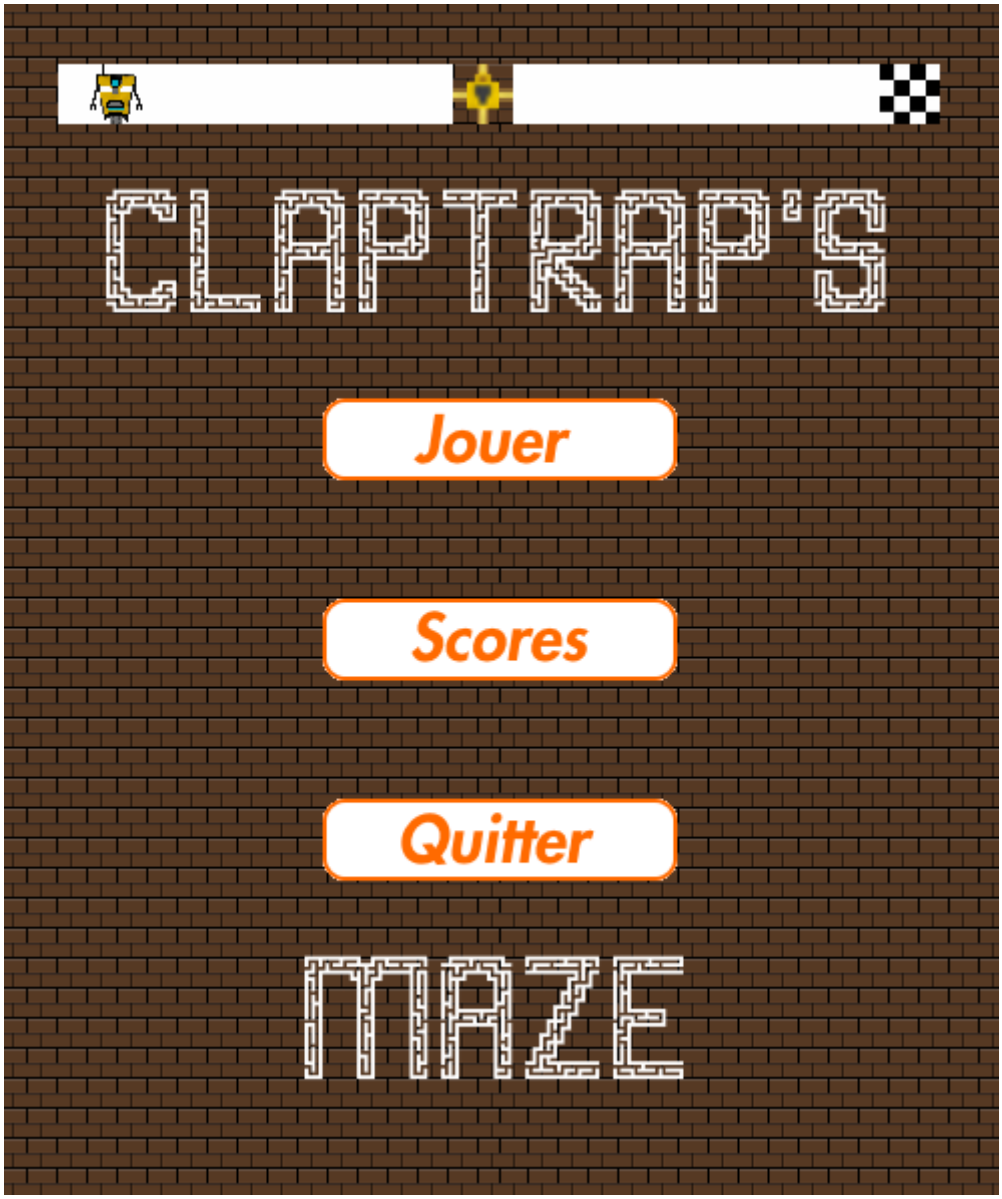
Damien CLAUZON Julien FLEURY

Dossier projet de Damien CLAUZON

CLAPTRAP'S MAZE



Sommaire

I – Présentation du projet

Le projet que nous avons réalisé au cours de l'année est un programme de labyrinthe.

Nous avons choisi de programmer un labyrinthe pour plusieurs raisons :
• Tout d'abord sur un critère personnel, nous voulions créer quelque chose dont le résultat nous plairait.
• Ensuite, l'idée d'un labyrinthe est très générale, il est possible de personnaliser tous les aspects d'un labyrinthe, à l'inverse d'un jeu de société dont les règles sont déjà pré-établies.
• Enfin, un labyrinthe peut être simple ou très complexe. Avec nos connaissances lors de notre choix, nous n'imaginions pas pouvoir réaliser des parties extrêmement complexes. Mais une fois que nous aurons plus de notions en programmation, il sera possible de faire évoluer le programme.

Les objectifs pour la réalisation de ce projet sont les suivants :
• respecter l'idée de labyrinthe, c'est-à-dire que des murs constituent les obstacles pour atteindre la ligne d'arrivée
• un ajout par rapport à un labyrinthe basique (éléments comme bonus, malus et clef)
• un système de classement, pour pouvoir comparer son temps avec celui d'un autre joueur, ou encore avec son propre temps

Pour réaliser le projet, nous devions programmer en Python avec l'aide de la bibliothèque tkinter.

II – Développement du projet

Structure globale du projet :
Les deux principales articulations du projet sont :
• le labyrinthe
`def SelectionNiveau():` Fenêtre de sélection du niveau (parmi 3)
`def Tutoriel():` Fenêtre « Aide » pour expliquer les éléments du jeu
`def Facile():` Fonction initialisant le niveau du labyrinthe (ici, le niveau facile)
`def Labyrinthe():` Fonction qui génère le labyrinthe et commence la partie
`def move(event):` Fonction permettant de se déplacer

• le tableau des scores
`def Scoreboard():` Permet de choisir entre les scores des trois niveaux
`def OngletFacile():` Ouvre la fenêtre correspondant au niveau choisi (ici, facile)
`def LireScore(niv):` Affiche les scores sous le format « premier / deuxième / troisième / dernier score enregistré »
`def TriScore(niv):` Fonction qui trie les scores pour en sortir les 3 premiers
`def TriDernierScore(niv):` Fonction qui trie le dernier score de chaque niveau

Planning
-Un brainstorming des fonctionnalités nécessaires pour commencer à coder, pour avoir une idée claire de ce qu'il faut coder.

-Chercher comment le noyau du jeu va fonctionner.

- Le codage du noyau a pris environ 1/4 du temps, et nous avons créé les schémas des labyrinthes pendant cette période.

