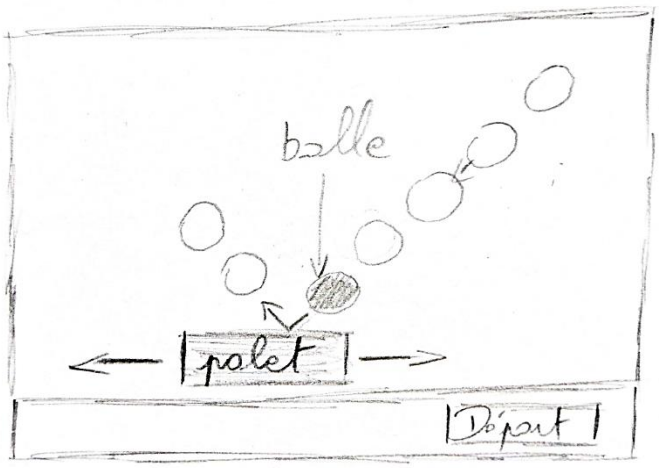
Info 14 - Tkinter – Balle rebondissante

OBJECTIFS : On réalise ici une application du type jeu qui permettra d'utiliser les fonctionnalités liées aux évènements et aux animations de tkinter. L'objectif est surtout de vous initier à la réalisation d'un projet.

L'évaluation de ce travail est basée sur le rendu du fichier *.py* qui sera constitué. **On ne demande pas** de constituer de fichier texte contenant des copies d'écran.

Le code python que l'on constituera étape par étape dans la suite, permet de créer le jeu suivant :

« Une balle se déplace automatiquement en diagonale et rebondit sur les côtés de la fenêtre. Le joueur déplace un palet horizontalement, avec les touches ⇨ et ⇦ du clavier, pour intercepter la balle qui descend.»



1. DEMARRAGE ET IMPORTATION DES BIBLIOTHEQUES :

⇨ Pour débuter, télécharger le dossier *tp14.zip* contenant les fichiers des images et son qui seront utilisés. Le dézipper dans votre espace personnel sur ***U:*** . Ouvrir un nouveau fichier *.py* dans le dossier contenant ces images et le sauvegarder sous le nom ***tp14.py*** .

⇨ Copié-collé le code donné ci-dessous, dans votre fichier *tp14.py* . Il permet d'importer les librairies et d'organiser votre fichier.

```
# Modules --------------------------------------------------------------
from tkinter import Tk , Menu , Canvas , Label , Entry , StringVar ,Button, Text
from PIL import Image, ImageTk # pip install pillow
from random import randint
from winsound import PlaySound, SND_ASYNC # pip install winaudio

# Fonctions ------------------------------------------------------------


# Variables globales ---------------------------------------------------

# Main -----------------------------------------------------------------
```

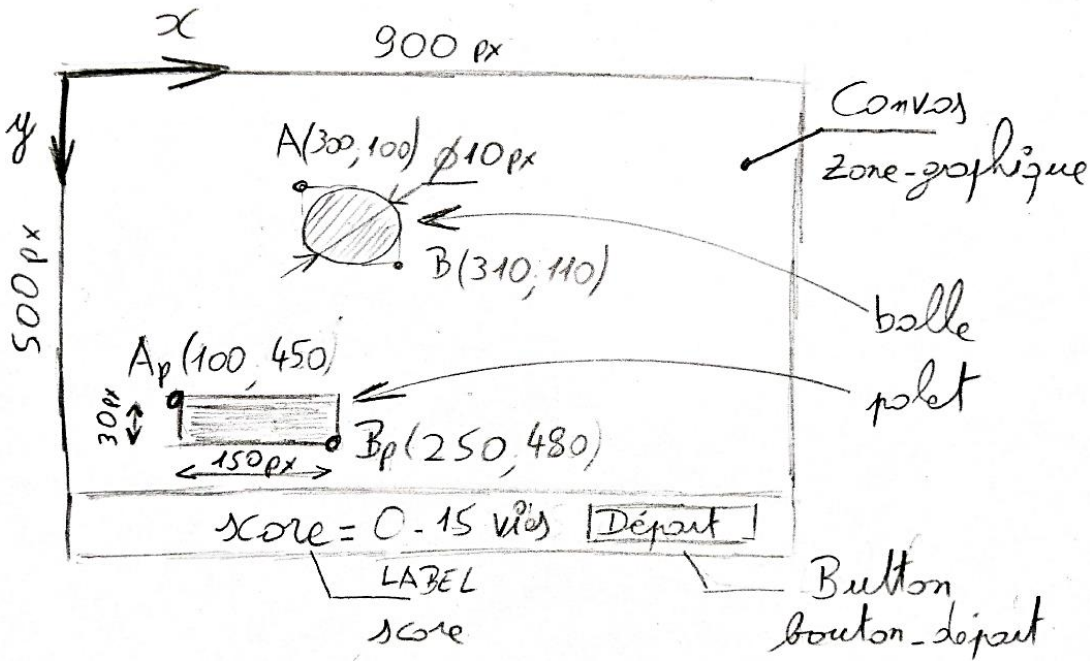
Si la librairie *winsound* n'est pas installée sur votre ordinateur, l'installer en exécutant dans la console la ligne : « *pip install winaudio* ».

Si l'exécution de *pip* nécessite une mise à jour, exécuter au préalable dans la console, la ligne :
« *pip install --upgrade pip --user* ».

2. CREATION DE LA PARTIE GRAPHIQUE :

On dimensionne l'aire de jeu en réalisant un croquis à main levée.

