École Centrale Lyon

UE INFO
Conception et Programmation Objet : TC-2
Rapport

Rapport TD5 - Jeu du Pendu

Élèves :
Groupe - B2B
Maria Carolina HEINRICHS
Pedro MAIA DA SILVA

Enseignant :
Mehdi AZAOUZI

4 avril 2023

Table des matières

1 Introduction

À travers ce travail dirigé, on a l'objectif de créer une programmation orientée au objet vers un Jeu du Pendu.

En rappel, ce divertissement consiste à tenter de trouver un mot qui est présenté de manière cachée (chaque lettre est remplacée par le symbole '*'). Pour ce faire, le participant sélectionne une lettre sur le clavier virtuel. Si cette lettre est incluse dans le mot, alors le mot caché est révélé en affichant la lettre choisie. Si en revanche la lettre sélectionnée ne se trouve pas dans le mot, alors le nombre de tentatives ratées augmente d'une unité et un élément additionnel est dessiné sur la potence. Le joueur remporte la partie s'il réussit à trouver le mot avant que le dessin de la potence ne soit complet (au-delà de 10 tentatives manquées). La représentation 1 illustre l'interface que nous allons développer.

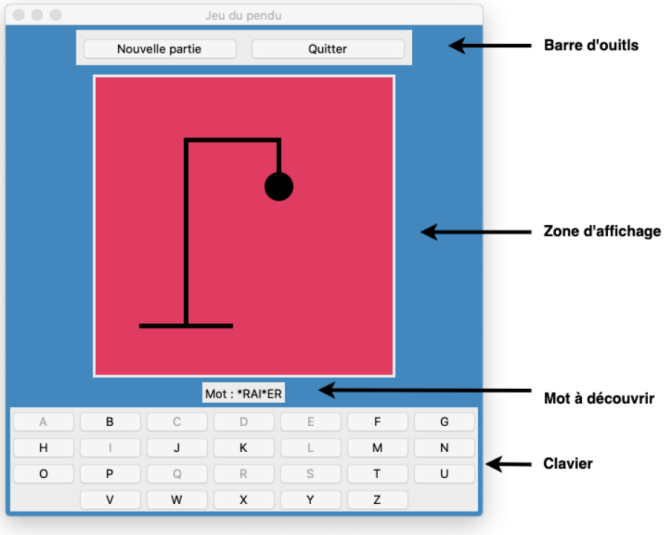

FIGURE 1 – Caption

Ainsi, à travers du Cahier de Charges donné, on a mis dans ce rapport le chemin suivi pour parvenir à l'amélioration du Jeu.

2 Cahier des Charges

Pour créer un bon fonctionnement du programme, on doit suivre le Cahier des Charges décrit ci-dessous.

On doit programmer l'interface statique en distinguant la classe Fenêtre Principale et la classe Zone Affichage pour obtenir une apparence proche de celle donnée en exemple, on doit avoir un placement des boutons à la disposition des boutons en grille. Après, l'application doit avoir une logique de jeu, des commandes de l'interface : Le Bouton "Quitter", Le bouton "Une Nouvelle partie" pour que partie qu'une puisse commencer que si le joueur appuie sur le bouton "Nouvelle partie"., Bouton Lettres "A...Z" où le fait d'appuyer sur une lettre du clavier virtuel doit provoquer un certain nombre d'actions et faire apparaître autant de fois que nécessaire la lettre cliquée dans le mot à découvrir, vérifier si la partie est Pendue, gagnée, ou si elle se poursuit et, pour finir, éventuellement,

compléter le dessin du pendu si la lettre n'est pas présente dans le mot (action traitée dans la partie suivante de cet énoncé).

Ensuite, pour le bon fonctionnement du Jeu, on doit programmer un traitement, une méthode dont l'objectif est de mettre à jour l'affichage du mot. Tel que si la partie est gagnée, bloquez l'utilisation du clavier, et affichez un texte qui indique au joueur que c'est gagné, sinon, la partie est Pendue quand le nombre de coups dépasse 10 (c'est le nombre d'éléments nécessaires pour dessiner entièrement le pendu). Bloquez alors l'utilisation du clavier, et affichez un texte qui indique au joueur que c'est Pendu !

Postérieurement, pour programmer le dessin du pendu, on doit créer un dessin progressif du pendu, au fur et à mesure des échecs du joueur. Et, ainsi, améliorer le code en développant le code qui permet au joueur de choisir les couleurs principales de l'application (par un menus, par des boutons...), en implémentant une technique "Undo" qui permet de revenir d'un ou plusieurs coups en arrière, au cours d'une partie, et an additionnant le score joueur Implémentez un système de sauvegarde des échecs et des succès d'un joueur (identifié par un pseudo demandé au joueur en début de partie) et un affichage de l'historique des parties du joueur et/ou de ses performances.

3 Principe des Solutions

Pour bien comprendre l'exercice,

En première lieu, on a créé un diagramme de classes UML 2, conçu à l'aide du site "diagrams.net", composant des classes qu'on aurait besoin pour respecter le Cahier des Charges. À partir d'une analyse des fonctionnalités que doit fournir l'application, on a identifié les classes nécessaires.