- Une fois le noyau principal fini, nous avons introduit les éléments supplémentaires au programme : bonus, piège, clef, porte, chronomètre. Nous avons aussi créé les images des différents éléments.

- Un second brainstorming afin d'avoir les exigences sur le tableau des scores

- Codage du tableau des scores avec un système de fichier "score par niveau" et de dernier score. Cette partie a pris le plus de temps, car elle fait appel à de nombreuses fonctions.

- Interface graphique du menu principal / sélection des niveaux / tableau des scores avec le menu interactif, et ajout d'une "aide" dans la sélection du niveau.

Répartition des tâches :

Julien s'est occupé de faire le design du labyrinthe et de réfléchir aux divers éléments du programme, comme les éléments supplémentaires ou encore le fonctionnement du tableau des scores.
Pour ma part, je me suis occupé du codage du projet, et de la réalisation des images.

Détail des fonctions les plus importantes :

```
def move(event):
    global x,y,BonusDestroy
    if event.keysym=='Up':

        collisionU=C.find_overlapping(x+14,y-16,x-14,y-44)
        bloc0=C.gettags(collisionU[0])
        if "mur" not in bloc0 and "porte" not in bloc0:
            y=y-30
            C.move(perso,0,-30)
            if "goal" in bloc0:
                Lab.destroy()
                Lab.quit()
                goal()
            if "bonus" in bloc0[0] :
                BonusDestroy=bloc0[0]
                bonus()
            if "piege" in bloc0:
                piege()
            if "clef" in bloc0:
                clef()
```

La fonction *move* est sûrement la fonction la plus importante pour le noyau du projet. Elle gère les déplacements du joueur, avec un système de détection des objets.

Ce système fonctionne de la manière suivante :
• la fonction *find_overlapping* permet de créer un rectangle qui va trouver les ID des objets à l'intérieur (lors d'un déplacement vers le haut, ce rectangle sera en haut du joueur)
• elle trouve le tag (mot attribué à un type d'objet, comme "mur" ou "clef") du premier objet trouvé (puisqu'il ne peut y avoir qu'un seul à la fois)
• elle exécute une action selon le tag trouvé. Si le tag n'est ni un mur, ni la porte, alors le personnage se déplace dans le sens de la touche. "x" et "y" permettent d'avoir les coordonnées du joueur en permanence.

```
def Labyrinthe():
[...]
for y in range(0,hauteur):
    for x in range(0,largeur):
        goal="+"+str(x)+"+"
        bonus="/"+str(x)+"/"
        piege="-"+str(x)+"-"
        vide=","+str(x)+","
        clef="!"+str(x)+"!"
        porte="  "+str(x)+"  "
        # Cree un bloc d'arrivee lorsque "+x+" est ecrit
        if goal in ligne[y]:
            C.create_image(15+30*x,15+30*y,image=Arrive,tags="goal")
        # Cree un bloc de bonus lorsque "/x/" est ecrit
        elif bonus in ligne[y]:
            BonusNombre+=1
            C.create_image(15+30*x,15+30*y,image=Bonus,tags="bonus%s"%BonusNombre)
        # Cree un bloc de piege lorsque "-x-" est ecrit
        elif piege in ligne[y]:
            C.create_image(15+30*x,15+30*y,image=Bonus,tags="piege")
        # Cree une clef lorsque "!x!" est ecrit
        elif clef in ligne[y]:
            C.create_image(15+30*x,15+30*y,image=Clef,tags="clef")
    # Cree une porte lorsque "  x  " est ecrit
        elif porte in ligne[y]:
            C.create_image(15+30*x,15+30*y,image=Porte,tags="porte")
    # Cree un carre lorsque ",x," n'est pas ecrit
        elif vide not in ligne[y]:
            C.create_image(15+30*x,15+30*y,image=Mur,tags="mur")
```

Les deux boucles *for* sont également indispensables pour la génération du labyrinthe.
Cette fonction permet de générer un labyrinthe à partir d'un fichier texte.

```
#1# (premiere ligne)
#2# ,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
#3# ,11,17,20,24,25,
#4# ,11,13,14,15,17,20,21,25,26,-13-
#5# ,1,2,3,4,5,6,7,11,15,17,21,22,+1+
#6# ,7,11,12,15,17,18,19,22,23,/7/23/
#7# ,1,2,3,4,5,6,7,12,13,15,17,19,23,24,/4/
#8# ,7,13,15,17,19,24,
#9# ,1,6,7,8,9,10,11,15,18,19,21,22,23,24,/2/
#10# ,1,5,6,11,12,13,14,15,16,18,20,21,24,/18/-4-
#11# ,1,16,24,§16§
#12# ,1,10,11,12,13,14,15,16,18,19,20,21,24,
#13# ,1,2,3,4,5,6,7,8,9,13,21,23,24,25,26,27,
#14# ,1,9,11,12,13,14,15,19,21,24,27,
#15# ,1,3,5,6,7,8,9,11,13,15,17,18,19,20,21,24,27,
#16# ,1,3,5,9,11,13,16,17,21,24,27,
#17# ,1,3,4,9,11,13,16,21,23,24,27,28,-23-
#18# ,1,4,5,6,7,8,9,13,14,15,16,19,20,21,24,
#19# ,1,4,11,18,19,23,24,
#20# ,1,4,6,7,8,9,11,13,14,15,16,17,18,22,23,
#21# ,1,2,4,6,11,16,17,19,21,22,
#22# ,2,4,5,6,7,8,9,10,11,12,13,14,15,16,19,21,
#23# ,2,19,21,22,23,24,25,26,
#24# ,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,26,
#25# ,3,9,14,19,20,21,22,23,24,25,26,
#26# ,3,4,5,6,9,10,11,12,14,17,22,26,/26/
#27# ,3,12,13,14,16,17,18,19,20,21,22,26,/3/-20-
#28# ,26,
#29# ,26,!26!
#30# (derniere ligne)
```

Lorsque le fichier est ouvert, les informations sont rentrées dans une liste où chaque ligne correspond à un élément.
La première boucle *for* gère le numéro de la ligne, et la seconde gère le numéro de la colonne.
Ainsi, le programme va générer ligne par ligne les éléments numérotés dans le fichier texte.
Le labyrinthe a une dimension de 900x900, et chaque bloc fait 30*30.
Ainsi il y a 30 colonnes et 30 lignes.
Exemple : la première boucle s'exécute, y=0 (1ère ligne).
Elle fait défiler x=0,x=1,x=2... en cherchant dans le fichier ce numéro sous la forme ,*nombre,*
S'il trouve un numéro correspondant à x, il va créer un bloc en fonction du format
S'il ne trouve aucun numéro correspondant à x, alors un mur est créé.

```
def TriScore(niv):
    global ListeComplete,DernierJoueur

    NomSave="scores/"+str(niv)+".txt"
    fich=open(NomSave,"r")
    ListeScore=fich.readlines()
    fich.close()

    ListeComplete=[]
    for compteur in range(0,len(ListeScore),2):
        namejoueur=ListeScore[compteur]
        timejoueur=ListeScore[compteur+1]
        if "\n" in timejoueur:
            timejoueur=timejoueur.replace("\n","")
        if "\n" in namejoueur:
            namejoueur=namejoueur.replace("\n","")

        ListeComplete.append([namejoueur,timejoueur])

    ListeComplete.sort(key=lambda colonne: colonne[1])
```

La fonction *TriScore* permet de trier la liste des scores d'un niveau. Elle est appelée deux fois dans le programme : pour afficher le tableau des scores, et pour déterminer le classement du dernier score enregistré.