On se basera dans la suite, sur les dimensions données ci-contre :



Pour plus de clarté, on compartimente le code lié à la création de la partie graphique, dans les fonctions suivantes :

- La fenêtre *tkinter* sera créée dans la fonction *creer_fenetre(),*
- Les widgets *Canvas*, *Label* et *Button* dans la fonction *creer_widgets(),*
- La balle et le palet dans la fonction *creer_graphisme().*

⇨ Créer la fonction *creer_fenetre()* :

```
# Fonctions ---------------------------------------------------------------
def creer_fenetre() :
    fenetre = Tk()
    fenetre.title("Baballe")
    return fenetre


# Variables globales ------------------------------------------------------

# Main --------------------------------------------------------------------
fenetre = creer_fenetre()

fenetre.mainloop()
```

Ne pas oublier

⇨ Créer la fonction *creer_widgets()* et la callback *go() :*

```
def creer_widgets() :
    zone_graphique = Canvas(fenetre, width=900, height=500 , bg = 'black')
    zone_graphique.grid(row = 0 , column = 0 , columnspan = 2)

    score = Label(fenetre, text = "Clique sur le bouton")
    score.grid(row = 1 , column = 0)

    bouton_depart = Button(fenetre, text = "Départ" , width = 12 , command = go)
    bouton_depart.grid(row = 1, column = 1)

    return zone_graphique,score,bouton_depart

def go() :
    bouton_depart.destroy()
    zone_graphique.grid(columnspan = 1)
    score.configure(text = "Le jeu a commencé")
```

On supprime ce widget avec la méthode *destroy()*

On modifie ce widget avec la méthode *configure()*

⇨ Appeler les fonctions *creer_fenetre()* et *creer_widgets()* dans le programme principal :

```
# Variables globales ------------------------------------------------------

# Main --------------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,score,bouton_depart = creer_widgets()

fenetre.mainloop()
```

⇨ Un bon moyen pour donner du sens … est de jeter un coup d'œil rapide à ce qu'il faut coder … pour ensuite l'écrire seul en utilisant au maximum **l'autocomplétion**. Par exemple, pour écrire la ligne :

```
zone_graphique = Canvas(fenetre, width=900,
```

On commence par écrire les 3 premières lettres `zon` … la variable `zone_graphique` est déjà reconnue. En tapant sur la touche *Tabulation* du clavier, on valide cette proposition d'autocomplétion et on continue …

en écrivant `zone_graphique = Can` …. la classe *Canvas* est reconnue. On accepte en appuyant sur la touche *tabulation*.

en écrivant `zone_graphique = Canvas(fen)` , la variable fenetre est reconnue, ….etc …

⇨ Pour écrire la ligne

```
zone_graphique,score,bouton_depart = creer_widgets()
```

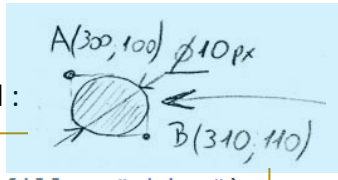
on utilise au maximum le COPIE-COLLE afin d'éviter les erreurs de recopies.

⇨ On vérifie toujours qu'une erreur ne s'est pas glissée dans l'écriture des variables. En **double-cliquant** sur le nom d'une variable, ce même nom est mis en surbrillance dans les autres parties du script : (exemple ci-dessous pour vérifier l'écriture correcte du nom « zone_*graphique* »)

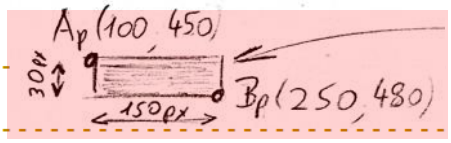
```
def creer_widgets() :
    zone_graphique = Canvas(fenetre, width=900, height=500 , bg = 'black')
    zone_graphique.grid(row = 0 , column = 0 , columnspan = 2)

    score = Label(fenetre, text = "Clique sur le bouton")
    score.grid(row = 1 , column = 0)

    bouton_depart = Button(fenetre, text = "Départ" , width = 12 , command = go)
    bouton_depart.grid(row = 1, column = 1)

    return zone_graphique,score,bouton_depart

def go() :
    bouton_depart.destroy()
    zone_graphique.grid(columnspan = 1)
    score.configure(text = "Le jeu a commencé")

# Variables globales ------------------------------------------------------

# Main --------------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,score,bouton_depart = creer_widgets()
```

⇨ et bien sûr, **exécuter très souvent le code** pour valider votre écriture. Ne pas attendre d'écrire 10 lignes avant de le faire …. Ne pas hésiter à insérer des *print(…)* dans le code pour afficher l'état des variables qui posent problèmes.

