Jeu du serpent Poo : Tkinter

Règles du jeu : deux serpents constitués d'une tête et d'anneaux se déplacent sur la scène.

Lorsqu'ils arrivent sur un bord, ils réapparaissent de l'autre côté.

Les flèches haut, bas, à droite ou à gauche permettent de changer la direction du premier serpent.

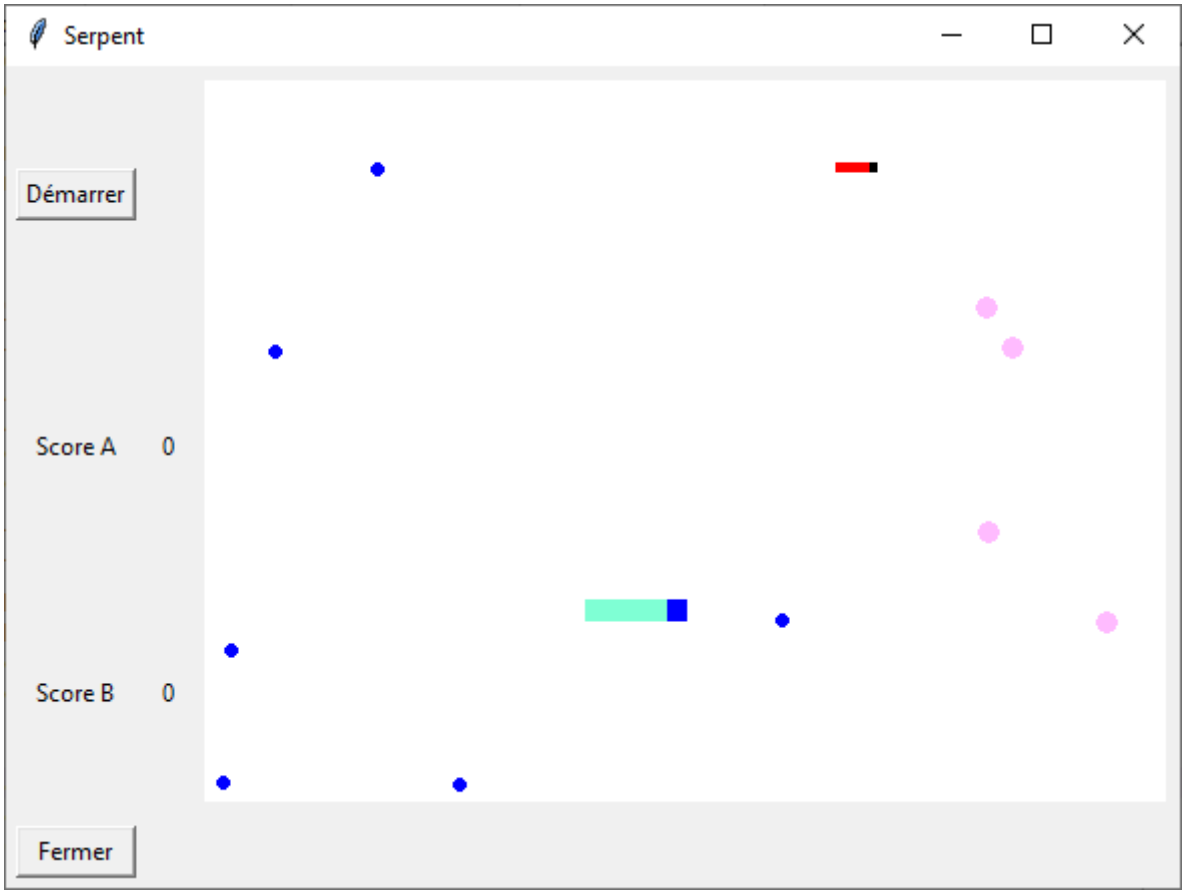
Les touches a, e, c ou w permettent de changer la direction du second serpent.

Sur la scène sont placées des pommes de différentes formes et couleurs. Lorsqu'un serpent mange une pomme, il marque un point si c'est une grosse pomme ou 3 points si c'est une petite pomme et son corps s'allonge.

Si le serpent se mord la queue, son score est diminué de 3 points.

Le jeu s'arrête lorsqu'on clique sur le bouton fin.

La partie démarre lorsqu'on clique sur le bouton démarrer.

