pack(side=TOP,padx=5,pady=5)
                self.__labelscore=Label(fscore,bd=2, relief=RAISED, bg='white',font=('Helvetica', 12))
                self.__labelscore.pack(side=LEFT,padx=5,pady=5)
                #timer
                self.__labeltime=Label(fscore,text='Temps restant: ',bd=2, relief=RAISED, bg='white',font=('Helvetica', 12))
                self.__labeltime.pack(side=LEFT,padx=5,pady=5)
                #Canevas d'affichage
                self.__zoneAffichage = ZoneAffichage(self,320,320,'snow2')
                self.__zoneAffichage.pack(padx=5, pady=5)

                #Partie où l'on affiche le mot et le résultat final
                self.__labelMot=Label(self,bd=5, relief=SUNKEN, bg='white',font=('Helvetica', 12))
                self.__labelMot.pack(padx=5,pady=5)
                #les buttons du clavier virtuel
                f2=Frame(self)
                f2.pack(side=TOP,padx=5,pady=5)
                self.__buttons = []
                for i in range(26):
                        l=chr(ord('A')+i)
                        self.__buttons.append(MonButton(f2,self,l,4))
                        self.__buttons[i].config(command=self.__buttons[i].cliquer)
                # Affichage du clavier
                for i in range(3):
                        for j in range(7):
                                self.__buttons[j+7*i].grid(padx=1,pady=1,row=i,column=j)
                for i in range(1):
                        for  j in range(5):
                                self.__buttons[21+j].grid(padx=1,pady=1,row=4,column=j+1)

                #on fait appel aux méthodes chargerMots et debutJeu dans le constructeur de la classe
                # pour pouvoir gérer l'initialisation des variables nécessaires pour jouer une partie
                self.chargerMots()
                self.debutJeu()
```

```
        """
        ------------------------------------------------------------
        Méthode newPartie pour initialiser les paramètres du jeu    |
        ------------------------------------------------------------
        """
    def newPartie(self):
        #On s'assure que le joueur a donné son son nom et son niveau (temps de la partie souhaitée)
        if self.nom.get() and self.timeLevel.get() and type(int(self.timeLevel.get()))==type(1) and int(self.timeLevel.get())>0:
            self.fenStart.withdraw() # on cache la fenêtre contenant le formulaire d'ouverture
            self.deiconify()               # on réactive la fenêtre principale
            self.__mot=self.nouveauMot() # on tire un mot aléatoire
            self.__nomJoueur=self.nom.get() # le nom du joueuer qui joue la partie
            print("mot: ",self.__mot)
            print("nom: ",self.__nomJoueur)
            #on ajoute le joueur dans la liste s'il n'y est pas encore sinon on met à jours ses infos
            if self.chercherJoueur(self.nom.get())[0]==False:
                self.__joueurs.append(Joueur(self.nom.get()))

            self.__motAffiche=len(self.__mot)*'*'
            self.__labelMot.config(text='Mot: '+self.__motAffiche,bg='white')
            self.__nbManques=0
            self.__nbGagne=0
            self.__lesmots=len(self.__mot)*['*']
            self.effacer() # on efface le canvas
            for b in self.__buttons:   #activation de tous les boutons du clavier virtuel
                b.config(state=NORMAL)
            #gestion du chrono (on récupère le temps entré par le joueur et on initialise
            # les variables tempsTotal et tempsRestant)
            self.tempsTotal=datetime.datetime.now() + datetime.timedelta(seconds=int(self.timeLevel.get()))
            self.__tempsRestant=int(self.timeLevel.get())
            #on fait appelle à la méthode update_clock définie plus bas
            self.update_clock()
            self.__labelscore.config(text=self.__nomJoueur) # on affiche "Succès:" dans le labelscore