⇨ Créer la fonction *créer_graphisme()* et l'appeler dans le programme principal :



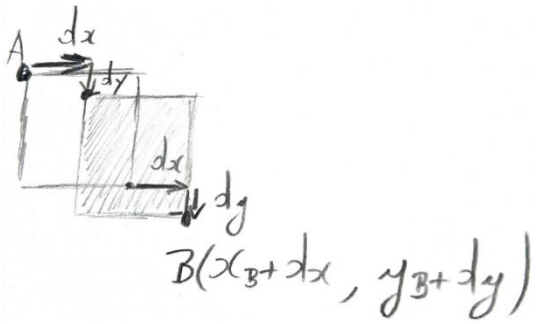

```
def creer_graphisme() :
    balle = zone_graphique.create_oval(300 , 100 , 310 , 110 , fill = "white")
    palet = zone_graphique.create_rectangle(100 ,450 ,250 ,480 , fill = "white")
    return  balle,palet

# Variables globales ----------------------------

# Main ------------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,score,bouton_depart = creer_widgets()
balle,palet = creer_graphisme()

fenetre.mainloop()
```



⇨ Tester le démarrage du jeu …

3. CREATION DE L'ANIMATION DE LA BALLE :

Pour déplacer la balle, on déplace les points A et B qui encadrent la balle, suivant une translation de vecteur $\vec{u}\begin{pmatrix} dx \\ dy \end{pmatrix}$. On suppose au départ que $dx = dy = 5\ px$. Ce déplacement est répété toutes les 50 ms en utilisant la méthode *after()* vue en cours.



⇨ Créer la fonction *animation_balle()* et appeler cette fonction dans la fonction *go()* qui est exécutée lorsque le joueur clique sur le bouton « *Départ* » :

```
def go() :
    bouton_depart.destroy()
    zone_graphique.grid(columnspan = 1)
    score.configure(text = "Le jeu a commencé")
    animation_balle()

def animation_balle():
    dx = 5
    dy = 5
    xA,yA,xB,yB = zone_graphique.coords(balle)
    zone_graphique.coords(balle , xA+dx ,yA+dy, xB+dx ,yB+dy)
    fenetre.after(50 ,  animation_balle)

# Variables globales ----------------------------

# Main ------------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,score,bouton_depart = creer_widgets()
balle,palet = creer_graphisme()

fenetre.mainloop()
```

Appel de la fonction

On **récupère** les coordonnées des points A et B

On **modifie** les coordonnées des points A et B

On répète l'exécution toutes les 50 ms

⇨ Tester ce script … toutes les 50 ms, la balle se déplace de 5 px vers la droite et de 5 px vers le bas …. la balle finit par sortir de la fenêtre.

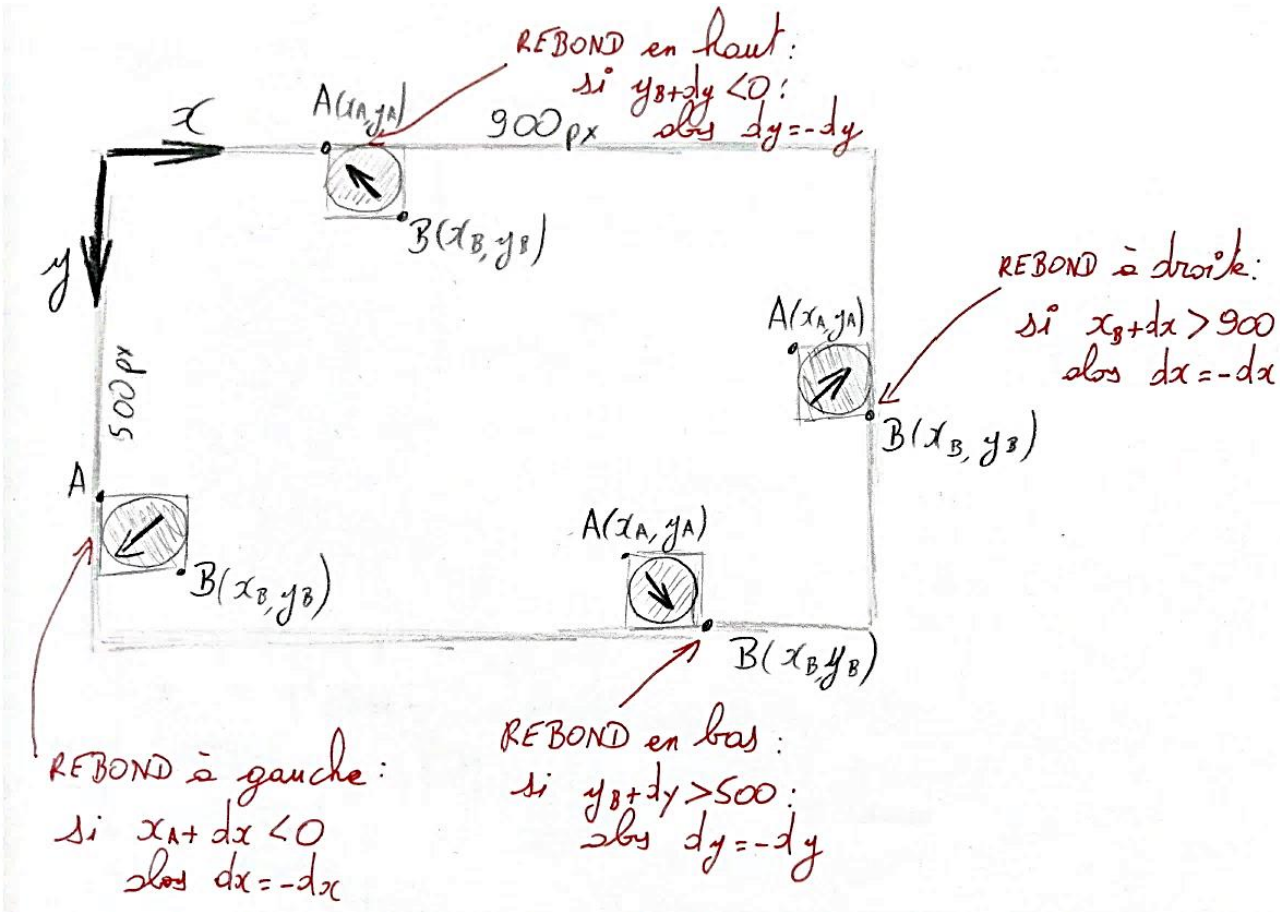
⇨ On modifie légèrement ce script, en mettant les valeurs de dx et dy en argument de la fonction *animation_balle().* Cela ne change rien pour l'instant, mais cela présente l'avantage de pouvoir modifier ultérieurement les valeurs de dx et dy, lors des appels répétés de la fonction.

```
def go() :
    bouton_depart.destroy()
    zone_graphique.grid(columnspan = 1)
    score.configure(text = "Le jeu a commencé")
    animation_balle(5,5)

def animation_balle(dx,dy):
    xA,yA,xB,yB = zone_graphique.coords(balle)
    zone_graphique.coords(balle , xA+dx ,yA+dy, xB+dx ,yB+dy)
    fenetre.after(50 ,  animation_balle , dx , dy)
```