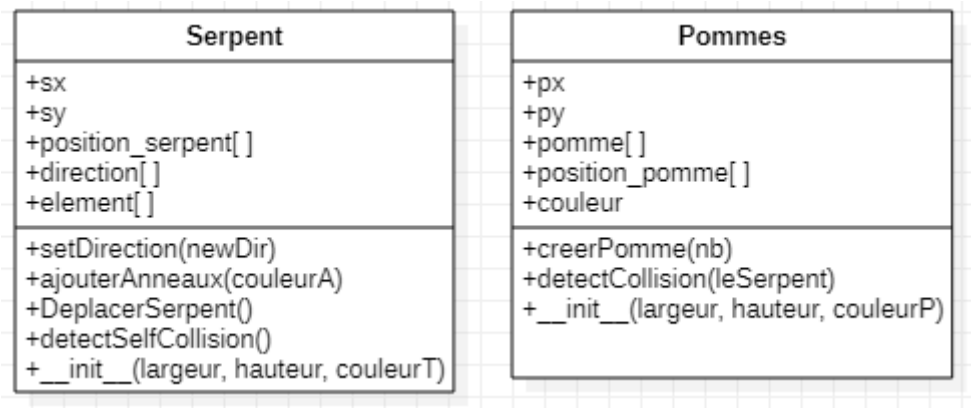


La programmation objet

Nous reprenons ici le même programme Tkinter, en structurant notre programme suivant la logique de la programmation objet.

Nous créons deux classes :

- La classe Serpent représente un serpent muni d'une tête et de x anneaux
- La classe Pommes représente un ensemble de pommes. Chaque ensemble est constitué de pommes de même taille et de même couleur. D'un ensemble à l'autre, la taille des pommes, leur couleur et leur nombre peuvent être différents.



La classe Serpent a 5 attributs d'instances, c'est-à-dire des attributs qui seront propres à chaque serpent créé avec cette classe.

- sx, sy : largeur et hauteur d'un élément du serpent (tête ou anneau)
- position_serpent[ ] : liste contenant la position de chaque élément du serpent
- direction [ ] : contient la direction dans laquelle la tête du serpent doit se déplacer
- element [ ] : liste contenant la référence à tous les éléments du serpent dessinés sur l'écran.

La classe Serpent possède 5 méthodes

- __init__ recevant 3 parametres : la largeur et la hauteur de la tête et des anneaux, ainsi que la couleur de la tête.
- setDirection : positionne la nouvelle direction de la tête du serpent
- ajouterAnneaux : ajoute 4 anneaux de couleurA au serpent
- DeplacerSerpent : déplace tous les éléments du serpent dans sa direction
- detectSelfCollision : détecte si le serpent se mord la queue.

La classe Pommes a 5 attributs d'instances, c'est-à-dire des attributs qui seront propres à chaque ensemble de pommes créé avec cette classe.

- px, py : largeur et hauteur d'une pomme de l'ensemble
- position_pomme [ ] : liste contenant la position de chaque pomme de l'ensemble
- couleur : des pommes de cet ensemble

- pomme [ ] : liste contenant la référence à toutes les pommes de cet ensemble dessinées sur l'écran.

La classe Pommes possède 3 méthodes

- __init__ recevant 3 parametres : la largeur et la hauteur des pommes, ainsi que la couleur de ces pommes.
- creerPomme : ajoute nb pommes à l'ensemble
- detectCollision : détecte une collision avec le serpent dont la référence est passée en paramètre.