        """
        ---------------------------------------------------------------------
        Méthode debutJeu qui affiche un formulaire dès l'excution du jeu|
        pour demander le nom et le niveau de jeu souhaité du joueur     |
        ---------------------------------------------------------------------
        """
    def debutJeu(self):
        self.withdraw() #on cache la fenetre principale
        # on crée un toplevel qui va demander le nom du joueur et son niveau
        self.fenStart = Toplevel(self, bd =5, relief =RAISED)
        self.fenStart.geometry('400x220+100+250')
        self.fenStart.title('Niveau')
        Label(self.fenStart,text="Nom Joueur:", font="arial 12 bold",bg='w'\
                'hite').grid(padx=5, pady=15, row=0 ,column=0)
        self.nom=Entry(self.fenStart,font="arial 15 ",width=10)
        self.nom.grid(padx=5,row=0 ,column=1)
        Label(self.fenStart,text="Temps souhaité(en s):", font="arial 12 bold",bg='w'\
                'hite').grid(padx=5, pady=15, row=1 ,column=0)
        self.timeLevel=Entry(self.fenStart,font="arial 15 ",width=8)
        self.timeLevel.grid(padx=5,row=1 ,column=1)
        Button(self.fenStart,text='commencer',bd=2, relief=RAISED,  overrelief=RIDGE,bg ='royal'\
        ' blue',command=self.newPartie).grid(padx=5,row=2 ,column=1)
        Button(self.fenStart,text='Quitter',bd=2, relief=RAISED,  overrelief=RIDGE,bg ='re'\
                'd',command=self.destroy).grid(padx=5,pady=10,row=3 ,column=1)
        # On appelle la méthode newPartie
        self.newPartie()
```

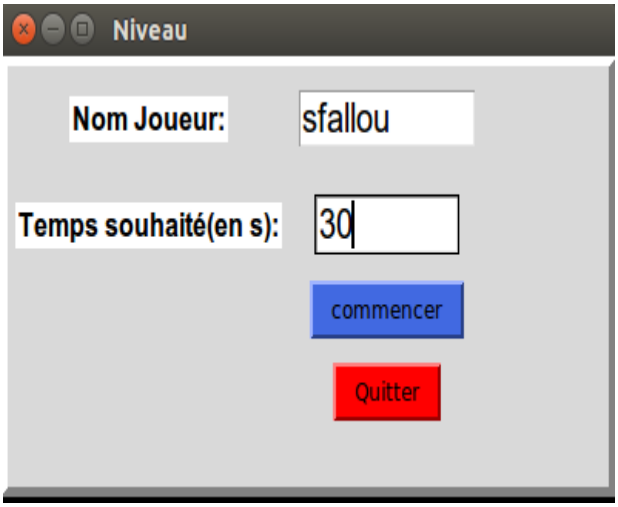
```
        """
        ------------------------------------------------------------
        Méthode update_clock pour gérer le chrono                   |
        ------------------------------------------------------------
        """
def update_clock(self):
        temps_depart = self.tempsTotal - datetime.datetime.now()
        m,s = temps_depart.seconds/60,temps_depart.seconds%60
        self.__labeltime.configure(text="Temps restant: "+"%02d:%02d"%(m,s))
        #on vérifie si le temps n'est pas encore fini
        if self.__tempsRestant>1:
                self.after(1000, self.update_clock) #et on actualise le chrono après chaque seconde
                self.__tempsRestant-=1                          # on décrémente tempsRestant
        #sinon on teste si la partie a été gagné ou perdu en vérifiant si tempsRestant=-100
        # correspondant à la valeur donnée à tempsRestant dans la méthode finPartie
        elif self.__tempsRestant!=-100:
                self.finPartie(False)

        """
        ------------------------------------------------------------------------------
        Méthode chargerMots pour récuperer les mots contenus dans le fichier mots.txt  |
        ------------------------------------------------------------------------------
        """
def chargerMots(self):
        fich=open('mots.txt','r')
        contenu=fich.read()
        self.__mots=contenu.split('\n')
        fich.close()




        """
        ------------------------------------------------------------------
        Méthode nouveauMot pour séléctionner au hasard un mot du fichier txt |
        ------------------------------------------------------------------
        """
def nouveauMot(self):
        return self.__mots[randint(0,len(self.__mots))]