1er appel de la fonction : ici $dx = dy = 5$

Avec la méthode after(), c'est comme cela qu'on place les arguments

⇨ On améliore encore le script en faisant rebondir la balle. Le rebond s'obtient en changeant le signe du déplacement dx lorsque la balle arrive sur les côtés gauche ou droit et celui du déplacement dy lorsque la balle arrive sur les côtés bas ou haut. La fenêtre a comme taille 900 x 500 :



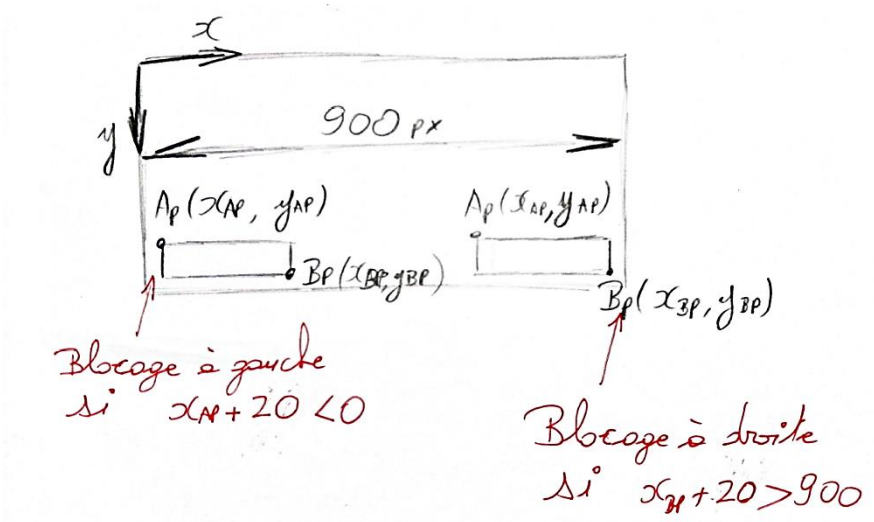
Le script modifié devient :

```
def animation_balle(dx,dy):
    xA,yA,xB,yB = zone_graphique.coords(balle)

    if xB + dx > 900 or xA + dx < 0 :
        dx = -dx
    if yB + dy > 500 or yA + dy < 0 :
        dy = -dy

    zone_graphique.coords(balle , xA+dx ,yA+dy, xB+dx ,yB+dy)
    fenetre.after(50 ,  animation_balle , dx , dy)
```

⇨ Tester l'ensemble.

4. DEPLACEMENT DU PALET :

Pour déplacer le palet, on déplace les points Ap et Bp qui encadrent le palet, de 20 px vers la droite si la touche ⇨ du clavier est actionnée et de 20 px vers la gauche pour la touche ⇦ . On est ainsi amené à créer 2 évènements clavier. Les fonctions callback correspondantes seront appelées *gauche()* et *droite()*. Cela donne le script suivant :