De cette manière, on a dessiné :

- Les classe « FenPrincipale », «Mon Boutton», «ZoneAffichage», «Dessin» et «Menu»

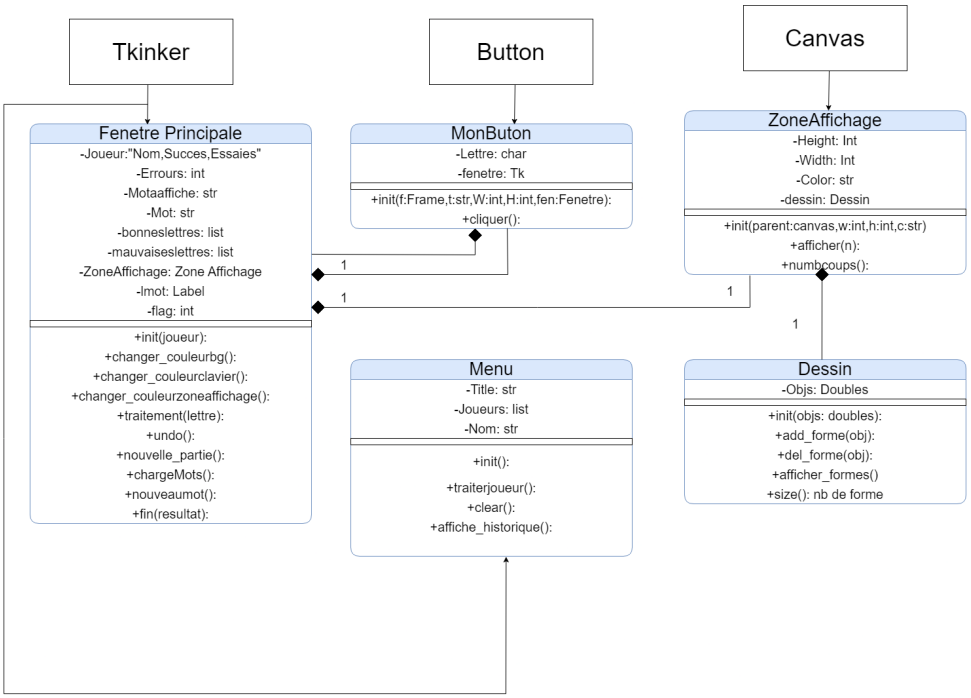

FIGURE 2 – Diagramme UML du Jeu de Pendu créée.

4 Implémentation :

Pour mise en ouvre l'exercice, on a implémenté le programme à partir du Cahier des Charges et, en conséquence, à partir du Diagramme UML (figure 2). On a utilisé la language Python 3.9.13.

On a utilisé des Modules :

1. Module formes :

   Cette ligne du Code nous donne la possibilité d'accéder aux classes du document formes.py, afin d'employer les procédures pour représenter des figures géométriques du Jeu du Pendu sur le Canvas.

2. Module tkinter

   Le module Tkinter nous octroie la possibilité d'utiliser les classes et les méthodes pour exploiter des composants graphiques, tels que des Frames, des boutons, des Labels et du Canvas, et pour afficher des fenêtres. C'est grâce à cette bibliothèque que nous pouvons créer des Interfaces Utilisateur Graphiques (GUI).

3. Module random

   On emploie la fonction randint de cette librairie pour produire un chiffre entier aléatoire qui correspond à un élément de la liste de mots. Le but est de sélectionner un terme inédit à chaque session de jeu.

On a aussi crée des Classes pour bien programmer le Jeu :

1. Classe « ZoneAffichage »

   Elle est une classe dérivée de la classe Canvas, qui permet de créer une zone graphique dans la fenêtre parent et de générer une liste contenant les figures géométriques qui constituent le Jeu du Pendu. Les paramètres w, h et c correspondent respectivement à sa largeur, sa hauteur et à la couleur de son arrière-plan.

   La méthode constructeur initie les attributs nécessaires pour créer un Canvas (la classe parente) et un objet de la classe "Dessin" pour contrôler une liste avec les figures du Jeu créées ensuite. Pour les objets des classes "Rectangles" et "Cercles", il est nécessaire de fournir les coordonnées des sommets supérieurs gauche et inférieur droit du rectangle extérieur des formes que nous souhaitons ajouter, ainsi que leur couleur.

   La méthode "afficher" permet d'afficher une forme dans la zone graphique en fonction du nombre de tentatives ratées par le joueur, et la méthode "numbcoups " retourne le nombre d'éléments dans la liste des figures, c'est-à-dire le nombre maximum de tentatives possibles.

2. Classe « MonBouton » :

   Cette Classe hérite de la classe Button et a pour but de gérer les boutons du clavier virtuel. Dans le constructeur de la classe, on retrouve l'appel du constructeur de la classe de base ainsi que des attributs liant le bouton à la fenêtre principale et à la lettre de l'alphabet.

   Lorsque la méthode « cliquer » est appelée, la condition du bouton est modifiée : il devient plus clair et ne peut plus être cliqué. En outre, nous vérifions si la lettre existe dans le mot à trouver et prenons les mesures appropriées en conséquence.