        """
        ------------------------------------------------------------------------------------------
        Méthode traitement qui constitue le coeur du programme en prenant en entrée une lettre |
        pour ensuite décider de l'affichage du pendu ou non en fonction du résultat du clic    |
        ------------------------------------------------------------------------------------------
        """
def traitement(self,letr):
        if self.__tempsRestant==1:
                self.finPartie(False)
        cpt=0
        for i in range(0,len(self.__mot)):
                if self.__mot[i]==letr:
                        self.__lesmots[i]=letr
                        cpt+=1
                        self.__nbGagne+=1
        if cpt==0:
                self.__nbManques+=1
                self.afficheElement(self.__nbManques)
                #tester si fin
                if self.__nbManques==8:
                        self.finPartie(False)
        else:
                self.__labelMot.config(text='Mot: '+''.join(self.__lesmots))
                if self.__nbGagne==len(self.__mot):
                        self.finPartie(True)
```

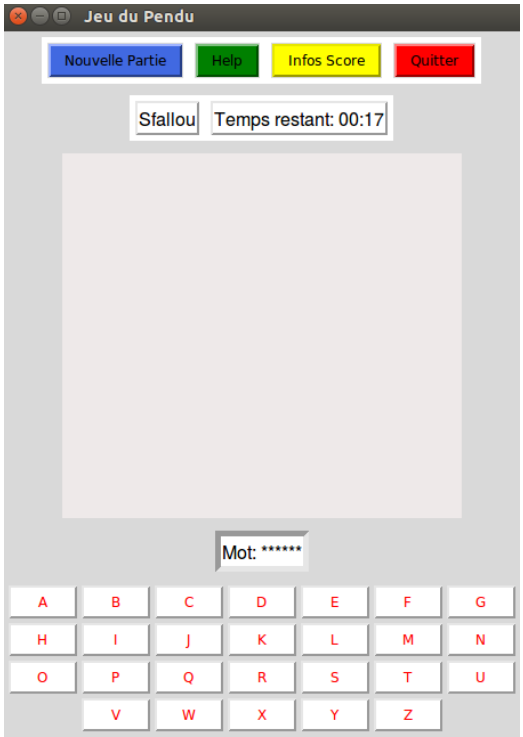
```
        ------------------------------------------------------------------------------
        Méthode finPartie qui prend un booléen et affiche le résultat final de la partie |
        ------------------------------------------------------------------------------
        """
    def finPartie(self,booleen):
        if booleen:
                message=self.__mot+" - Bravo! Vous avez gagné."
                couleur='green'
                self.__tempsRestant=-100 #on fixe temps restant à -100 pour arrêter le chrono
                #on affiche la coupe
                self.afficheCoupe()
                #son (applaudissement) à jouer
                audio="clap.wav"
        else:
                message="Vous avez perdu! Le mot était: "+self.__mot
                couleur='red'
                self.__tempsRestant=-100
                #On affiche le pendu complet
                for i in range(1,9):
                        self.afficheElement(i)
                #son (buzzer) à jouer
                audio="Buzzer.wav"
        #on actualise le message et la couleur du labelmot
        self.__labelMot.config(text=message,bg=couleur)
        #on desactive le clavier
        for but in self.__buttons:
                but.config(state=DISABLED)
        #on met à jour le score du joueur
        self.chercherJoueur(self.__nomJoueur)[1].setScores(booleen)
        # on actualise labelscore
        self.__labelscore.config(text=str(self.__nomJoueur))
        # on appelle la methode playAudio pour jouer le son
        self.playAudio(audio)
```

```
        """
        ------------------------------------------------------------------------------
        Méthode afficherElement qui fait appelle à la méthode afficherElement de ZoneAffichage|
        ------------------------------------------------------------------------------
        """
    def afficheElement(self,nb):
        self.__zoneAffichage.afficherElement(nb)

        """
        ------------------------------------------------------------------------------
        Méthode afficherCoupe qui fait appelle à la méthode afficherCoupe de ZoneAffichage    |
        ------------------------------------------------------------------------------
        """
    def afficheCoupe(self):
        self.__zoneAffichage.afficherCoupe()
        """
        ------------------------------------------------------------------------------
        Méthode playAudio qui joue le son entré en paramètre                          |
        ------------------------------------------------------------------------------
        """
    def playAudio(self,audio):
        wave_obj = sa.WaveObject.from_wave_file(audio)
        play_obj = wave_obj.play()
        play_obj.wait_done()

        """
        ------------------------------------------------------------
        Méthode qui sert juste à effacer le canevas                |
        ------------------------------------------------------------
        """
    def effacer(self):
        """ Efface la zone graphique """
        self.__zoneAffichage.delete(ALL)