D'abord, elle ouvre le fichier du score correspondant au niveau, sous cette forme :

```
Dorian
01:34
Julien
00:35
Jonathan
02:22
Maud
02:00
Claire
04:24
```

La boucle *for* permet de lire les lignes (rentrée dans une liste) deux par deux, c'est-à-dire joueur par joueur. Elle attribue le nom à la première des deux lignes, et le score à la seconde. Elle rajoute ce nom et ce score dans la "ListeComplete" contenant à la fin de la boucle *for* tous les noms et scores enregistrés.
Cette liste est ensuite triée selon le "rang 1" de la première liste, c'est-à-dire selon le temps.
Par exemple, la fichier ci-contre donne avant le tri :
[['Dorian', '01:34'], ['Julien', '00:35'], ['Jonathan', '02:22'], ['Maud', '02:00'], ['Claire', '04:24']]
Et après le tri :
[['Julien', '00:35'], ['Dorian', '01:34'], ['Maud', '02:00'], ['Jonathan', '02:22'], ['Claire', '04:24']]

Problèmes rencontrés pendant le développement du projet :

• Le brainstorming de départ a beaucoup changé avec le temps.
L'idée d'objets cachés n'avait pas de place dans le programme, nous avons préféré garder un nombre d'éléments différents assez bas. Il en est de même pour les idées de "personnages différents", car la vitesse ne peut pas être changée, et la vision est propre à chaque labyrinthe (le design de chaque niveau est fait exprès lors de la création du labyrinthe, par exemple dans le niveau difficile on peut voir la ligne d'arrivée dès le départ)

• Le système du tableau des scores a pris beaucoup plus de temps que prévu.
Nous imaginions qu'une fois la partie labyrinthe terminée, le programme serait fini. Mais faire un tableau des scores était plus compliqué que prévu, et au lieu de faire un tableau des scores exhaustifs, nous nous sommes orientés vers un "Top 3" des meilleurs scores.

III – Bilan

Le résultat de ces trois mois de programmation a dépassé certaines de nos attentes initiales. Nous imaginions plus d'éléments dans le labyrinthe, mais au final le fait qu'il y ait moins d'éléments permet de vite comprendre le fonctionnement. L'interface graphique globale du programme est également meilleure que nos attentes.

Le système de génération du labyrinthe est meilleur que celui imaginé à la base :
La première idée que l'on avait eu était d'attribuer des coordonnées pour chaque bloc existant. Sachant qu'il y a 900 blocs au total, nous avons vite abandonné cette idée. Ensuite, nous avons décidé de faire une liste contenant chaque bloc qui n'est pas un mur, et remplir le reste avec des murs. En développant cette idée, nous avons crée un fichier texte avec un formatage spécial pour les différents blocs.
Cela permet de créer ou modifier des labyrinthes très facilement.

Il y a de multiples améliorations possibles au programme actuel :
• faire en sorte que la fenêtre s'adapte à la résolution de l'écran, pour que le programme fonctionne sur des résolutions inférieures à une hauteur de 900
• optimiser les ralentissements car le grand nombre de murs consomme beaucoup de ressources, en particulier sur des ordinateurs peu puissants

Il existe de nombreux prolongements possibles à ce programme :
• augmentation du nombres de labyrinthes disponibles
• section "création de labyrinthe" où le joueur pourrait créer ses propres labyrinthes, qui seraient enregistrés sous le même format que ceux existants actuellement
• section "labyrinthe aléatoire" avec un labyrinthe aléatoire selon une hauteur et une largeur définie par le joueur
• contrôle du personnage à la souris au lieu du clavier (avec un nouveau système de collision, pour que la souris ne puisse pas dépasser un mur)

La réalisation de ce projet m'a appris beaucoup. J'ai pu me rendre compte qu'il vaut mieux diviser le plus possible le travail, et se concentrer sur une tâche à la fois. J'ai également approfondi mes connaissances en Python, et particulièrement dans la bibliothèque tkinter en faisant des recherches. Plus généralement, la réalisation du projet a ouvert ma curiosité sur la programmation.

Sources :

Documentation de certaines fonctions de la bibliothèque tkinter (canvas, window, geometry manager)