3. Classe « Menu » : Cette classe est responsable de la création de la fenêtre Menu du jeu, dans laquelle nous pouvons démarrer une nouvelle partie, voir l'historique des joueurs, supprimer cet historique et quitter le jeu.
   — « Méthode constructeur » : Méthode responsable de l'initialisation de la classe et de ses boutons, en plus de permettre la configuration des lecteurs qui seront gérés par la fonction suivante.
   — « Méthode TraiterJoueur » : Méthode chargée d'initialiser le joueur et de l'inclure dans l'historique lors du démarrage d'un nouveau jeu, entraînant la fermeture de la fenêtre du menu et l'apparition de la fenêtre principale, où le jeu est joué
   — « Méthode Clear » : Méthode chargée de nettoyer complètement les joueurs et leurs données respectives de l'historique des matchs en modifiant le fichier au format txt.
   — « Méthode Affichehistorique » : Méthode chargée de montrer l'historique des matchs de chacun des joueurs déjà inscrits, indiquant le nom, le nombre de matchs et le nombre de victoires de chacun.
4. Classe « FenPrincipale » :
   Cette Classe hérite de la classe Tk, son but est de définir la fenêtre principale sur laquelle on place des widgets pour rendre l'application jouable. Elle développe plusieurs attributs du Jeu, donc, on la sépare en plusieurs méthodes :
   — « Méthode constructeur » :
   Dans cette méthode, on observe l'initialisation de la classe de base et des attributs pour gérer le jeu. On peut également voir le code permettant d'effacer l'historique d'un joueur stocké dans les fichiers texte via les fonctions with, open, as et write.
   Au début, il n'y a aucun nom enregistré (pas encore de joueur) et les touches du clavier virtuel sont désactivées. Les widgets comprennent des cadres contenant des boutons (Nouvelle partie, Tous les scores, Quitter, Commencer, Votre score), des étiquettes (Entrez votre nom), un champ de saisie pour entrer le nom du joueur (Entry) et le clavier virtuel. Enfin, une zone graphique est créée pour afficher les formes du pendu et une étiquette pour le mot.
   — « Méthode Traitement » :
   Lorsqu'un bouton d'une lettre est sélectionné, si cette lettre est présente dans le mot à deviner, les caractères correspondants sont révélés. Sinon, on augmente le nombre d'erreurs et affiche une forme supplémentaire sur le Canvas. On vérifie également si le nombre maximum d'erreurs est atteint ou si le mot est entièrement deviné.
   — « Méthode Nouvelle Partie » :
   Cette méthode initialise une partie en sélectionnant un mot au hasard dans une liste et en affichant des astérisques en fonction de sa longueur. Au début, aucun élément n'est affiché sur le Canvas et tous les boutons de lettres sont activés.
   — « Méthode NouveauMot » :
   Cette méthode renvoie un mot aleatoire de la liste de mots.
   — « Méthode ChargeMots » :
   Dans cette méthode, on ouvre le fichier de mots et on ajoute chaque ligne, qui correspond à un mot, à la liste de mots.

— « Méthode Fin » : La partie se termine si le nombre de manques atteint le maximum ou si le mot est entièrement découvert. Les détails de la partie, y compris le nom du joueur, le mot joué, le résultat et le score actualisé, sont enregistrés dans les listes de toutes les parties. Les boutons des lettres sont désactivés et un message s'affiche pour informer si le joueur a gagné ou Pendu, avec la réponse. Enfin, deux fichiers txt sont créés : l'un pour toutes les parties et l'autre pour le joueur actuel.
— « Méthode changer_couleurbg » : Méthode chargée de changer la couleur de fond à l'aide de la fonction "colourchooser" qui a été importée de la bibliothèque TK.
— « Méthode changer_couleurclavier » : Méthode chargée de changer la couleur du clavier à l'aide de la fonction "colorchooser" importée de la bibliothèque TK.
— « Méthode changer_couleurzoneaffichage » : Méthode chargée de changer la couleur du clavier à l'aide de la fonction "colorchooser" importée de la bibliothèque TK.
— « Méthode undo » : Méthode chargée d'annuler le dernier coup. Pour cela, plusieurs variables et attributs ont été utilisés afin de permettre de changer toutes les caractéristiques nécessaires pour tourner complètement un coup, donnant au joueur la possibilité de refaire aussi bien les bons que les mauvais coups.

5 Le Code ("Devoir2_MAIADASILVA_HEINRICHS.py") :

```
from tkinter import *
from random import randint
from formes import *
from tkinter import colorchooser

class ZoneAffichage(Canvas): # classe de la zone d'affichage
  def __init__(self, parent, w, h, c):
    Canvas.__init__(self, master=parent, width=w, height=h, bg=c) # création de la zone d'affichage avec les paramètres donnés

    ymax= h;

    self.__dessin = Dessin() # Objet de la classe Dessin qui contient les formes à dessiner

    # Création des formes qui composent le pendu
    f = Rectangle(100,ymax-50,100,5,couleurs[5])
    self.__dessin.add_forme(f)

    f = Rectangle(100,ymax-150,5,200,couleurs[5])
    self.__dessin.add_forme(f)

    f = Rectangle(148,ymax-250,100,5,couleurs[5])
    self.__dessin.add_forme(f)

    f = Rectangle(200,ymax-236,5,30,couleurs[5])
    self.__dessin.add_forme(f)