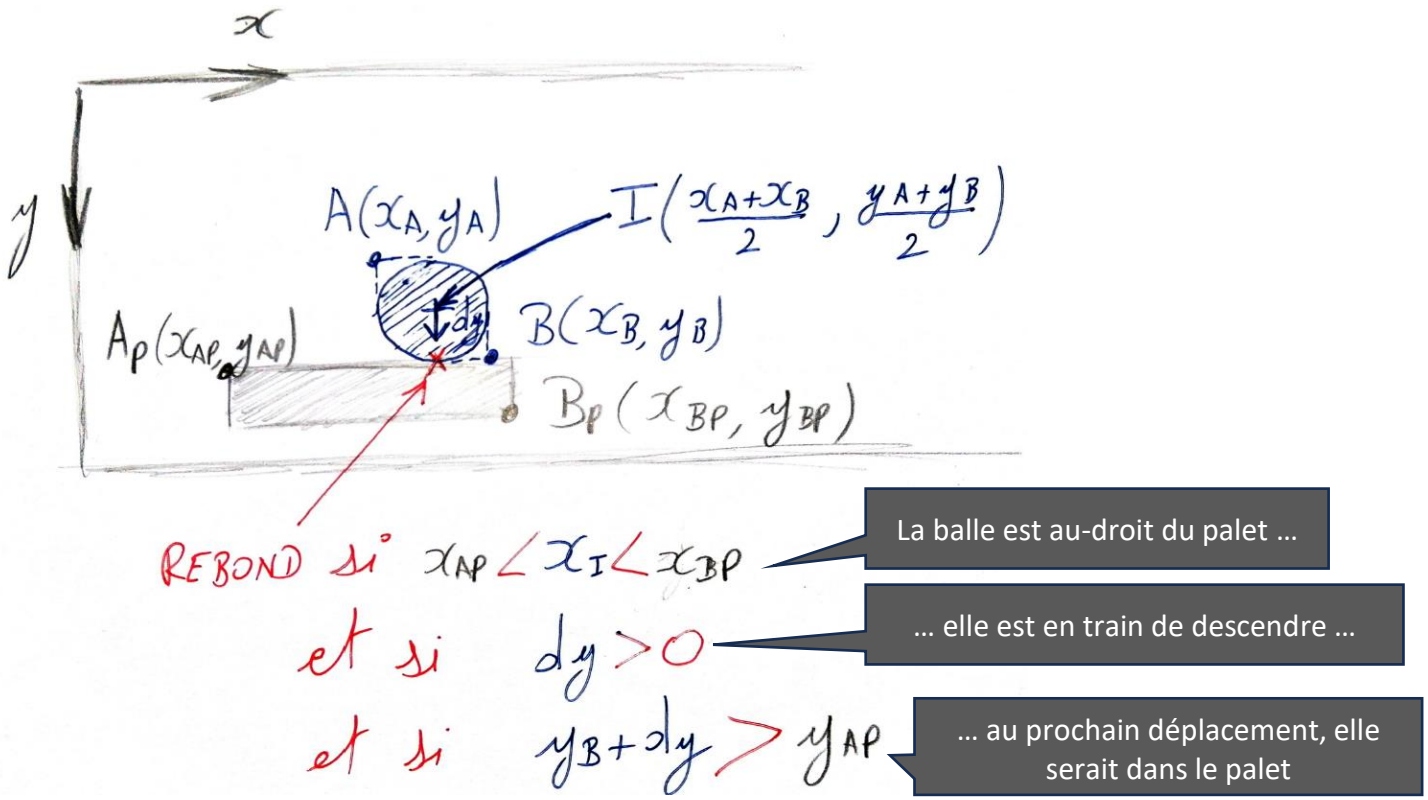

```
def droite(event):
    xAP,yAP,xBP,yBP = zone_graphique.coords(palet)
    if xBP + 20 <= 900 :
        zone_graphique.coords(palet,xAP+20,yAP,xBP+20,yBP)

def gauche(event):
    xAP,yAP,xBP,yBP = zone_graphique.coords(palet)
    if xAP - 20 >= 0 :
        zone_graphique.coords(palet,xAP-20,yAP,xBP-20,yBP)
```

```
# Main ----------------------------------------------
fenetre = creer_fenetre()
zone_graphique,score,bouton_depart = creer_widgets()
balle,palet = creer_graphisme()
fenetre.bind("<Right>",droite)
fenetre.bind("<Left>",gauche)

fenetre.mainloop()
```

⇨ Tester l'ensemble.

5. REBOND DE LA BALLE SUR LE PALET :

La balle rebondit sur le côté haut du palet lorsque les conditions suivantes sont vérifiées :



Pour ne pas surcharger la fonction *animation_balle(),* on gère le rebond sur le palet dans une nouvelle fonction nommée *rebond_ballePalet().* Comme ici les variables ne sont pas définies dans le programme principal, il s'agit de les passer en argument et de retourner celles qui sont modifiées (pas la peine de retourner les autres). Cela donne le script ci-dessous :

```
def animation_balle(dx,dy):
    xA,yA,xB,yB = zone_graphique.coords(balle)

    if xB + dx > 900 or xA + dx < 0 :
        dx = -dx
    if yB + dy > 500 or yA + dy < 0 :
        dy = -dy

    dy = rebond_ballePallet(xA,yA,xB,yB,dy)

    zone_graphique.coords(balle , xA+dx ,yA+dy, xB+dx ,yB+dy)
    fenetre.after(50 ,  animation_balle , dx , dy)

def rebond_ballePallet(xA,yA,xB,yB,dy) :
    xAP,yAP,xBP,yBP = zone_graphique.coords(palet)
    xI = (xA + xB)//2
    if xAP < xI and xI < xBP and dy > 0 and yB+dy > yAP :
        dy = -dy
    return dy
```

On appelle cette nouvelle fonction.

6. CHANGEMENT DE COULEURS A CHAQUE REBOND SUR LES COTES DE LA FENETRE :

On améliore le rendu visuel en changeant la couleur de la balle **et** de la fenêtre après chaque rebond sur les côtés de la fenêtre.

On écrit dans un premier temps le script d'une fonction nommée *couleurAlea()* qui retourne un string définissant un code couleur hexadécimal (par exemple *'#18AF0D'*).

⇨ Ecrire ce script. Exécuter le fichier *tp14.py* et tester dans la console le fonctionnement de cette nouvelle fonction. Par exemple :

```
>>> couleurAlea()
'#0A85FB'

>>> couleurAlea()
'#E41049'
```

```
def couleurAlea() :
    hexa = "0123456789ABCDEF"
    couleur = "#"
    for i in range(6) :
        num = randint(0,15)
        couleur = couleur + hexa[num]
    return couleur
```

⇨ Modifier le script de la fonction *animation_balle()* pour obtenir ce changement de couleur :

```
def animation_balle(dx,dy):
    xA,yA,xB,yB = zone_graphique.coords(balle)

    modifCouleur = False
    if xB + dx > 900 or xA + dx < 0 :
        dx = -dx
        modifCouleur = True
    if yB + dy > 500 or yA + dy < 0 :
        dy = -dy
        modifCouleur = True

    if modifCouleur == True :
        zone_graphique.itemconfigure(balle,fill=couleurAlea())
        zone_graphique.configure(bg = couleurAlea())

    dy = rebond_ballePallet(xA,yA,xB,yB,dy)

    zone_graphique.coords(balle , xA+dx ,yA+dy, xB+dx ,yB+dy)
    fenetre.after(50 ,  animation_balle , dx , dy)
```

On modifie une caractéristique de la balle avec ***itemconfigure(),*** car la balle est un élément graphique inséré dans le Canvas nommé ici *zone_graphique*

On modifie une caractéristique du Canvas *zone_graphique* avec ***configure()*** car le Canvas est un widget.