http://effbot.org/tkinterbook/

http://tkinter.fdex.eu/index.html

Images pixelisées dessinées à partir d'images trouvés sur internet

https://www.brik.co/

https://pixabay.com/

```
from tkinter import*
from time import*
from random import*

# Definition de la fenetre du menu principal
mainF=Tk()
mainF.title("Claptrap's Maze")
mainF.resizable(width=False,height=False)
mainF.iconbitmap("img/icone/claptrap.ico")


# Images a charger
ImgPremier=PhotoImage(file="img/couronne1.gif")
ImgSecond=PhotoImage(file="img/couronne2.gif")
ImgTroisieme=PhotoImage(file="img/couronne3.gif")
Bonus=PhotoImage(file="img/bonus.gif")
Mur=PhotoImage(file="img/mur.gif")
Arrive=PhotoImage(file="img/goal.gif")
Clef=PhotoImage(file="img/clef.gif")
Porte=PhotoImage(file="img/porte1.gif")
ImgChrono=PhotoImage(file="img/chrono.gif")
Claptrap=PhotoImage(file="img/claptrapORIGINAL.gif")
background_image=PhotoImage(file="img/background.gif")
background_mainmenu=PhotoImage(file="img/background_mainmenu.gif")
background_mainmenu_bonus1=PhotoImage(file="img/background_mainmenu_bonus1.gif")
background_mainmenu_bonus2=PhotoImage(file="img/background_mainmenu_bonus2.gif")
Cercle=PhotoImage(file="img/cercle.gif")
NewCercle=PhotoImage(file="img/newcercle.gif")

JouerBouton=PhotoImage(file="img/bouton/jouer.gif")
TableauBouton=PhotoImage(file="img/bouton/tableau.gif")
QuitterBouton=PhotoImage(file="img/bouton/quitter.gif")


# Mouvement du personnage :
# Modifie les coordonnees du personnage en permanence
# Verifie la touche appuyee avant d'effectuer le deplacement
# Fonction detaillee dans le dossier-projet

def move(event):
    global x,y,BonusDestroy
    if event.keysym=='Up':

        collisionU=C.find_overlapping(x+14,y-16,x-14,y-44)
        bloc0=C.gettags(collisionU[0])
        if "mur" not in bloc0 and "porte" not in bloc0:
            y=y-30
            C.move(perso,0,-30)
            if "goal" in bloc0:
                Lab.destroy()
                Lab.quit()
                goal()
            if "bonus" in bloc0[0] :
                BonusDestroy=bloc0[0]
                bonus()
            if "piege" in bloc0:
                piege()
            if "clef" in bloc0:
                clef()

    if event.keysym=='Down':

        collisionD=C.find_overlapping(x-14,y+16,x+14,y+44)
        bloc0=C.gettags(collisionD[0])
        if "mur" not in bloc0 and "porte" not in bloc0:
            C.move(perso,0,30)
            y=y+30
            if "goal" in bloc0:
                Lab.destroy()
                Lab.quit()
                goal()
            if "bonus" in bloc0[0] :
                BonusDestroy=bloc0[0]
                bonus()
            if "piege" in bloc0:
                piege()
            if "clef" in bloc0:
                clef()


    if event.keysym=='Left':

        collisionL=C.find_overlapping(x-16,y-14,x-44,y+14)
        bloc0=C.gettags(collisionL[0])
        if "mur" not in bloc0 and "porte" not in bloc0:
            C.move(perso,-30,0)
            x=x-30
            if "goal" in bloc0:
                Lab.destroy()
                Lab.quit()
                goal()
            if "bonus" in bloc0[0] :
                BonusDestroy=bloc0[0]
                bonus()
            if "piege" in bloc0:
                piege()
            if "clef" in bloc0:
                clef()

    if event.keysym=='Right':

        collisionR=C.find_overlapping(x+16,y+14,x+44,y-14)
        bloc0=C.gettags(collisionR[0])
        if "mur" not in bloc0 and "porte" not in bloc0:
            C.move(perso,30,0)
            x=x+30
```

```
            if "goal" in bloc0:
                Lab.destroy()
                Lab.quit()
                goal()
            if "bonus" in bloc0[0] :
                BonusDestroy=bloc0[0]
                bonus()
            if "piege" in bloc0:
                piege()
            if "clef" in bloc0:
                clef()
    return


# Definition du labyrinthe qui fonctionne selon deux boucles "for" en lisant dans un fichier formate specialement
# Le bonus marche differemment, celui-ci doit etre supprime un par un donc il necessite des tags differents
BonusNombre=0
def Labyrinthe():
    global perso,Pers,BonusNombre,Chrono_Arreter

    # Detruit la fenetre de selection de niveau pour eviter l'ouverture de plusieurs niveaux
    Niveau.destroy()
    Niveau.quit()

    Chrono_Arreter=0

    # Choix de la difficulte :
    # - Ouvre le fichier correspondant (du meme nom)
    PathNiveau="niveau/"+str(niveau)+".txt"
    fich=open(PathNiveau,"r")
    ligne=fich.readlines()
    fich.close()

    # Generation des carres du labyrinthe :
    # - Premiere boucle pour la hauteur "y", deuxieme pour largeur "x"

    for y in range(0,hauteur):
        for x in range(0,largeur):
            goal="+"+str(x)+"+"
            bonus="/"+str(x)+"/"
            piege="-"+str(x)+"-"
            vide=","+str(x)+","
            clef="!"+str(x)+"!"
            porte="  "+str(x)+"  "
            # Cree un bloc d'arrivee lorsque "+x+" est ecrit
            if goal in ligne[y]:
                C.create_image(15+30*x,15+30*y,image=Arrive,tags="goal")
            # Cree un bloc de bonus lorsque "/x/" est ecrit
            elif bonus in ligne[y]:
                BonusNombre+=1
                C.create_image(15+30*x,15+30*y,image=Bonus,tags="bonus%s"%BonusNombre)
            # Cree un bloc de piege lorsque "-x-" est ecrit
            elif piege in ligne[y]:
                C.create_image(15+30*x,15+30*y,image=Bonus,tags="piege")
            # Cree une clef lorsque "!x!" est ecrit
            elif clef in ligne[y]:
                C.create_image(15+30*x,15+30*y,image=Clef,tags="clef")
        # Cree une porte lorsque "  x  " est ecrit
            elif porte in ligne[y]:
                C.create_image(15+30*x,15+30*y,image=Porte,tags="porte")
        # Cree un carre lorsque ",x," n'est pas ecrit
            elif vide not in ligne[y]:
                C.create_image(15+30*x,15+30*y,image=Mur,tags="mur")