        """
```

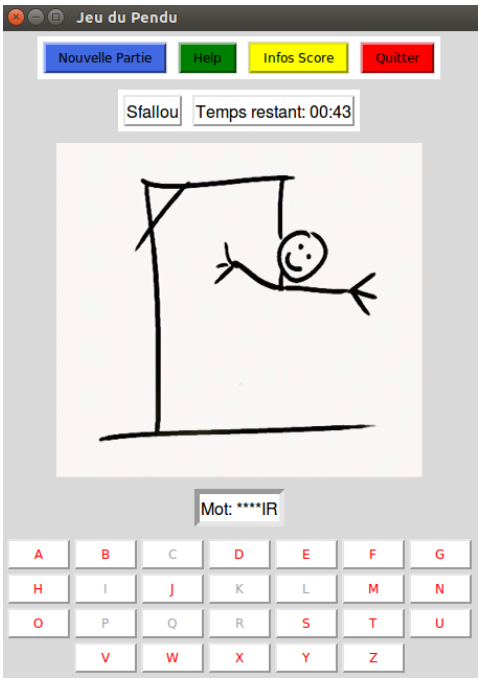
```
        """
        ------------------------------------------------------------------------
        Méthode help qui sert d'indication en donnant les lettre extremes du mot |
        ------------------------------------------------------------------------
        """
    def help(self):
        showinfo('Indication','Le mot commence par : '+self.__mot[:1]+' et termine par : '+self.__mot[-1:])

        """
        ------------------------------------------------------------------------------
        Méthode chercherJoueur qui renvoie le joueur de la liste dont le nom est donné|
        ------------------------------------------------------------------------------
        """
    def chercherJoueur(self,nom):
        for joueur in self.__joueurs:
            if joueur.getNom()==nom:
                return True,joueur
        return False,None

        """
        ------------------------------------------------------------------------
        Méthode qui renvoie les scores de tous les joueurs de la partie        |
        ------------------------------------------------------------------------
        """
    def infosScore(self):
        string=""
        for j in self.__joueurs:
            scores=j.getScores()
            string+="\nJoueur : "+j.getNom()+"\nParties jouées: "+str(scores[0])+"\nParties"\
             "gagnées: "+str(scores[1])+"\nParties Perdues: "+str(scores[2])
            string+="\n----------------"
        showinfo('Scores','\n'+string)

########################## Fin de la classe ######################
```

3. LES RESULTATS DE L'EXECUTION DU PROGRAMME



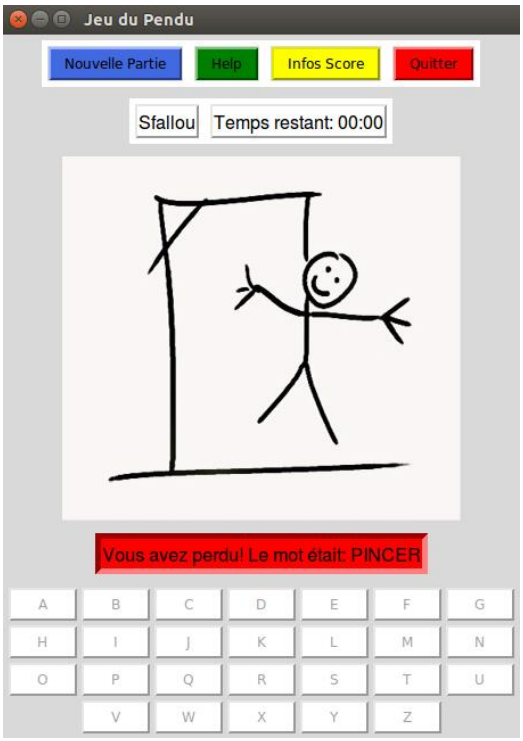


Lancement du jeu



Jeu en cours

help



Partie perdue

Partie gagnée

Information sur le score

IV. ANNEXE

Diagrammes des classes UML de Forme, Rectangle, Cercle et Dessin

Forme
- xc:double - yc:double - c:couleur
+init(x:double,y:double,c:couleur) +set_centre(x:double,y:double) +get_centre():xdouble,y:double +set_couleur(c:couleur) +get_couleur();c:couleur +deplacement(dx:double,dy:double)

Rectangle
- l:double - h:double
+init(x:double,y:double,c:couleur,l:double,h:double) +set_dim(l:double,h:double) +get_dim():ldouble,h:double +perimetre():p:double +surface():s:double

Cercle
- d:double
+init(x:double,y:double,c:couleur,d:double) +set_dim(d:double) +get_dim():d:double +perimetre():p:double +surface():s:double

Dessin
- formes : list
+init() +add_forme(f:Forme) +del_forme(i:int) +print_forme() +afficher(can:ZoneAffichage) +affchier_element(can:ZoneAffichage,n:int) +select_forme(x:int,y:int)