Remarque : le script de cette fonction animation_*balle()* devient déjà conséquent. Pour faciliter sa relecture, les actions réalisées par ces lignes sont bien séparées et dans mesure du possible ne sont pas trop mélangées. Ici dans animation_*balle()*, on récupère les coordonnées de la balle, puis on gère l'éventuelle sortie de la fenêtre, puis on modifie les couleurs, puis on traite le rebond sur palet, puis on modifie les coordonnées de la balle et enfin on répète la fonction. On peut rajouter des commentaires, … mais l'expérience montre qu'il ne faut pas en abuser. Ici c'est inutile.

7. MISE EN PLACE D'UN COMPTEUR DE REBOND BALLE SUR PALET :

On améliore encore le code, en affichant dans le widget *score*, le nombre de rebonds de la balle sur le palet. On en profite pour ajouter un son et augmenter le diamètre de la balle de 10 px après chaque rebonds.

Cette amélioration entraine la changement ci-dessous du script de la fonction *rebondBallePallet()* :

```
def rebond_ballePallet(xA,yA,xB,yB,dy) :
    global N
    xAP,yAP,xBP,yBP = zone_graphique.coords(palet)
    xI = (xA + xB)//2
    if xAP < xI and xI < xBP and dy > 0 and yB+dy > yAP :
        dy = -dy
        xB = xB + 10
        yB = yB + 10
        N = N + 1
        score.configure(text = f"score = {N} ")
        PlaySound("bruit.wav", SND_ASYNC)
    return xB,yB,dy
```

On retourne aussi xB et yB car ces valeurs ont été modifiées

Le diamètre de la balle augmente de 10 px

compteur de rebonds

Affichage du nombre de rebonds dans le widget placée dans la variable *score*

et de la fonction *animation_balle() :*

```
def animation_balle(dx,dy):
    xA,yA,xB,yB = zone_graphique.coords(balle)

    modifCouleur = False
    if xB + dx > 900 or xA + dx < 0 :
        dx = -dx
        modifCouleur = True
    if yB + dy > 500 or yA + dy < 0 :
        dy = -dy
        modifCouleur = True

    if modifCouleur == True :
        zone_graphique.itemconfigure(balle,fill=couleurAlea())
        zone_graphique.configure(bg = couleurAlea())

    xB,yB,dy = rebond_ballePallet(xA,yA,xB,yB,dy)

    zone_graphique.coords(balle , xA+dx ,yA+dy, xB+dx ,yB+dy)
    fenetre.after(50 ,  animation_balle , dx , dy)
```

On actualise les variables retournées

et du programme principal pour initialiser la variable N est qui est déclarée en variable globale dans la fonction *rebondBallePallet()* : (variables globales de préférence en majuscule)

```
# Variables globales ------------------------------------
N = 0
# Main --------------------------------------------------
fenetre = creer_fenetre()
```

Initialisation de N dans le programme principal

⇨ Réaliser ces modifications … bien sûr, toujours en leur donnant du sens. Tester l'ensemble …

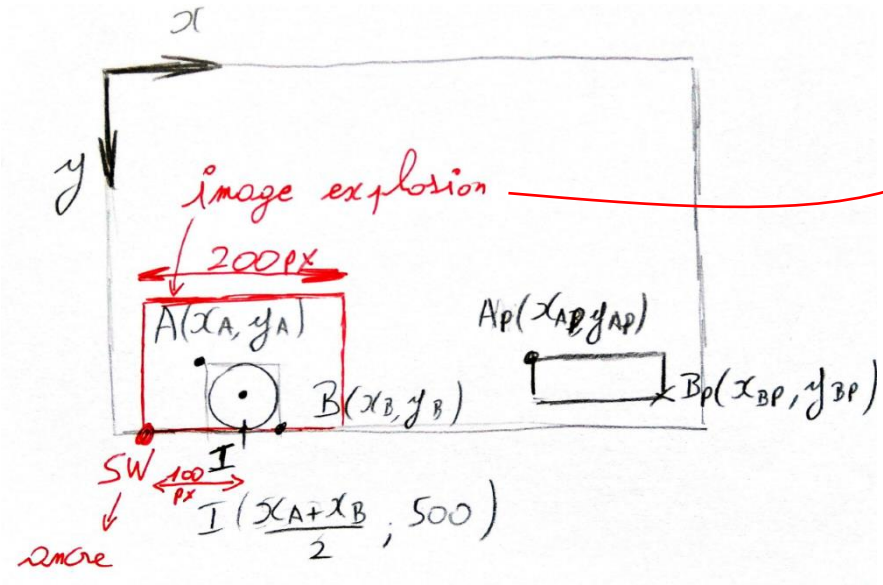
8. MAUVAISE INTERCEPTION DE LA BALLE QUI TOMBE :

On continue l'amélioration de ce jeu en gérant à présent les rebonds entre la balle et le côté inférieur de la fenêtre. Cette situation se produit lorsque le joueur n'arrive pas à intercepter la balle qui tombe. On propose alors :

- au contact balle / fenêtre, de faire apparaître une image d'explosion,
- de raccourcir la largeur du palet de 10 px,

Figure : fichier 'explosion.png'

On repère sur le croquis ci-dessous les dimensions des éléments à gérer :



Pour faciliter la lisibilité du code, on insère dès le lancement du jeu, une image transparente sur le coté inférieur de la fenêtre. Lorsque la balle rebondit sur ce côté, il suffit alors de modifier le fichier lié à cette image.

