École Centrale Lyon

UE PRO
Conception Programmation Objet
Rapport

BE5 - Le Jeu du Pendu

Élèves :
Nathan KÜHR
Rémy GASMI

Enseignant :
Daniel MULLER

28 mars 2022

Table des matières

1 Introduction

L'objectif de ce BE est d'apprendre les bases de la programmation d'une interface graphique à l'aide de la librairie "tkinter", en mettant en place un jeu connu sous le nom "Jeu du Pendu". Pour rappel, le jeu consiste à trouver un mot dont le joueur connaît seulement le nombre de lettres au début. À chaque étape, le joueur doit deviner une lettre. Au cas où la lettre serait contenue dans le mot, alors sa position dans le mot est affichée. Dans le cas contraire, un coup de pinceau, élément du pendu, est ajouté. Le joueur a gagné s'il parvient à trouver toutes les lettres avant que le bonhomme dessiné soit pendu. Nous avons choisi un nombre de 10 coups manqués.

2 Structure de l'Application

2.1 La Fenêtre d'Acceuil

Afin d'obtenir un bel affichage, nous avons pris la décision "d'habiller" le code avec une première fenêtre :

FIGURE 1 – Fenêtre de départ

Voici le code relatif à ce premier affichage :

```
class HomePage(Tk):
    def __init__(self,color):
        Tk.__init__(self)
        #Image de l'écran d'acceuil
        self.__color = color
        self.configure(bg = self.__color)
        self.__im = Image.open('pendujeuv.png')
        self.__logo = ImageTk.PhotoImage(self.__im, master=self)
        # le logo doit être un attribut ou bien l'image est écrasée par le garbage collector et l'on obitent un canva vide
        self.__dessin = Canvas(self,width = self.__im.size[0], height = self.__im.size[1], bg = self.__color)
        self.__logo1 = self.__dessin.create_image(0,0,anchor = NW,  image = self.__logo)
        self.__dessin.pack()
        self.title('Acceuil')
        #Pour que les fenêtres apparaissent au milieu de l'écran de l'utilisateur peu importe
        #son écran
        self.longueur_ecran = self.winfo_screenwidth()
        self.largeur_ecran= self.winfo_screenheight()
        self.geometry('250x250' +"+"+ str(int((self.longueur_ecran-500)/2)) +"+"+ str(int((self.largeur_ecran-500)/2)))
        self.minsize(width=100, height=100)
        self.resizable(False,False)
        self.iconbitmap('jeu.ico')
        #bouton play
        self.__Frame = Frame(self, bg = self.__color)
        self.__Frame.pack()
        buttonPlay = Button(self.__Frame, text = 'Play', font = ('Courier',10), command = self.Play, bg = self.__color)
        buttonPlay.pack(padx = 10, pady = (10,0))

    def Play(self):
        self.destroy()
        registration = FenPrincipale(self.__color)
        registration.mainloop()
```

FIGURE 2 – Code de la fenêtre de départ

On utilise la commande d'import : "from PIL import Image, ImageTk" afin de pouvoir afficher cette image de pendu.

Comme l'apparition de la fenêtre dépend de la taille de l'écran utilisé, nous avons utilisé les commandes "winfoscreenheight()" et "winfoscreenwidth()" afin de récupérer la taille de l'écran de l'utilisateur et de pouvoir centrer l'apparition de la fenêtre à chaque lancement du code.

Nous avons voulu explorer toutes les possibilités qu'offre l'affichage de Tkinter notamment avec les commandes "resizable" et "iconbitmap".

Resizable nous permet ici de forcer la géométrie et d'empêcher l'utilisateur d'agrandir ou de réduire cette première fenêtre.

Iconbitmap permet de changer le petit icon plume que Tkinter affiche par défaut en haut à gauche de la fenêtre et de mettre une autre image au format ico.

```
if __name__ == "__main__":
    fen = HomePage("white")
    fen.mainloop()
```

FIGURE 3 – Lancement du Code

Enfin, on implémente le bouton "Play". Ce dernier détruit alors la fenetre d'acceuil et construit la fenêtre de jeu à l'aide de la Class "FenetrePrincipale".