```

```
27      f = Cercle(200,ymax-200,30,couleurs[5])
28      self.__dessin.add_forme(f)
29      f = Rectangle(200,ymax-170,15,50,couleurs[5])
30      self.__dessin.add_forme(f)
31
32      f = Rectangle(178,ymax-170,30,5,couleurs[5])
33      self.__dessin.add_forme(f)
34
35      f = Rectangle(220,ymax-170,30,5,couleurs[5])
36      self.__dessin.add_forme(f)
37
38      f = Rectangle(195,ymax-130,5,40,couleurs[5])
39      self.__dessin.add_forme(f)
40
41      f = Rectangle(205,ymax-130,5,40,couleurs[5])
42      self.__dessin.add_forme(f)
43
44    def afficher(self,n): #Associe la zone d'affichage a l'image du pendu
       correspondant au nombre d'erreurs
45      self.__dessin.affiche_formes(self,n)
46
47    def numbcoups(self): #Affiche le nombre de coups possible = nombre d'
       image  - 1
48      return self.__dessin.size()
49
50
51  class MonBouton(Button): # Classe des boutons du clavier, avec la lettre
       correspondante
52    def __init__(self, parent, fenetre, lettre, w):
53      Button.__init__(self, master=parent, text=lettre, width=w) # cré
       ation du bouton avec la lettre correspondante
54      self.__lettre = lettre
55      self.__fenetre = fenetre #MonBoutonLettre est associé à la fenêtre
       de jeu
56
57    def cliquer(self): # fonction qui est appelée quand on clique sur le
       bouton
58      self.config(state=DISABLED) # Après avoir cliqué sur le bouton, il
       ne peut plus être cliqué et il devient plus clair
59      self.__fenetre.traitement(self.__lettre) # On appelle la fonction
       traitement de la fenêtre de jeu
60
61  class Menu(Tk): # classe de la première fenêtre, le menu
62    def __init__(self):
63      Tk.__init__(self)
64      self.title('Menu')
65
66      f = open('historique.txt', 'r') # ouverture du fichier contenant l'
       historique des scores
67      s = f.read() #Conversion du fichier en chaine de caractères
68      self.__joueurs = s.split('\n') #Création d'une liste contenant les
       noms des joueurs
69      #["Joueur1, nbparties, nbvictoire"]
70      f.close()
71
72      Label(self, text='Entrez Votre Nom!').pack(side=TOP , padx=5, pady
       =5) # Création d'un label pour demander le nom du joueur
```

```
73
74      f1 = Frame(self) # Création d'une frame pour contenir le champ de
        saisie et le bouton     f1.pack(side=TOP, padx=5, pady=5)
75
76      self.nom = StringVar() # Création d'une variable de type StringVar
        pour contenir le nom du joueur
77      text = Entry(f1, textvariable=self.nom, bg ='white',fg='black') # Cr
        éation d'un champ de saisie pour le nom du joueur
78      text.focus_set() # Le curseur est placé dans le champ de saisie
79      text.pack(side = TOP, padx=5, pady=5)
80
81      # Création d'un bouton pour lancer la partie
82      Button(f1,text='Jouer',width=15,command=self.traiterjoueur).pack(
        side=TOP,padx=5,pady=5)
83
84      # Création d'un bouton pour quitter le menu
85      Button(f1, text='Quitter', width=15, command=self.destroy).pack(side
        =TOP, padx=5, pady=5)
86
87      # Création d'un bouton pour afficher l'historique des scores
88      Button(f1, text='Historique', width=15, command=self.
        affiche_historique).pack(side=TOP, padx=5, pady=5)
89
90      # Création d'un bouton pour effaer l'historique des scores
91      Button(f1, text='Effacer Historique', width=15, command=self.clear).
        pack(side=TOP, padx=5, pady=5)
92
93  def traiterjoueur(self): # Realise les actions en fonction du nom du
        joueur
94      for j in self.__joueurs: # On parcourt la liste des joueurs
95          if j.split(',')[0] == self.nom.get(): # Si le nom du joueur est
        dans la liste
96              self.destroy() # On détruit la fenêtre du menu
97              fen = FenPrincipale(j) # On crée la fenêtre de jeu
98              fen.mainloop() # On lance la fenêtre de jeu
99              return # On sort de la fonction
100     with open('historique.txt', 'a') as f: # Si le nom du joueur n'est
        pas dans la liste, on l'ajoute
101         f.write("\n" + self.nom.get() + ",0,0") # On ajoute le nom du
        joueur, le nombre de victoires et le nombre de parties
102     with open('historique.txt', 'r') as f: #On actualise la liste des
        joueurs
103         self.__joueurs = f.read().split('\n')
104         print(self.__joueurs)
105     self.destroy() # On détruit la fenêtre du menu
106     fen = FenPrincipale(self.__joueurs[-1]) # On crée la fenêtre de jeu
107     fen.mainloop() # On lance la fenêtre dex jeu
108
109 def clear(self): # Fonction qui efface l'historique des scores
110     open('historique.txt', 'w').close() # On ouvre le fichier en mode é
        criture et on le vide
111
112 def affiche_historique(self): # Fonction qui affiche l'historique des
        scores
113     text = open('historique.txt', 'r').read() # On ouvre le fichier en
        mode lecture et on le convertit en chaine de caractères
114     s = text.split('\n') # On crée une liste contenant les noms des
```

```
    joueurs
115     for e in s:       if e != '': #Condition pour ne pas afficher les
    lignes vides
116         Label(self, text = 'Nom : ' + e.split(',')[0] + ' Nombre de
    parties jouées : ' + e.split(',')[2] + ' Nombre de victoires : ' + e.
    split(',')[1]).pack(side=TOP , padx=5, pady=5) # Création d'un label
    pour afficher le nom du joueur et son taux de victoire
117
118 class FenPrincipale(Tk):
119   def __init__(self, joueur): # Création de la fenêtre de jeu
120     Tk.__init__(self)
121     self.title('Pendu')
122     self.__joueur = joueur # On récupère le nom du joueur
123     self.__errours = 0 # On initialise le nombre d'erreurs à 0
124     self.__motaaffiche = '' # On initialise le mot affiché à une chaine
    de caractères vide
125     self.__mot = '' # On initialise le mot à trouver à une chaine de
    caractères vide
126     self.__bonneslettres = [] # On initialise la liste des bonnes
    lettres à une liste vide
127     self.__mauvaiseslettres = [] # On initialise la liste des mauvaises
    lettres à une liste vide
128     f1 = Frame(self) #frame pour contenir les 2 boutons principaux
129     f1.pack(side=TOP, padx=5, pady=5)
130
131     #Creation du bouton pour quitter le jeu
132     Button(f1, text='Quitter', width=15, command=self.destroy).pack(side
    =LEFT, padx=5, pady=5)
133
134     #Creation du bouton pour lancer une nouvelle partie
135     Button(f1, text='Nouvelle Partie', width=15, command=self.
    nouvelle_partie).pack(side=LEFT, padx=5, pady=5)
136
137     #Creation du bouton pour undo une joue
138     Button(f1, text='Undo', width=15, command=self.undo).pack(side=LEFT,
     padx=5, pady=5)
139
140     #Creation du bouton pour changer le couleur du background
141     Button(f1, text='Couleur BG', width=15, command=self.
    changer_couleurbg).pack(side=LEFT, padx=5, pady=5)
142
143     #Creation du bouton pour changer le couleur du clavier
144     Button(f1, text='Couleur Clavier', width=15, command=self.
    changer_couleurclavier).pack(side=LEFT, padx=5, pady=5)
145
146     #Creation du bouton pour changer le couleur de la zone d'affichage
147     Button(f1, text='Couleur ZoneAffichage', width=15, command=self.
    changer_couleurzoneaffichage).pack(side=LEFT, padx=5, pady=5)
148
149     self.__zoneAffichage = ZoneAffichage(self,320,320,'snow2') # Cré
    ation de la zone d'affichage
150     self.__zoneAffichage.pack(side=TOP, padx=5, pady=5)