On commence par écrire le script de la fonction *creerImagesTk()* qui convertit les fichiers images *explosion.png* et *transparent.png* et les place dans le dictionnaire *dicImagesTk* crée dans le programme principal :

```
def creerImagesTk() :
    dicImagesTk["explosion.png"] = ImageTk.PhotoImage(Image.open("explosion.png") , master = fenetre)
    dicImagesTk["transparent.png"] = ImageTk.PhotoImage(Image.open("transparent.png") , master = fenetre)
```

On écrit ensuite le script de la fonction *insererImages()* qui insère une image lié au fichier *transparent.png* avec une ancre '*sw*' sur le point de coordonnées $(100\,,500)$. Le numéro d'insertion de cette image est placé dans la variable *explosion* qui est retournée dans le programme principal afin de pourvoir être accessible en lecture, dans toutes les autres fonctions.

```
def insererImages() :
    explosion = zone_graphique.create_image(100, 500 ,anchor = "sw",image = dicImagesTk["transparent.png"])
    return explosion
```

On crée le dictionnaire *dicImagesTk* et on appelle ces 2 fonctions dans le programme principal :

Les images étant à présent accessibles, on peut continuer.

```
# Variables globales ------------------------------
N = 0
dicImagesTk = {}
# Main --------------------------------------------
fenetre = creer_fenetre()
zone_graphique,score,bouton_depart = creer_widgets()
balle,palet = creer_graphisme()
creerImagesTk()
explosion = insererImages()

fenetre.bind("<Right>",droite)
fenetre.bind("<Left>",gauche)

fenetre.mainloop()
```

Pour ne pas surcharger le code de la fonction *animation_balle()*, on écrit le script de ce que l'on souhaite faire, dans une fonction nommée *rebond_coteInferieur()* :

```
def rebond_coteInferieur(xA,xB) :
    zone_graphique.itemconfigure(explosion, image = dicImagesTk["explosion.png"])
    xI = (xA + xB)//2 - 100
    zone_graphique.coords(explosion , xI , 500)

    xpA,ypA,xpB,ypB = zone_graphique.coords(palet)
    zone_graphique.coords(palet , xpA+5 ,ypA ,xpB-5 ,ypB)
```

On lie explosion au bon fichier

Abscisse du point de rebond

On raccourcit le palet de 10 px

On appelle cette nouvelle fonction dans *animation_balle()* :

```
def animation_balle(dx,dy):
    xA,yA,xB,yB = zone_graphique.coords(balle)

    zone_graphique.itemconfigure(explosion, image = dicImagesTk["transparent.png"])
    if yB + dy > 500 : rebond_coteInferieur(xA,xB)

    modifCouleur = False
    if xB + dx > 900 or xA + dx < 0 :
        dx = -dx
        modifCouleur = True
    if yB + dy > 500 or yA + dy < 0 :
        dy = -dy
        modifCouleur = True

    if modifCouleur == True :
        zone_graphique.itemconfigure(balle,fill=couleurAlea())
        zone_graphique.configure(bg = couleurAlea())

    xB,yB,dy = rebond_ballePallet(xA,yA,xB,yB,dy)

    zone_graphique.coords(balle , xA+dx ,yA+dy, xB+dx ,yB+dy)
    fenetre.after(50 ,  animation_balle , dx , dy)
```

On rend l'image explosion transparente

On appelle cette nouvelle fonction si la balle sort de la fenêtre par le bas

9. AUGMENTATION DE LA VITESSE :

Pour avoir une difficulté progressive au cours d'une partie, on propose d'augmenter la vitesse de la balle après chaque rebond sur le palet.

Pour simplifier l'ajout de cette fonctionnalité, on propose de créer une variable globale TEMPS initialisé à 50 ms dans le programme principal (variables globales de préférence en majuscule).

```
# Variables globales ------------------------------------------
N = 0
dicImagesTk = {}
TEMPS = 50
# Main --------------------------------------------------------
fenetre = creer_fenetre()
```

⇨ On déclare cette variable comme globale dans les fonctions *animation_balle()* et rebond_ballePallet() :

```
def animation_balle(dx,dy):
    global TEMPS
    xA,yA,xB,yB = zone_graphique.coords(balle)
```

```
def rebond_ballePallet(xA,yA,xB,yB,dy) :
    global N,TEMPS
    xAP,yAP,xBP,yBP = zone_graphique.coords(palet)
```

⇨ Dans la fonction *animation_balle(),* on remplace la durée de répétition qui était de 50, par cette nouvelle variable TEMPS.

```
fenetre.after(TEMPS ,  animation_balle , dx , dy)
```