2.2 La Fenêtre Principale

La fenêtre principale correspond donc à la fenêtre où le jeu se déroulera. Voici notre fenêtre :

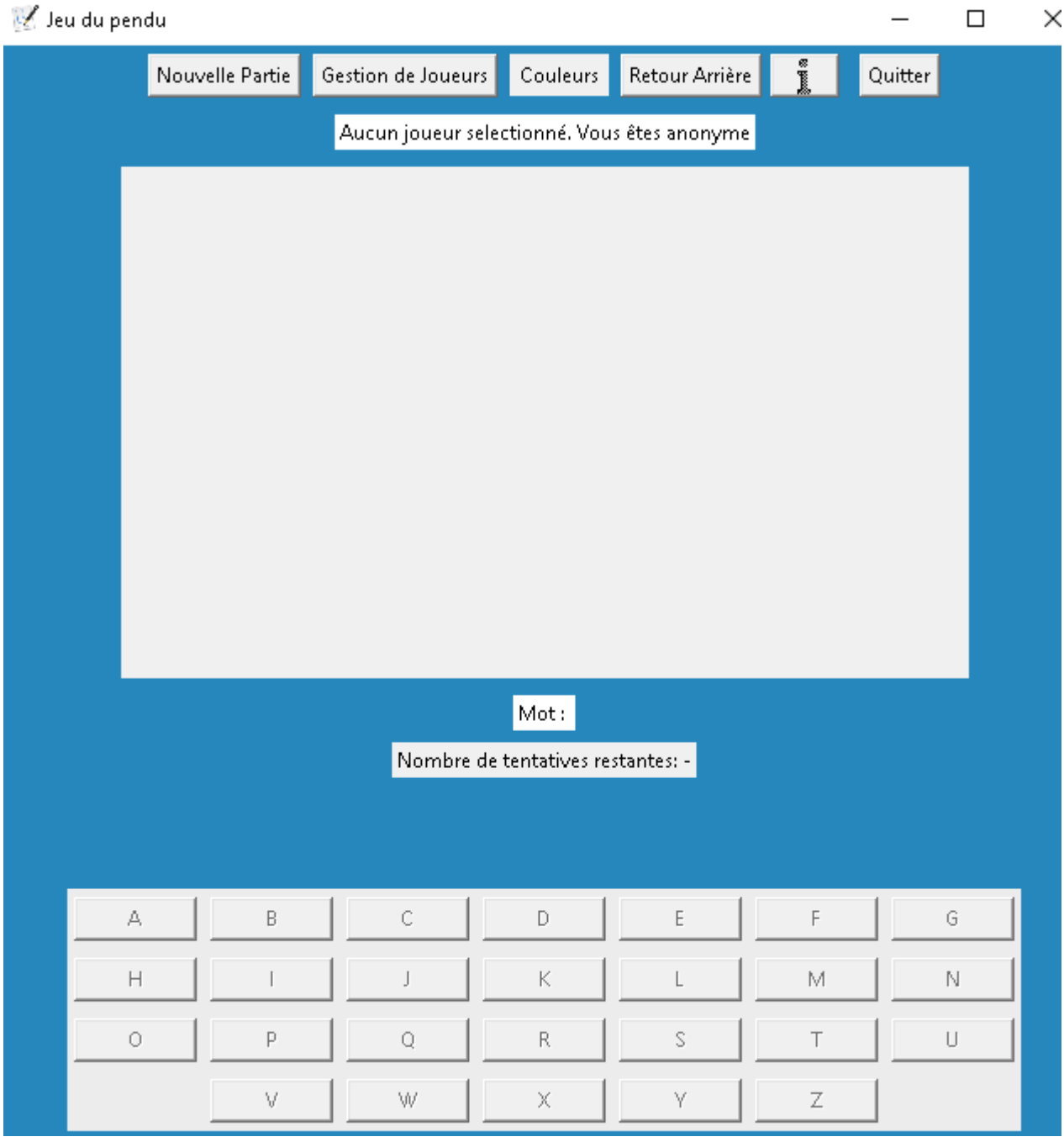

FIGURE 4 – Fenêtre Principale

Son architecture est la suivante :
- L'ensemble des boutons et donc des options sont placés dans une barre d'outils qui est implémentée par un *Frame*.
- Un Label pour dire à l'utilisateur s'il joue en anonyme ou s'il s'est identifié.
- Une Zone d'affichage du pendu.
- Un Label pour voir quel mot il faut deviner.
- Un Label pour compter le nombre de tentatives restantes.
- Un clavier avec l'ensemble des lettres.