151     self.__zoneAffichage.afficher(0)
152     self.__lmot = Label(self, text='Mot : ') # Création d'un label pour
    afficher le mot à trouver
153     self.__lmot.pack(side=TOP)
154     self.__flag = 0 # On initialise le flag à 0
```

```
155
156     f2 = Frame(self) # Création d'une frame pour contenir le clavier
    f2.pack(side=TOP, padx=5, pady=5)
157
158     self.__boutons = [] # Création d'une liste pour contenir les boutons
     du clavier
159     for i in range(26): # Création des boutons du clavier
160       t = chr(ord('A')+i) # On crée une chaine de caractères contenant
    la lettre du bouton
161       self.__boutons.append(MonBouton(f2,self,t,4)) # On ajoute le
    bouton à la liste
162       self.__boutons[i].config(command = self.__boutons[i].cliquer) # On
     associe une fonction au bouton
163
164     for i in range(3): #Affiche les 3 premières lignes du clavier
165       for j in range(7):
166         self.__boutons[i*7+j].grid(row=i, column=j)
167
168     for j in range(5):
169       self.__boutons[21+j].grid(row=3, column=j+1)
170
171     self.nouvelle_partie() # On lance une nouvelle partie
172
173   def changer_couleurbg(self):
174     coulor = colorchooser.askcolor()[1]
175     self.configure(background= coulor)
176
177   def changer_couleurclavier(self):
178     coulor = colorchooser.askcolor()[1]
179     for i in range(26):
180       self.__boutons[i].configure(background= coulor)
181
182   def changer_couleurzoneaffichage(self):
183     coulor = colorchooser.askcolor()[1]
184     self.__zoneAffichage.configure(background= coulor)
185
186
187   def traitement(self,lettre):
188     k = 0
189     lettres = list(self.__motaaffiche) # On crée une liste contenant les
     lettres du mot affiché
190
191     for i in range(len(self.__mot)): # On parcourt le mot à trouver
192       if self.__mot[i] == lettre: # Si la lettre du mot à trouver est é
    gale à la lettre cliquée
193         k += 1 # On incrémente k
194         lettres[i] = lettre # On remplace les * par lettre cliquée
195
196     self.__motaaffiche = ''.join(lettres) # On convertit la liste en
    chaine de caractères
197
198     if k ==0: #si la lettre n'est pas dans le mot
199       self.__errours += 1 # On incrémente le nombre d'erreurs
200       self.__mauvaiseslettres.append(lettre) # On ajoute la lettre à la
    liste des mauvaises lettres
201       self.__flag = 0 # On remet le flag à 0
202       self.__zoneAffichage.afficher(self.__errours-1) # On affiche l'
```

```
    image correspondante
203     if self.__errours-1 >= self.__zoneAffichage.numbcoups(): # Si le
    nombre d'erreurs est supérieur au nombre d'images          self.fin(False)
204                 # False pour dire que le joueur a perdu
205     else: # Si la lettre est dans le mot
206       self.__lmot.config(text='Mot : '+self.__motaaffiche) # On affiche
    le mot avec les lettres trouvées
207       self.__bonneslettres.append(lettre) # On ajoute la lettre à la
    liste des bonnes lettres
208       self.__flag = 1 # On met le flag à 1
209
210       if self.__mot == self.__motaaffiche: # Si le mot est trouvé
211         self.fin(True) # True pour dire que le joueur a gagné
212
213   def undo(self): #Fonction pour undo le dernier coup
214     #Si le flag est à 1, on undo la derniere lettre bonne
215     if self.__flag == 1:
216       l = self.__bonneslettres.pop() #On enleve la derniere lettre bonne
     de la liste
217       self.__motaaffiche = self.__motaaffiche.replace(l,'*') #On
    remplace la lettre par un *
218       self.__lmot.config(text='Mot : '+self.__motaaffiche)
219
220       #On change l'etat du bouton
221       for b in self.__boutons:
222         if b['text'] == l:
223           b.config(state=NORMAL)
224
225     if self.__flag == 0:
226       l = self.__mauvaiseslettres.pop() #On enleve la derniere lettre
    mauvaise de la liste
227       self.__errours -= 1 #On enleve une erreur
228       #on delete l'image et on affiche la precedente
229       self.__zoneAffichage.delete(ALL)
230       self.__zoneAffichage.afficher(self.__errours-1)
231       for b in self.__boutons:
232         if b['text'] == l:
233           b.config(state=NORMAL)
234
235   def nouvelle_partie(self):
236     self.__zoneAffichage.delete(ALL) # pour supprimer tout ce qui se
    trouve sur le Canvas
237     self.__zoneAffichage.afficher(0) # On affiche l'image de départ
238     self.chargeMots() # On charge la liste des mots
239     self.__mot = self.nouveaumot() # On choisit un mot au hasard dans la
     liste des mots
240     self.__motaaffiche = len(self.__mot)*'*' # On crée une chaine de
    caractères contenant autant de * que de lettres dans le mot
241     self.__lmot.config(text='Mot : '+self.__motaaffiche) # On affiche le
     mot avec les * à la place des lettres
242     self.__errours = 1 # On remet le nombre d'erreurs à 1 (car on a déjà
     affiché l'image de départ)
243
244
245     for b in self.__boutons:
246       b.config(state=NORMAL) # On réactive tous les boutons du clavier
247
```

```
248   def chargeMots(self):
249     f = open('mots.txt','r') # On ouvre le fichier mots.txt en lecture
        s = f.read() # On lit le fichier
250     self.__mots = s.split('\n') # On convertit le fichier en liste
251     f.close() # On ferme le fichier
252
253   def nouveaumot(self):
254     return self.__mots[randint(0,len(self.__mots)-1)] # On choisit un
      mot au hasard dans la liste des mots
255
256
257
258
259
260   def fin(self,resultat):
261     for b in self.__boutons:
262       b.config(state=DISABLED) # On désactive tous les boutons du
      clavier
263
264     if resultat : # Si le joueur a gagné
265       self.__lmot.config(text=self.__mot + '------VICTORY!') # On
      affiche le mot et VICTORY!
266
267       f = open('historique.txt','r')  # On ouvre le fichier historique
      en lecture
268       s = f.read().split("\n") # On lit le fichier et on le convertit en
       liste
269       f.close() # On ferme le fichier
270
271       for i in range(len(s)):
272
273         if s[i] == self.__joueur: # On cherche le joueur dans la liste
274           s[i] = s[i].split(',')[0]+','+ str(int(s[i].split(',')[1])+1)+
      ','+str(int(s[i].split(',')[2])+1) # On incrémente le nombre de
      parties jouées et le nombre de parties gagnées
275           self.__joueur=s[i] # On met à jour le joueur
276
277       with open('historique.txt','w') as f: # On ouvre le fichier
      historique en écriture
278         f.write('\n'.join(s)) # On écrit la liste dans le fichier
279
280     else: # Si le joueur a perdu
281       self.__lmot.config(text='DEFEAT! Le mot était : '+self.__mot) # On
       affiche le mot et DEFEAT!
282
283       f=open('historique.txt','r') # On ouvre le fichier historique en
      lecture
284       s=f.read().split("\n") # On lit le fichier et on le convertit en
      liste
285       f.close() # On ferme le fichier
286
287       for i in range(len(s)):
288         if s[i] == self.__joueur:
289           s[i] = s[i].split(',')[0]+','+ str(int(s[i].split(',')[1]))+',
      '+str(int(s[i].split(',')[2])+1) # On incrémente le nombre de parties
       jouées
290           self.__joueur=s[i] # On met à jour le joueur
```

```
291
292         with open('historique.txt','w') as f: # On ouvre le fichier
        historique en écriture          f.write('\n'.join(s))
293         # On écrit la liste dans le fichier
294
295 fen = Menu() # On ouvre la fenêtre du menu
296 fen.mainloop()
```