                # L'ancienne version d'un bloc etait C.create_rectangle(0+30*x,30*y,30+30*x,30+30*y,fill="...",width=0,tags="bloc")


    # Definition du personnage, il apparait toujours au meme endroit au debut
    Pers=PhotoImage(file="img/claptrap.gif")
    perso=C.create_image(45,45,image=Pers,tags="Personnage")

    # Bind pour se deplacer
    Lab.bind('<KeyPress>',move)


    # Affiche le labyrinthe, precedemment reduit
    Lab.deiconify()

    # Mise en place du chronometre
    CreateChrono()


    C.mainloop()

# Ligne d'arrivee
def goal():
    global Goal,temps,temps_var,Classement,Chrono_Arreter

    Chrono_Arreter=1
    mainF.deiconify()

    # Simule l'entree dans score pour retrouver le classement (utile pour afficher sans avoir encore enregistre dans le fichier)
    # Pour l'ecriture "1er" et "neme" verifie le classement
    CheckClassement("pour victoire")
    if int(Classement)==1:
        TexteClassement=str(Classement)+"er"
    elif int(Classement)>=1:
        TexteClassement=str(Classement)+"eme"

    # Fenetre pour enregistrer le score. Affiche le classement et le temps, demande le nom du joueur pour enregistrer
    Goal=Toplevel()
    Goal.title("Niveau termine")

    Goal.resizable(width=False,height=False)
    Goal.iconbitmap("img/icone/claptrap.ico")

    ScoreTexte="Bravo ! Vous avez fini en "+temps+". Vous etes "+TexteClassement+". Entrez votre nom : "
    TexteWin=Label(Goal,text=ScoreTexte,font="Constantia 20")
```

```
    TexteWin.pack(side="top",padx=10,pady=10)

    temps_var=StringVar()
    ChampNom=Entry(Goal,textvariable=temps_var,font="Constantia 18")
    ChampNom.pack()

    # Execute "SaveScore()" lorsque le bouton est appuye
    BoutonNom=Button(Goal,text="Enregistrer",font="Constantia 15",justify="center",overrelief="groove",activeforeground="blue",activebackgro
    BoutonNom.pack(padx=10,pady=10)

# Fonction qui sert a enregistrer un score. Elle enregistre alors le nom et le temps dans un fichier de score commun au niveau, et egalement
def SaveScore():
    global ScoreJoueur,NomJoueur
    Goal.destroy()
    Goal.quit()

    NomSave="scores/"+str(niveau)+".txt"

    # Enregistre le score dans le fichier commun au niveau
    ScoreJoueur=temps
    NomJoueur=temps_var.get()
    fich=open(NomSave,"a")
    fich.write(NomJoueur+"\n")
    fich.write(ScoreJoueur+"\n")
    fich.close()

    # Enregistre en tant que "dernier score" du niveau, avec son classement
    CheckClassement("pour enregistrer")

    if niveau=="facile":
        DerniereSave="scores/dernier-facile.txt"
        fich=open(DerniereSave,"w")
        fich.write(NomJoueur+"\n")
        fich.write(ScoreJoueur+"\n")
        fich.write(Classement+"\n")
        fich.close()
    elif niveau=="moyen":
        DerniereSave="scores/dernier-moyen.txt"
        fich=open(DerniereSave,"w")
        fich.write(NomJoueur+"\n")
        fich.write(ScoreJoueur+"\n")
        fich.write(Classement+"\n")
        fich.close()
    elif niveau=="difficile":
        DerniereSave="scores/dernier-difficile.txt"
        fich=open(DerniereSave,"w")
        fich.write(NomJoueur+"\n")
        fich.write(ScoreJoueur+"\n")
        fich.write(Classement+"\n")
        fich.close()


# Methode permettant de recuperer le classement d'un score.
# Execute deux fois : l'une pour afficher simplement le classement sans l'enregistrer, le second servant a enregistrer le classement (ce qui
def CheckClassement(string):
    global ListeComplete,Classement,ScoreJoueur
    TriScore(niveau)

    if string=="pour victoire":
        ScoreJoueur=temps
        ListeComplete.append(["",ScoreJoueur])
        ListeComplete.sort(key=lambda colonne: colonne[1])
        for k in range(len(ListeComplete)):
            if ScoreJoueur in ListeComplete[k]:
                Classement=str(k+1)
                break


    if string=="pour enregistrer":

        for k in range(len(ListeComplete)):
            if ScoreJoueur in ListeComplete[k]:
                Classement=str(k+1)
                break

# Bonus : supprime le bonus sur lequel le joueur a marche, puis choisi un bonus aleatoire augmentant temporairement la vision
def bonus():
    global perso,Pers

    # Supprimer l'item correspondant au tag du bonus "trouve" (rappel : chaque bonus a un nom different)
    DetruireBonus=C.find_withtag(BonusDestroy)
    C.delete(DetruireBonus)

    r=randint(1,100)
    if 0<=r<=10:
        RandomPerso="img/claptrap5.gif"
    elif 11<=r<=30:
        RandomPerso="img/claptrap4.gif"
    elif 31<=r<=50:
        RandomPerso="img/claptrap3.gif"
    else:
        RandomPerso="img/claptrap2.gif"

    C.delete(perso)
    Pers=PhotoImage(file=RandomPerso)
    perso=C.create_image(x,y,image=Pers,tags="Personnage")



    C.after(7500,bonus2)

# Deuxieme partie de bonus, remettant la vision normale du joueur
def bonus2():
    global perso,Pers
    C.delete(perso)
    Pers=PhotoImage(file="img/claptrap.gif")
    perso=C.create_image(x,y,image=Pers,tags="Personnage")
```

```
# Piege : teleporte le joueur au coordonnees du depart
def piege():
    global x,y
    x=45
    y=45
    C.coords(perso,45,45)
    Lab.mainloop()

# Clef : detruit la clef puis detruit la porte
def clef():

    #Detruire la clef
    clefdetruire=C.find_withtag("clef")
    C.delete(clefdetruire)

    #Detruire le mur
    portedetruire=C.find_withtag("porte")
    C.delete(portedetruire)

    #Affiche la clef en bas de l'ecran
    img_clef=Label(Lab,image=Clef,bg="black")
    img_clef.pack(anchor="s")

# Bouton pour quitter le jeu (sur la fenetre principale)
def Quit():
    mainF.destroy()


# Lorsque le joueur appuie sur le bouton "Facile" dans la selection du niveau.
# Defini les coordonnees du personnage (pour gerer les mouvements), la hauteur/largeur a 30/30 (pareil pour tous les niveaux), defini la var
def Facile():
    global niveau,hauteur,largeur,x,y
    x=45
    y=45
    hauteur=30
    largeur=30
    niveau="facile"
    Lab.title("Claptrap's Maze - Difficulte : FACILE")
    Labyrinthe()