2.2.1 Les Boutons

Concernant les Boutons, ils sont construits et placés comme vu en TD avec la méthode pack :
Le bouton "Nouvelle Partie" permet donc de lancer le jeu avec la fonction implémenter en cours.
Le bouton "Gestion de Joueurs" permet de se créer un compte (et de jouer avec) afin de sauvegarder nos performances tout au long du jeu.
Le bouton "Couleur" pour changer les couleurs de la zone d'affichage du pendu ainsi que de la pourtre et de la silhouette pendue.
Le bouton "Retour Arrière" pour pouvoir revenir en arrière en cas d'échec.
Le bouton avec le symbole Info ouvre une autre fenêtre pour donner plus d'informations sur la fenêtre de jeu. Elle permet également d'afficher un message au joueur.
Le bouton "Quitter" pour sortir du jeu.

3 Améliorations du Jeu

Pour améliorer la version du Jeu du Pendu du TD, nous avons réorganisé profondément la structure de notre application. Dans la figure suivant, est proposé un schéma UML qui modélise cette structure.

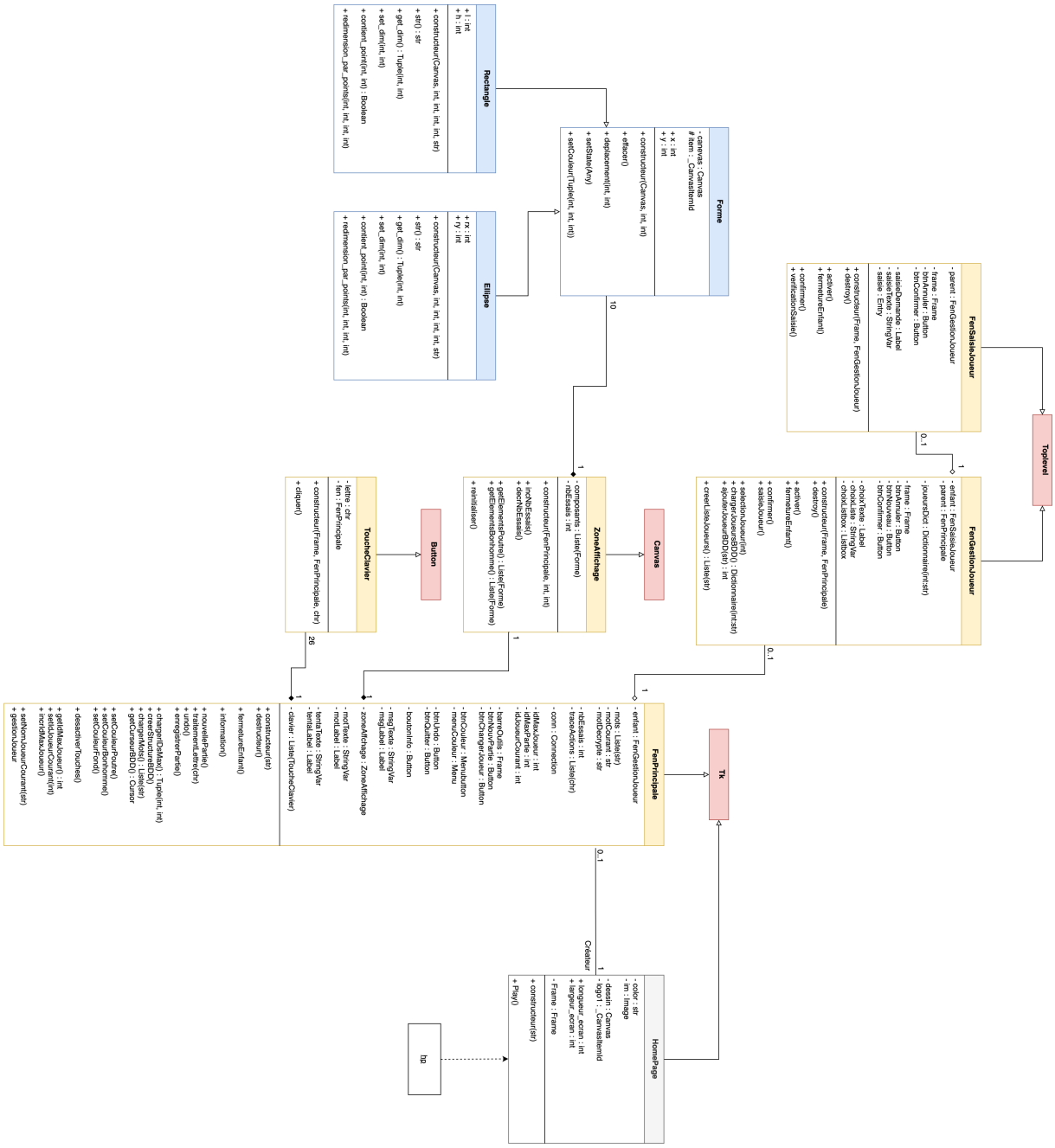

FIGURE 5 – Schéma UML de notre application

3.1 La Fenêtre de Gestion des Joueurs

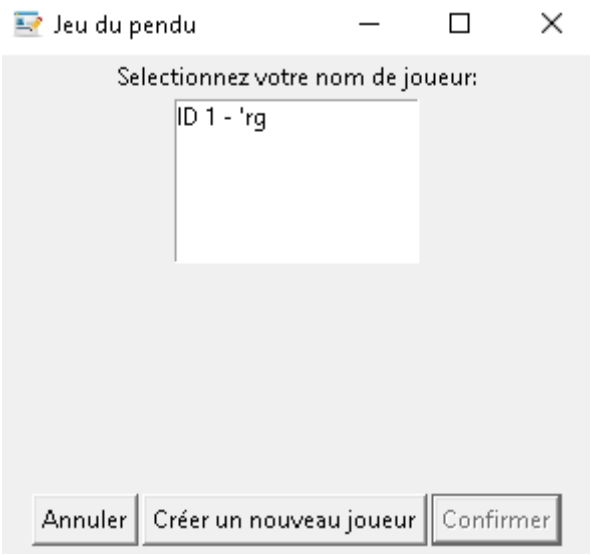

FIGURE 6 – Fenêtre de création et de chargement de profil

L'objectif de cette fenêtre est de pouvoir créer un compte et de sauvegarder ses performances. Pour cela, on a implémenté une première méthode, nommée $ajouterJoueurBDD()$, qui permet à l'utilisateur de créer un compte qui sera directement relié à un numéro ID afin de ne pas mélanger les comptes en cas de similarité.

```
def ajouterJoueurBDD(self, pseudo):
    # Préparer la demande sql
    sql = 'INSERT INTO Joueurs (idjoueur,pseudo) VALUES (?,?)'
    # Objet qui permet la requête sql
    curseur = self.__parent.getCurseurBDD()

    # L'ID que le joueur obtiendra
    id = self.__parent.getIdMaxJoueur() + 1
    try:
        # Executer la demande
        curseur.execute(sql, (id, pseudo))
        # Si réussi, incrémenter idMaxJoueur
        self.__parent.incrIdMaxJoueur()
        # Et ajouter le joueur à la bdd
        self.__joueursDict[id] = pseudo
    except Exception as err:
        print("Une exception a été relévé: '" + str(err) + "', type d'exception: '" + type(err).__name__)