Listing 1 – Code du jeu du pendu

6 Résultats et Analyse

En examinant les résultats obtenus, on peut constater que le menu s'affiche conformément aux spécifications (3).

Au démarrage, le joueur doit saisir son nom pour commencer à jouer.Lorsqu'on clique sur "Jouer", le menu change pour tenter de deviner le mot, où on a des lettres déjà découvertes et des astérisques représentant les lettres manquantes (4). Toutes les fonctionnalités du programme sont opérationnelles et répondent aux attentes. Le fichier texte peut être facilement modifié pour ajouter ou supprimer des mots.

Les boutons "Quitter" et "Nouvelle partie" fonctionnent correctement (5). Lorsque le mot est entièrement découvert, la réponse est affichée avec le message "Victory" (7) et, sinon, la réponse est affichée avec le message "Defeat ! Le mot était : ****" (8).

La visualisation de l'historique est également réussie (6). En outre, la mise à jour du fichier historique.txt est effectuée comme prévu. Si l'on procède à un changement de joueur, la fenêtre "Historique" ainsi que le fichier contenant l'historique de toutes les parties restent inchangés et, après le jeu du nouveau joueur, il est additionné des nouveaux résultats, plusieurs parties, on a constaté que les scores et les fichiers sont bien mis à jour. Enfin, nous effectuons la suppression de l'historique avec succès. Il est aussi bien d'additionner, qu'il est possible de changer les couleurs de la Zone d'affichage, du clavier, du 'Back Ground', et, aussi, annuler un choix (9).