# Meme chose pour le niveau moyen que pour le niveau facile
def Moyen():
    global niveau,hauteur,largeur,x,y
    x=45
    y=45
    hauteur=30
    largeur=30
    niveau="moyen"
    Lab.title("Claptrap's Maze - Difficulte : MOYEN")
    Labyrinthe()

# Meme chose pour le niveau difficile que pour le niveau facile
def Difficile():
    global niveau,hauteur,largeur,x,y
    x=45
    y=45
    hauteur=30
    largeur=30
    niveau="difficile"
    Lab.title("Claptrap's Maze - Difficulte : DIFFICILE")
    Labyrinthe()

# Correspond au bouton "aide" de la selection du niveau.
# Ouvre une fenetre dans laquelle est expliquee les differents elements du jeu.
def Tutoriel():

    Aide=Toplevel()
    Aide.title("Description des elements du labyrinthe")
    Aide.resizable(width=False,height=False)
    Aide.iconbitmap("img/icone/claptrap.ico")

    a="Claptrap, votre personnage : vous devez le deplacer pour finir le labyrinthe"
    LabelTuto=Label(Aide,text=a,image=Claptrap,compound="left",font="Constantia 12",padx=10,pady=10).pack()

    b="Les murs : ils servent a delimiter votre parcours. Vous ne pouvez pas les traverser."
    LabelTuto=Label(Aide,text=b,image=Mur,compound="left",font="Constantia 12",padx=10,pady=10).pack()

    c="Le bloc secret : quand vous atteignez ce bloc, un effet aura lieu sur votre parcours. A vous de le decouvrir !"
    LabelTuto=Label(Aide,text=c,image=Bonus,compound="left",font="Constantia 12",padx=10,pady=10).pack()

    d="La clef : l'element indispensable de votre periple qui vous permettra d'ouvrir..."
    LabelTuto=Label(Aide,text=d,image=Clef,compound="left",font="Constantia 12",padx=10,pady=10).pack()

    e="... la porte : elle s'ouvre une fois la clef en main. Le chemin debloque vous permettra d'atteindre..."
    LabelTuto=Label(Aide,text=e,image=Porte,compound="left",font="Constantia 12",padx=10,pady=10).pack()

    f="... la ligne d'arrivee : une fois atteinte, la partie est gagnee."
    LabelTuto=Label(Aide,text=f,image=Arrive,compound="left",font="Constantia 12",padx=10,pady=10).pack()

    g="Le chronometre : votre temps est compte pour atteindre la ligne d'arrivee ! Tentez de battre votre meilleur score lors de futures pa
    LabelTuto=Label(Aide,text=g,image=ImgChrono,compound="left",font="Constantia 12",padx=10,pady=10).pack()


# Fenetre de selection du niveau
# D'abord, ferme le labyrinthe ou la fenetre de selectiondu niveau si ils existaient deja
def SelectionNiveau():
    global Niveau

    try:
        Lab.destroy()
        Lab.quit()
        Niveau.destroy()
        Niveau.quit()
    except:
        False
```

```
    Niveau=Toplevel()

    Niveau.title("Selection du niveau")
    Niveau.resizable(width=False,height=False)
    Niveau.iconbitmap("img/icone/claptrap.ico")

    canvasniveau=Canvas(Niveau)
    canvasniveau.grid(row=0,column=0,rowspan=2,columnspan=3)
    canvasniveau.create_image(0,0,image=background_image,anchor="nw")

    Bfacile=Button(canvasniveau,text="Facile",font="Constantia 15",justify="center",overrelief="groove",activeforeground="blue",activebackgr
    Bfacile.grid(row=0,column=0,padx=20,pady=30)

    Bmoyen=Button(canvasniveau,text="Moyen",font="Constantia 15",justify="center",overrelief="groove",activeforeground="blue",activebackgrou
    Bmoyen.grid(row=0,column=1)

    Bdifficile=Button(canvasniveau,text="Difficile",font="Constantia 15",justify="center",overrelief="groove",activeforeground="blue",active
    Bdifficile.grid(row=0,column=2,padx=20)

    Tuto=Button(canvasniveau,text="Aide",font="Constantia 15",justify="center",overrelief="groove",activeforeground="blue",activebackground=
    Tuto.grid(row=1,column=1,padx=20,pady=20)

    # Demarre la fonction qui initialise le labyrinthe
    LabFen()

    Niveau.mainloop()

# Fonction qui initialise le labyrinthe avant que le niveau soit selectionne
def LabFen():
    global C,Lab

    Lab=Toplevel()

    Lab.title("En attente du choix du niveau...")
    Lab.resizable(width=False,height=False)
    Lab.iconbitmap("img/icone/claptrap.ico")
    Lab.attributes("-fullscreen",1)
    Lab.config(bg="black")
    Lab.iconify()
    C=Canvas(Lab,width=900,height=900,background="white")
    C.pack()
    C.mainloop()

# Fonction qui permet l'affichage des scores.
# Utilise les deux fonctions "TriScore" et "TriDernierScore"
# Selon le nombre de scores, et s'il existe ou non un dernier score, la fonction va afficher differement le tableau des scores.

def LireScore(niv):

    TriScore(niv)
    TriDernierScore(niv)


    if len(DernierJoueur)<=3:
        if int(DernierJoueur[2])>1:
            d=str(DernierJoueur[2])+"eme"+" "+str(DernierJoueur[0])+" "+str(DernierJoueur[1])
        elif int(DernierJoueur[2])==1:
            d=str(DernierJoueur[2])+"er"+" "+str(DernierJoueur[0])+" "+str(DernierJoueur[1])
    else:
        d=DernierJoueur


    if len(ListeComplete)==0:
        LabelScore=Label(FenScore,text="Aucun score n'est enregistre pour ce niveau.",font="Constantia 20",padx=10,pady=10)
        LabelScore.pack()
    elif len(ListeComplete)==1:
        premier=ListeComplete[0]
        a=str(premier[0])+" "+str(premier[1])
        LabelPremier=Label(FenScore,text=a,image=ImgPremier,compound="left",font="Constantia 20",padx=10,pady=10)
        LabelPremier.pack()
        LabelDernier=Label(FenScore,text=d,font="Constantia 20",bg="gray",padx=10,pady=10)
        LabelDernier.pack()

    elif len(ListeComplete)==2:
        premier=ListeComplete[0]
        a=str(premier[0])+" "+str(premier[1])
        second=ListeComplete[1]
        b=str(second[0])+" "+str(second[1])
        LabelPremier=Label(FenScore,text=a,image=ImgPremier,compound="left",font="Constantia 20",padx=10,pady=10)
        LabelPremier.pack()
        LabelSecond=Label(FenScore,text=b,image=ImgSecond,compound="left",font="Constantia 20",padx=10,pady=10)
        LabelSecond.pack()
        LabelDernier=Label(FenScore,text=d,font="Constantia 20",bg="gray",padx=10,pady=10)
        LabelDernier.pack()

    elif len(ListeComplete)>=3:
        premier=ListeComplete[0]
        a=str(premier[0])+" "+str(premier[1])
        second=ListeComplete[1]
        b=str(second[0])+" "+str(second[1])
        troisieme=ListeComplete[2]
        c=str(troisieme[0])+" "+str(troisieme[1])

        LabelPremier=Label(FenScore,text=a,image=ImgPremier,compound="left",font="Constantia 20",padx=10,pady=10)
        LabelPremier.pack()
        LabelSecond=Label(FenScore,text=b,image=ImgSecond,compound="left",font="Constantia 20",padx=10,pady=10)
        LabelSecond.pack()
        LabelTroisieme=Label(FenScore,text=c,image=ImgTroisieme,compound="left",font="Constantia 20",padx=10,pady=10)
        LabelTroisieme.pack()
        LabelDernier=Label(FenScore,text=d,font="Constantia 20",bg="gray",padx=10,pady=10)
        LabelDernier.pack()

    FenScore.mainloop()


# Fonction qui permet de trier la liste des scores d'un niveau
```

```
# Fonction detaillee dans le dossier-projet
def TriScore(niv):
    global ListeComplete,DernierJoueur

    NomSave="scores/"+str(niv)+".txt"
    fich=open(NomSave,"r")
    ListeScore=fich.readlines()
    fich.close()

    ListeComplete=[]
    for compteur in range(0,len(ListeScore),2):
        namejoueur=ListeScore[compteur]
        timejoueur=ListeScore[compteur+1]
        if "\n" in timejoueur:
            timejoueur=timejoueur.replace("\n","")
        if "\n" in namejoueur:
            namejoueur=namejoueur.replace("\n","")

        ListeComplete.append([namejoueur,timejoueur])

    print(ListeComplete)
    ListeComplete.sort(key=lambda colonne: colonne[1])
    print(ListeComplete)

# Fonction qui permet de trier le dernier score d'un niveau
# Le fichier "dernier-niveau" poss de uniquement trois lignes, ou une seule s'il est vide.
# S'il est vide, il sera affich  qu'il n'y a aucun score.
# Sinon, la fonction renvoie une liste contenant le nom, score et classement
def TriDernierScore(niv):
    global DernierJoueur
    if niv=="facile":
        DerniereSave="scores/dernier-facile.txt"
        fich=open(DerniereSave,"r")
        DernierJoueur=fich.readlines()
        fich.close()
        if len(DernierJoueur)>=2:
            DernierJoueur[0]=DernierJoueur[0].replace("\n","")
            DernierJoueur[1]=DernierJoueur[1].replace("\n","")
            DernierJoueur[2]=DernierJoueur[2].replace("\n","")
        else:
            DernierJoueur="Il n'y a pas de score enregistre recemment."
    elif niv=="moyen":
        DerniereSave="scores/dernier-moyen.txt"
        fich=open(DerniereSave,"r")
        DernierJoueur=fich.readlines()
        fich.close()
        if len(DernierJoueur)>=2:
            DernierJoueur[0]=DernierJoueur[0].replace("\n","")
            DernierJoueur[1]=DernierJoueur[1].replace("\n","")
            DernierJoueur[2]=DernierJoueur[2].replace("\n","")
        else:
            DernierJoueur="Il n'y a pas de score enregistre recemment."

    elif niv=="difficile":
        DerniereSave="scores/dernier-difficile.txt"
        fich=open(DerniereSave,"r")
        DernierJoueur=fich.readlines()
        fich.close()
        if len(DernierJoueur)>=2:
            DernierJoueur[0]=DernierJoueur[0].replace("\n","")
            DernierJoueur[1]=DernierJoueur[1].replace("\n","")
            DernierJoueur[2]=DernierJoueur[2].replace("\n","")
        else:
            DernierJoueur="Il n'y a pas de score enregistre recemment."

# Fenetre du tableau des scores
def Scoreboard():
    global LB

    LB=Toplevel()
    LB.title("Tableau des scores")
    LB.resizable(width=False,height=False)
    LB.iconbitmap("img/icone/claptrap.ico")

    LBcanvas=Canvas(LB)
    LBcanvas.pack()
    LBcanvas.create_image(0,0,image=background_image,anchor="nw")

    ScoreFacile=Button(LBcanvas,text="Facile",font="Constantia 15",justify="center",overrelief="groove",bg="white",activeforeground="blue",a
    ScoreFacile.pack(side="left",padx=20,pady=20)

    ScoreMoyen=Button(LBcanvas,text="Moyen",font="Constantia 15",justify="center",overrelief="groove",bg="white",activeforeground="blue",act
    ScoreMoyen.pack(side="left",padx=20)

    ScoreDifficile=Button(LBcanvas,text="Difficile",font="Constantia 15",justify="center",overrelief="groove",bg="white",activeforeground="b
    ScoreDifficile.pack(side="left",padx=20,pady=20)

    LB.mainloop()

# Onglet facile du tableau des scores
# Utilise la fonction "LireScore" pour afficher les scores du niveau "facile"
def OngletFacile():
    global FenScore
    FenScore=Toplevel()
    FenScore.title("Niveau : FACILE")
    FenScore.resizable(width=False,height=False)
    FenScore.iconbitmap("img/icone/claptrap.ico")
    LireScore("facile")
    return

# Onglet moyen du tableau des scores
# Utilise la fonction "LireScore" pour afficher les scores du niveau "moyen"
def OngletMoyen():
    global FenScore
    FenScore=Toplevel()
    FenScore.title("Niveau : MOYEN")
    FenScore.resizable(width=False,height=False)
```

```
    FenScore.iconbitmap("img/icone/claptrap.ico")
    LireScore("moyen")
    return

# Onglet difficile du tableau des scores
# Utilise la fonction "LireScore" pour afficher les scores du niveau "difficile"
def OngletDifficile():
    global FenScore
    FenScore=Toplevel()
    FenScore.title("Niveau : DIFFICILE")
    FenScore.resizable(width=False,height=False)
    FenScore.iconbitmap("img/icone/claptrap.ico")
    LireScore("difficile")
    return

# Fonction qui cree l'interface du chronometre, et recupere la premiere input du temps
# Execute a la fin la fonction "Chrono"
def CreateChrono():
    global debut,TexteChrono
    # Temps de debut, recupere une seule fois par niveau
    debut=time()
    FrameChrono=Frame(Lab,bg="black")
    FrameChrono.pack(anchor="s")

    TexteChrono=Label(FrameChrono,text="00:00",font="Helvetica 16",fg="white",bg="black")
    TexteChrono.pack(side="right")

    imgchrono=Label(FrameChrono,image=ImgChrono,bg="black")
    imgchrono.pack(side="left")

    Chrono()


    Lab.mainloop()