        return None
    # Retourner l'ID du joueur
    return id
```

FIGURE 7 – Fonction AjouterJoueurBdd

Cette méthode se base sur la requête SQL $'INSERT\ INTO\ Joueurs(idjoueur, pseudo)\ VALUES(?$, nous donnant accès.

On a également implémenté une deuxième méthode permettant de charger un compte précédemment créer, nommée $chargerJoueursBDD$.

Cette méthode se base sur la requête SQL "$SELECTidjoueur, pseudoFROMJoueurs$".

```python
def chargerJoueursBDD(self):
    # Préparer la demande sql
    sql = "SELECT idjoueur, pseudo FROM Joueurs"
    # Objet qui permet la requête sql
    curseur = self.__parent.getCurseurBDD()
    # --- Essai de récupérer la liste de noms ---
    try:
        curseur.execute(sql)
    except sqlite3.OperationalError as err:
        # Probablement le tableau n'existe pas encore
        # Dans ce cas, (ré)créer la structure de la bdd
        if "no such table" in str(err):
            self.__parent.creerStructureBDD()
        else:
            print("Une exception a été relévé: '" + str(err) + "', type d'exception: '" + type(err).__name__)
            return None
    except Exception as err:
        print("Une exception a été relévé: '" + str(err) + "', type d'exception: '" + type(err).__name__)
        return None
    # --- Succès ---
    # Stocker les pseudo par les ID uniques
    joueursDict = {}

    for element in curseur.fetchall():

        # ID = element[0], pseudo = element[1]
        joueursDict[element[0]] = element[1]

    return joueursDict
```

FIGURE 8 – Fonction ChargerJoueurBdd

3.2 Le Bouton Undo

La fonction "*retourenarrire*" est codée à l'aide de la fonction suivante :

Le principe réside dans la création d'une pile. Lorsque le joueur clique sur une lettre, elle est stockée dans une pile, implémentée à l'aide de la méthode *append*() d'une liste. La fonction undo permet ensuite de récupérer la plus récente lettre en l'enlevant de la pile (*pop*()). En-tout-cas, la touche correspondante à la lettre sera dégrisé. Puis, si la lettre était dans le mot, elle est remplacée par des astérisques, et si elle ne l'était pas, on met à la fois à jour l'affichage du mot et des tentatives ainsi que le dessin du pendu.