FIGURE 3 – Menu Initial du Jeu du Pendu

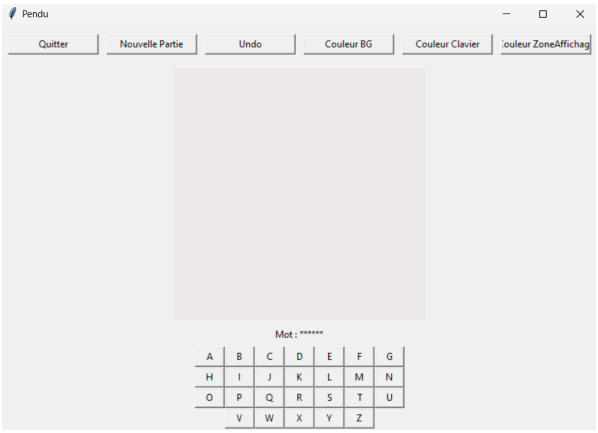

FIGURE 4 – Construction du jeu

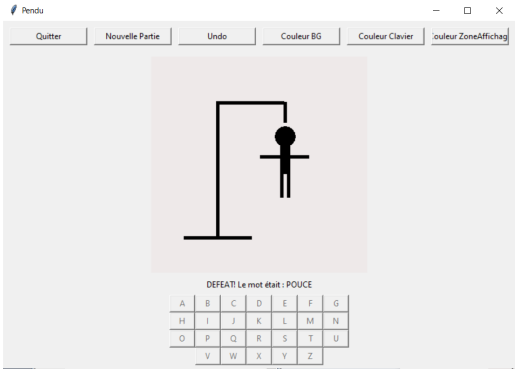

FIGURE 5 – Fenêtre du Jeu et fonctionnement des boutons

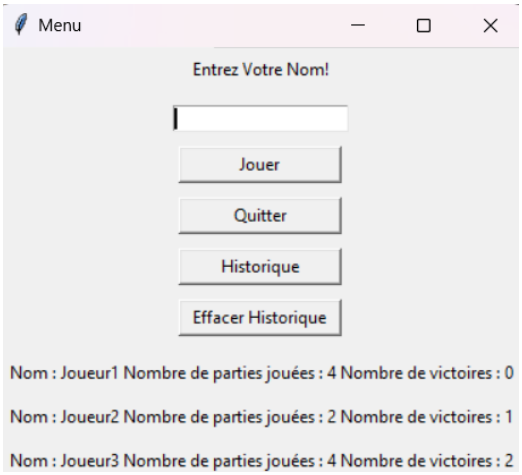

FIGURE 6 – Historique du Jeu

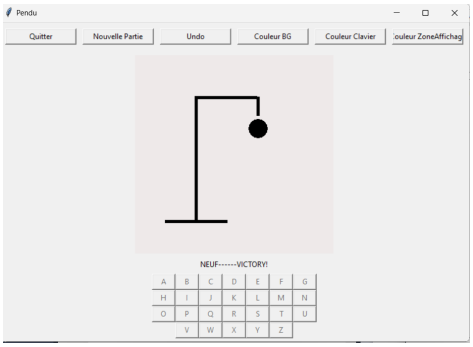

FIGURE 7 – Fenêtre du Jeu pour une Victoire

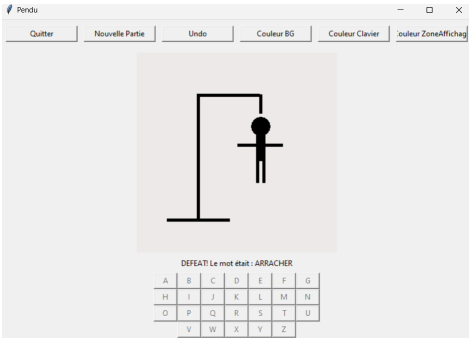

FIGURE 8 – Fenêtre du Jeu pour une Défaite

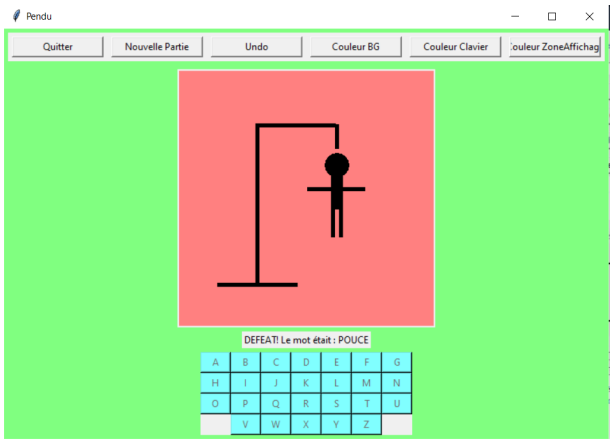

FIGURE 9 – Changement de couleurs

De cette manière, il est clair que le modèle implémenté réussit les tests proposés et, par conséquent, représente bien ce qui était demandé.

7 Conclusion

En conclusion de ce travail, on peut remarquer que le code créé à partir du Cahier des Charges a obtenu une réponse qui l'a satisfait. Ainsi, il est démontré que la procédure adoptée dans ces codes est bien adaptée aux situations requises. En plus, l'utilisation d'un Cahier des Charges clair et d'une illustration bien faite du Diagramme de Classe (UML) permet une vision plus complète du problème, c'est-à-dire, quels sont les attributs de chaque classe et les méthodes nécessaires à suivre, ce qui facilite l'approche et la résolution de l'exercice.

Pour finir, on note que les codes ont été obtenus de manière claire et concise, démontrant le succès de la méthode utilisée.