# Fait fonctionner le chronometre en actualisant chaque seconde le temps affiche
# Le temps affiche est une difference entre le temps de debut et le temps de fin
Chrono_Arreter=0
def Chrono():
    global debut,temps,Chrono_Arreter
    # Temps recupere chaque seconde, permettant de faire la difference entre "debut" et "fin"
    fin=time()
    difference=fin-debut

    # Formate le temps sous le format "MINUTE : SECONDE"
    local=localtime(difference)
    temps=strftime("%M:%S",local)

    TexteChrono.config(text=temps)
    TexteChrono.update()

    if Chrono_Arreter==0:
        # S'actualise chaque seconde
        Lab.after(1000,Chrono)


# Chaque fois que la souris bouge, deplace le cercle sur la souris si c'est possible et deplace egalement Claptrap (y=48 correspond a la hau
# Si la porte est fermee, Claptrap ne peut pas la depasser (x=44 correspond au depart, 212 a la porte)
# Si la porte est ouverte, Claptrap peut se deplacer au-dela (x=455 correspond a la fin de l'arrivee)

def MenuMove(event):
    #global Cercle_Menu,Claptrap_Menu
    mx,my=event.x,event.y

    if 0<=mx<=500 and 0<=my<=600:
        canvasmenu.coords(Cercle_Menu,mx,my)
    if PorteOuverte==0:
        if mx>44 and mx<212:
            canvasmenu.coords(Claptrap_Menu,mx,48)
    if PorteOuverte==1:
        if mx>44 and mx<455:
            canvasmenu.coords(Claptrap_Menu,mx,48)
        return


# En appuyant sur le clic, verifie quel objet est appuye pour agir en consequence
def MenuCheck(event):
    global PorteOuverte,Cercle_Menu,Reset_Check,Clignotement_Status
    mx,my=event.x,event.y


    # Verifie si le clic est sur la ligne d'arrivee - execute le clignotement du niveau, en annulant "Reset_Check"
    if PorteOuverte==1:
        if mx>425 and 33<=my<=63 and Clignotement_Status==0:
            canvasmenu.delete(Cercle_Menu)
            Clignotement_Status=1
            Reset_Check=0
            Clignotement()


    # Verifie si le clic est sur la clef - cela detruit la clef et la porte
    if abs(mx-clefx)<=16 and abs(my-clefy)<=16:
        canvasmenu.delete(Clef_Menu)
        canvasmenu.delete(Porte_Menu)
        PorteOuverte=1


    # Verifie si le clic est sur le bonus - cela agrandit le cercle autour du curseur
    elif abs(mx-bonusx)<=10 and abs(my-bonusy)<=10 and Clignotement_Status==0:
        canvasmenu.delete(Bonus_Menu)
        canvasmenu.delete(Cercle_Menu)
        Cercle_Menu=canvasmenu.create_image(mx,my,image=NewCercle)


    # Verifie pour les boutons. x=250 / y(Jouer)=220 / y(Tableau)=320 / y(Quitter)=420
    # Hauteur d'un bouton : 41
```

```
    # Largeur d'un bouton : 177
    elif abs(mx-250)<=88 and abs(my-220)<=20:
        SelectionNiveau()

    elif abs(mx-250)<=88 and abs(my-320)<=20:
        Scoreboard()

    elif abs(mx-250)<=88 and abs(my-420)<=20:
        Quit()


# Execute quand la ligne d'arrivee est appuyee (si la porte est ouverte)
# Fait clignoter le menu entre deux images, mais arrete si le bouton de reset (entree) est appuye (verifie par Reset_Check)
def Clignotement():
    global Fond_Menu,Reset_Check
    if Reset_Check==0:
        canvasmenu.delete(Fond_Menu)
        Fond_Menu=canvasmenu.create_image(0,0,image=background_mainmenu_bonus1,anchor="nw")
        canvasmenu.lift(Claptrap_Menu)
        canvasmenu.lift(Jouer)
        canvasmenu.lift(Tableau)
        canvasmenu.lift(Quitter)

        mainF.after(500,Clignotement2)

def Clignotement2():
    global Fond_Menu,Reset_Check
    if Reset_Check==0:
        canvasmenu.delete(Fond_Menu)
        Fond_Menu=canvasmenu.create_image(0,0,image=background_mainmenu_bonus2,anchor="nw")
        canvasmenu.lift(Claptrap_Menu)
        canvasmenu.lift(Jouer)
        canvasmenu.lift(Tableau)
        canvasmenu.lift(Quitter)
        mainF.after(500,Clignotement)

# En appuyant sur la touche entree, cela reinitialise
def ResetMenu(event):
    global clefx,clefy,bonusx,bonusy,PorteOuverte,Cercle_Menu,Claptrap_Menu,Porte_Menu,Clef_Menu,Bonus_Menu,Reset_Check,Jouer,Tableau,Quitte
    canvasmenu.delete("all")
    canvasmenu.unbind("<Motion>")
    canvasmenu.unbind("<Button-1>")
    Fond_Menu=canvasmenu.create_image(0,0,image=background_mainmenu,anchor="nw")
    Jouer=canvasmenu.create_image(250,220,image=JouerBouton)
    Tableau=canvasmenu.create_image(250,320,image=TableauBouton)
    Quitter=canvasmenu.create_image(250,420,image=QuitterBouton)
    Claptrap_Menu=canvasmenu.create_image(44+15,48,image=Claptrap,tags="Claptrap")
    PorteOuverte=0
    Porte_Menu=canvasmenu.create_image(242,48,image=Porte)
    clefx,clefy=randint(50,450),randint(100,500)
    Clef_Menu=canvasmenu.create_image(clefx,clefy,image=Clef)
    bonusx,bonusy=randint(50,450),randint(100,500)
    Bonus_Menu=canvasmenu.create_image(bonusx,bonusy,image=Bonus)
    Cercle_Menu=canvasmenu.create_image(event.x,event.y,image=Cercle)
    Clignotement_Status=0
    canvasmenu.bind("<Motion>",MenuMove)
    canvasmenu.bind("<Button-1>",MenuCheck)
    Reset_Check=1



canvasmenu=Canvas(mainF,width=500,height=600)
canvasmenu.pack()
Fond_Menu=canvasmenu.create_image(0,0,image=background_mainmenu,anchor="nw")

Jouer=canvasmenu.create_image(250,220,image=JouerBouton)
Tableau=canvasmenu.create_image(250,320,image=TableauBouton)
Quitter=canvasmenu.create_image(250,420,image=QuitterBouton)


# Fonctionnalites pour le menu interactif
# Sert de mini-tutoriel pour introduire les elements basiques du jeu
# Utilise les fonctions : MenuMove - MenuCheck - ResetMenu

Claptrap_Menu=canvasmenu.create_image(44+15,48,image=Claptrap,tags="Claptrap")
Porte_Menu=canvasmenu.create_image(242,48,image=Porte)
clefx,clefy=randint(50,450),randint(100,500)
Clef_Menu=canvasmenu.create_image(clefx,clefy,image=Clef)
bonusx,bonusy=randint(50,450),randint(100,500)
Bonus_Menu=canvasmenu.create_image(bonusx,bonusy,image=Bonus)
Cercle_Menu=canvasmenu.create_image(250,300,image=Cercle)
PorteOuverte=0
Reset_Check=0
Clignotement_Status=0

canvasmenu.bind("<Motion>",MenuMove)
canvasmenu.bind("<Button-1>",MenuCheck)
mainF.bind("<Return>",ResetMenu)


mainF.mainloop()
```