3.3 Le Changement des Couleurs

La touche changement de couleur permet au joueur de changer les couleurs dans la zone d'affichage, c'est-à-dire du fond de la zone, de la poutre ainsi que du bonhomme. Si le joueur clique sur la touche, alors une sélection de ces trois options sera affichée, implémentée par les éléments graphiques *Menu* et *Menubutton*. Puis, à l'aide de l'outil *colorchooser*, le joueur pourra sélectionner la couleur souhaitée. Afin d'implémenter cette fonctionnalité, nous avons ajouté une fonction appelée *setCouleur*() dans la classe *Forme*. Ci-bas, est affichée une partie de l'implémentation dans notre code.

```
def undo(self):
    # Si la trace est vide, abandonner
    if (len(self.__traceActions) == 0):
        return
    # Trouver la lettre qui a été cliquée
    lettre = self.__traceActions.pop()
    # Mettre à jour le mot décrypté
    succes = False
    # Parcourir le mot, lettre par lettre
    for (i, c) in enumerate(self.__motCourant):
        if c == lettre:
            # Changer cette fois-ci la lettre pour l'astérisque
            self.__motDecrypte = self.__motDecrypte[:i] + "*" + self.__motDecrypte[i+1:]
            # Le mot contient la lettre rentrée
            succes = True
    # Mettre à jour l'affichage du mot décrypté
    self.__motTexte.set("Mot : " + self.__motDecrypte)
    self.__motLabel.update() # Bug macOs nécessite un update() manuel
    # Dégriser la touche
    self.__touches[ord(lettre) - ord('A')].config(state=NORMAL)
    # Si c'était une bonne tentative, il ne reste plus rien à faire
    if succes == True:
        return
    #
    # Dans le cas contraire, mettre à jour l'affichage du pendu ainsi que l'affichage des tentatives restantes
    # Cacher l'élement du pendu
    self.__zoneAffichage.decrNbEssais()
    # Décrémenter le nombre de mauvais esssais
    self.__nbEssais -= 1
    # Mettre à jour l'affichage de tentatives restantes
    self.__tentaTexte.set("Nombre de tentatives restantes: " + str(10 - self.__nbEssais))
    self.__tentaLabel.update() # Bug macOs nécessite un update() manuel
```

FIGURE 9 – Fonction retour en arrière

```
def setCouleurPoutre(self):

    # Demander une couleur au joueur
    c = colorchooser.askcolor(title='Choisissez votre couleur')[1]

    # Changer la couleur de chaque élement
    for e in self.__zoneAffichage.getElementsPoutre():
        e.setCouleur(c)


def setCouleurBonhomme(self):

    # Demander une couleur au joueur
    c = colorchooser.askcolor(title='Choisissez votre couleur')[1]

    # Changer la couleur de chaque élement
    for e in self.__zoneAffichage.getElementsBonhomme():
        e.setCouleur(c)


def setCouleurFond(self):

    # Demander une couleur au joueur
    c = colorchooser.askcolor(title='Choisissez votre couleur')[1]

    # Changer la couleur de la zone d'affichage
    self.__zoneAffichage.config(bg = c)
```

FIGURE 10 – Fonction pour changer la couleur du dessin et de la zone d'affichage

4 Conclusion

Coder le jeu du pendu nous a permis de mettre en application de manière très concrète l'ensemble des TD précédents de cette action de formation.

Nous avons utilisé notre code "Formes" pour dessiner le pendu, utiliser tkinter pour coder un environnement de jeu à la fois visuel et optimisé.

Les possibilités sont quasiment infinies. Nous aurions pu en effet prévoir une zone permettant de taper "un code de triche" qui aurait activé le bouton "Retour en arrière".

Nous aurions pu également prévoir un mode permettant à l'utilisateur de choisir le mot à faire deviner à un second joueur, etc.