8 Annexe

8.1 Code source : ("*formes.py*")

Pour gérer le jeu du pendu, on a employé les formes développées lors des travaux dirigés 1 et 3. Le code correspondant est présenté ci-dessous.

```
from math import pi # pour le calcul du diametre et de la surface du cercle
from tkinter import *
from tkinter.messagebox import * # boite de dialogue

couleurs = ['blue','red','green','yellow','white','black']

# ***** classe Forme *****
class Forme:
    def __init__(self,x,y,c): # Constructeur
        self.__xc = x # attribut prive : le nom est precede de "__"
        self.__yc = y
        self.__couleur = c

    def get_centre(self): # retourne les coordonnes du centre
        return self.__xc,self.__yc

    def set_centre(self,x,y): # change les coordonnes du centre
        self.__xc = x
        self.__yc = y

    def get_couleur(self): # retourne la couleur choisie
        return self.__couleur

    def set_couleur(self,c): # change la couleur
        self.__couleur = c

    def deplacement(self,dx,dy): # ajoute dx et dy aux coordonnes du centre
        self.__xc += dx
        self.__yc += dy

    def affiche(self,can): # affiche la figure sur la zone graphique 'can'
        pass
```

```
33
34      def selection(self,x,y): # verifie si x et y sont sur la forme
35          pass
36
37  # ***** classe Rectangle *****
38  class Rectangle(Forme):
39      def __init__(self,x,y,l,h,c): # Constructeur
40          Forme.__init__(self,x,y,c) # Le constructeur de la classe
        derivee doit faire appel a celui de la classe de base
41          self.__l = l  # largeur du rectangle
42          self.__h = h # hauteur du rectangle
43
44      def get_dim(self):
45          return self.__l,self.__h # retourne les dimensions du rectangle
46
47      def set_dim(self,l,h): # change les dimensions du rectangle
48          self.__l = l
49          self.__h = h
50
51      def perimetre(self): # retourne le perimetre du rectangle
52          return 2*(self.__l+self.__h)
53
54      def surface(self): # retourne la surface du rectangle
55          return self.__l*self.__h
56
57      def __str__(self): # pour la partie optionnelle (Dessin). Permet d'
        afficher un objet avec la fonction print
58          return 'Rectangle - centre : {} | dimensions : {} | couleur : {}
         | perimetre : {} | surface : {}'.format(self.get_centre(), self.
        get_dim(), self.get_couleur(), self.perimetre(),self.surface())
59
60      def affiche(self,can):          # affiche le rectangle sur la zone
        graphique 'can'
61          x,y = self.get_centre()
62          can.create_rectangle(x-self.__l//2, y-self.__h//2, x+self.__l
        //2, y+self.__h//2, outline=self.get_couleur(), fill=self.get_couleur
        ())
63
64      def selection(self,x,y): # verifie si x et y sont sur le rectangle
65          xc,yc = self.get_centre()
66          return (x >= xc-self.__l//2) and (y>= yc-self.__h//2) and (x <=
        xc+self.__l//2) and (y <= yc+self.__h//2)
67
68  # ***** classe Carre *****
69  class Carre(Rectangle):
70      def __init__(self,x,y,l,c): # Constructeur
71          Rectangle.__init__(self,x,y,l,l,c) # Constructeur de la classe
        de base
72
73      def get_dim(self): # retourne le cote du carre
74          l,l = Rectangle.get_dim(self)
75          return l
76
77      def set_dim(self,l): # change le cote du carre
78          Rectangle.set_dim(self,l,l)
79  # Remarque: il est inutile de surcharger perimetre et surface de
        Rectangle
```

```
80 # qui fonctionnent grace a l'initialisation h=1 !
81
82     def __str__(self): # pour la partie optionnelle (Dessin). Permet d'
   afficher un objet avec la fonction print
83         return 'Carre - centre : {} | dimensions : {} | couleur : {} |
   perimetre : {} | surface : {}'.format(self.get_centre(), self.get_dim
   (), self.get_couleur(), self.perimetre(),self.surface())
84
85
86 # ***** classe Cercle *****
87 class Cercle(Forme):
88     def __init__(self,x,y,d,c): # constructeur
89         Forme.__init__(self,x,y,c) # constructeur de la classe de base
90         self.__d = d
91
92     def get_dim(self): # retourne le diametre du cercle
93         return self.__d
94
95     def set_dim(self,d): # change le diametre du cercle
96         self.__d = d
97
98     def perimetre(self): # retourne le perimetre du cercle
99         return pi*self.__d
100
101     def surface(self): # retourne la surface du cercle
102         return pi*self.__d**2/4
103
104     def __str__(self): # pour la partie optionnelle (Dessin). Permet d'
   afficher un objet avec la fonction print
105         return 'Cercle - centre : {} | dimensions : {} | couleur : {} |
   perimetre : {} | surface : {}'.format(self.get_centre(), self.get_dim
   (), self.get_couleur(), self.perimetre(),self.surface())
106
107     def affiche(self,can): # affiche le cercle sur la zone graphique '
   can'
108         x,y = self.get_centre()
109         can.create_oval(x-self.__d//2, y-self.__d//2, x+self.__d//2, y+
   self.__d//2, outline=self.get_couleur(), fill=self.get_couleur())
110
111     def selection(self,x,y): # verifie si x et y sont sur le cercle
112         xc,yc = self.get_centre()
113         return (x >= xc-self.__d//2) and (y>= yc-self.__d//2) and (x <=
   xc+self.__d//2) and (y <= yc+self.__d//2)
114
115 # ***** classe Dessin *****
116 class Dessin:
117     def __init__(self):
118         self.__formes = [] # liste de formes
119
120     def add_forme(self,f): # ajoute la forme f dans la liste formes
121         self.__formes.append(f)
122
123     def del_forme(self,position): # supprime la forme d'apres sa
   position dans la liste
124         del(self.__formes[position])
125
126     def print_formes(self):  # affiche toutes les formes dans la liste
```

```
127         print('--- Dessin ----')
128         for f in self.__formes:
129             print(f)
130 
131     def affiche_formes(self,can,nb): # affiche les nb premiers formes
        dans la liste sur la zone graphique 'can' (nb maximum = nombre d'
        elements de la liste)
132         i=0
133         while (i<len(self.__formes) and i<nb):
134             self.__formes[i].affiche(can)
135             i+=1
136 
137     def selection_formes(self,x,y): # montre la forme selectionnee, s'il
         y en a
138         for f in self.__formes:
139             if f.selection(x,y) :
140                 showinfo('Forme sélectionnée',f.__str__())
141                 break
142 
143     def size(self): # retourne le nombre de formes dans la liste
144         return len(self.__formes)
```

Listing 2 – Code utilisé pour créer les formes du Jeu du Pendu