⇨ Dans la fonction *rebond_ballePallet()*, on baisse de 5 % cette durée de répétition à chaque rebond :

```
def rebond_ballePallet(xA,yA,xB,yB,dy) :
    global N,TEMPS
    xAP,yAP,xBP,yBP = zone_graphique.coords(palet)
    xI = (xA + xB)//2
    if xAP < xI and xI < xBP and dy > 0 and yB+dy > yAP :
        dy = -dy
        xB = xB + 10
        yB = yB + 10
        N = N + 1
        score.configure(text = f"score = {N} ")
        PlaySound("bruit.wav", SND_ASYNC)
        TEMPS = int(TEMPS * 0.95)
    return xB,yB,dy
```

int() car TEMPS doit être un entier

10. GESTION DE LA FIN DE PARTIE :

Dans le scénario imaginé, la partie se termine lorsque la largeur du palet devient nulle. On se contente dans ce paragraphe de repérer cette issue et d'arrêter le jeu ensuite.

Pour pouvoir arrêter le mouvement de la balle en fin de partie, on crée une variable booléenne DRAPEAU dans le programme principal.

```
# Variables globales --------
N = 0
dicImagesTk = {}
TEMPS = 50
DRAPEAU = True
# Main ----------------------
fenetre = creer_fenetre()
```

⇨ Dans le fonction *animation_balle(),* on conditionne la répétition de la fonction à la valeur de cette variable DRAPEAU :

```
def animation_balle(dx,dy):
    global TEMPS,DRAPEAU
    xA,yA,xB,yB = zone_graphique.coords(balle)
    ...
    if DRAPEAU == True :fenetre.after(TEMPS ,  animation_balle , dx , dy)
```

⇨ Dans la fonction *rebond_coteInferieur()*, cette variable DRAPEAU prend la valeur *False* si la largeur du palet devient nulle :

```
def rebond_coteInferieur(xA,xB) :
    global DRAPEAU
    zone_graphique.itemconfigure(explosion, image = dicImagesTk["explosion.png"])
    xI = (xA + xB)//2 - 100
    zone_graphique.coords(explosion , xI , 500)

    xpA,ypA,xpB,ypB = zone_graphique.coords(palet)
    zone_graphique.coords(palet , xpA+5 ,ypA ,xpB-5 ,ypB)

    if xpB - xpA <= 0 :
        zone_graphique.coords(palet , 0 ,0 ,0 ,0)
        DRAPEAU = False
```

Le palet disparait car ses dimensions deviennent nulles, mais la variable *palet* existe toujours pour éviter toute erreur.

⇨ Tester ce script

11. FINAL UN PEU PLUS TRAVAILLE :

On donne ci-dessous le script d'une fonction nommée *final().* Elle permet d'améliorer le visuel de fin de jeu.

```
def final(n,texte) :
    n = n + 1
    xA,yA,xB,yB = zone_graphique.coords(balle)
    zone_graphique.coords(balle,xA-5,yA-5,xB+5,yB+5 )
    if n%10 == 0 :
        coul = couleurAlea()
        zone_graphique.itemconfigure(texte,font= 'Arial '+str(n), fill=coul)
    else :
        zone_graphique.itemconfigure(texte,font= 'Arial '+str(n))
    if n < 150 : fenetre.after(20,final, n,texte)
```

On récupère les coordonnées de la balle et augmente le diamètre de celle-ci de 20 px

On agrandit le texte et on change la couleur si n est un multiple de 10

On répète cette fonction 150 fois, toutes les 20 ms.

Cette fonction *final()* est appelée dans la fonction *rebond_coteInferieur()* qui gère la fin de partie.

```
def rebond_coteInferieur(xA,xB) :
    global DRAPEAU
    zone_graphique.itemconfigure(explosion, image = dicImagesTk["explosion.png"])
    xI = (xA + xB)//2 - 100
    zone_graphique.coords(explosion , xI , 500)

    xpA,ypA,xpB,ypB = zone_graphique.coords(palet)
    zone_graphique.coords(palet , xpA+5 ,ypA ,xpB-5 ,ypB)

    if xpB - xpA <= 140 :
        zone_graphique.coords(palet , 0 ,0 ,0 ,0)
        DRAPEAU = False
        texteFin = zone_graphique.create_text(450,250,anchor='center',text='FIN',font='Arial 1')
        final(0,texteFin)
```

On insère *final()* un texte de taille de font 1 px

On appelle la fonction *final()*

⇨ Uploader le fichier sur nsibranly.fr avec le code **tp13**

12. CONCLUSION :

On espère que la création de ce jeu vous a plu. Vous y avez écrit plus d'une centaine de lignes pythons. Dans le cadre du projet à venir, vous aurez à réaliser le même type de code pour créer une application que vous aurez imaginé personnellement.

Dans ce tp, il y a pas mal de mouvements, des rebonds, des collisions, … Tout cela se gère dans un premier temps sur des croquis faits à main levée. Un mouvement ce n'est simplement qu'une modification de coordonnées. Avant de commencer à coder, votre application doit être imaginé sur papier, à travers de nombreux croquis … . Plus votre projet sera travaillé au départ sur papier, moins vous perdrez de temps ensuite à l'écriture du code.

D'autre part, lorsque le nombre de lignes devient important, il est nécessaire de bien compartimenter les différentes actions à réaliser dans des fonctions. Le nom choisi pour les fonctions doit être explicite. Ne pas surcharger le script de ces fonctions : En créer de nouvelles si nécessaire. Il ne faut bien sûr pas exagérer dans l'autre sens. S'il y a trop de fonctions, cela devient contre-productif.