Le code de la classe Serpent

```
class Serpent:
    def __init__(self,largeur, hauteur,couleurT):
        self.sx=largeur
        self.sy=hauteur
        self.direction=[1,0]
        self.position_serpent=[[randint(40,largeurFond-40),randint(40,hauteurFond-40)]]
        self.elements=[fond.create_rectangle(
                                self.position_serpent[0][0],
                                self.position_serpent[0][1],
                                self.position_serpent[0][0]+self.sx,
                                self.position_serpent[0][1]+self.sy,
                                fill=couleurT,
                                outline=couleurT)]
    def setDirection(self, newDir):
        self.direction=newDir

    def ajouterAnneaux(self,couleurA):
        #Position du dernier anneau
        x=self.position_serpent[len(self.position_serpent)-1][0]
        y=self.position_serpent[len(self.position_serpent)-1][1]
```

```
        #Suivant la direction les coordonnées des nouveaux anneaux changent
        dirx=self.direction[0]*self.sx
        diry=self.direction[1]*self.sy
        for i in range(4):
            x = x-dirx
            y = y-diry
            self.position_serpent.append([x,y])
            self.elements.append(fond.create_rectangle(
                                self.position_serpent[i+1][0],
                                self.position_serpent[i+1][1],
                                self.position_serpent[i+1][0]+self.sx,
                                self.position_serpent[i+1][1]+self.sy,
                                fill=couleurA,
                                outline=couleurA))

    def deplacerSerpent(self):
        x=self.position_serpent[0][0]
        dirx=self.direction[0]*self.sx
        y=self.position_serpent[0][1]
        diry=self.direction[1]*self.sy
        newPos = [x+dirx,y +diry]
        self.position_serpent.insert(0,newPos)
        self.position_serpent.pop()
        #Mise à jour du dessin sur l'écran
        i=0
        while i<len(self.position_serpent):
            fond.coords(self.elements[i],self.position_serpent[i][0],
                                self.position_serpent[i][1],
                                self.position_serpent[i][0]+self.sx,
                                self.position_serpent[i][1]+self.sy)
```

```
            i+=1

        #vérifier si la tête a touché un bord: le serpent doit ressortir de l'autre côté.
        x=self.position_serpent[0][0]
        y=self.position_serpent[0][1]
        if x <= 0:
            self.position_serpent[0][0]=480
        if x >= 480:
            self.position_serpent[0][0] = 0
        if y <= 0:
            self.position_serpent[0][1] = 360
        if y>= 360:
            self.position_serpent[0][1] = 0

    def detectSelfCollision(self):
        "Détection des collisions avec lui-même"
        xs=self.position_serpent[0][0]
        ys=self.position_serpent[0][1]
        i=3
        while i<len(self.position_serpent):
            xp=self.position_serpent[i][0]
            yp=self.position_serpent[i][1]
            if xs > xp+self.sx or xs+self.sx<xp or ys>yp+self.sy or ys+self.sy < yp:
                i+=1
            else:
                return (True)
        return (False)
```

Le code de la classe Pommes

```
class Pommes:
    def __init__(self,largeur, hauteur,couleurP):
        self.px=largeur
        self.py=hauteur
        self.pomme=[]
        self.position_pomme=[]
        self.couleur = couleurP

    def creerPomme(self, nb):
        for i in range(nb):
            xp = randint(self.px,largeurFond-self.px)
            yp = randint(self.py,hauteurFond-self.py)
            self.position_pomme.append([xp,yp])
            self.pomme.append(fond.create_oval(
                                self.position_pomme[i][0],
                                self.position_pomme[i][1],
                                self.position_pomme[i][0]+self.px,
                                self.position_pomme[i][1]+self.py,
                                fill=self.couleur,
                                outline=self.couleur))

    def detectCollision(self, leSerpent):
        "Détection des collisions avec le serpent"
        xs=leSerpent.position_serpent[0][0]
        ys=leSerpent.position_serpent[0][1]
        i=0
        while i<len(self.position_pomme):
            xp=self.position_pomme[i][0]
            yp=self.position_pomme[i][1]
```

```
            if xs > xp+self.px or xs+leSerpent.sx<xp or ys>yp+self.py or
ys+leSerpent.sy < yp:
                i+=1
            else:
                #si collision la pomme change de place
                xp=randint(6,largeurFond-self.px)
                yp=randint(6,hauteurFond-self.py)
                self.position_pomme[i]=[xp,yp]
                fond.coords(self.pomme[i],
                            self.position_pomme[i][0],
                            self.position_pomme[i][1],
                            self.position_pomme[i][0]+self.px,
                            self.position_pomme[i][1]+self.py,
                            )
                return (True)
        return (False)
```

La construction de l'interface et l'initialisation du jeu

```
#=================================================
    largeurFond=480
    hauteurFond=360
    scA=0
    scB=0
    #-----------------------------------------------
    ecran=tk.Tk()
    ecran.title("Serpent")
    #bouton pour lancer le jeu
    demarrer=tk.Button(ecran,text="Démarrer", command = debutPartie)
    demarrer.grid(row=0,column=0, padx = 5, pady = 5)
    #Labels Score : A pour le premier serpent et B pour le second
```

```
LabelScoreA= tk.Label(ecran, text="Score A" )
LabelScoreA.grid(row=1,column=0, padx = 5, pady = 5)
textA=tk.StringVar()
textA.set("0")
ScoreA= tk.Label(ecran, textvariable=textA )
ScoreA.grid(row=1,column=1, padx = 5, pady = 5)
LabelScoreB= tk.Label(ecran, text="Score B" )
LabelScoreB.grid(row=2,column=0, padx = 5, pady = 5)
textB=tk.StringVar()
textB.set("0")
ScoreB= tk.Label(ecran, textvariable=textB )
ScoreB.grid(row=2,column=1, padx = 5, pady = 5)
#bouton pour fermer la fenêtre
finPartie=tk.Button(ecran,text="Fermer",command = finPartie)
finPartie.grid(row=3,column=0,sticky='ew', padx = 5, pady = 5)
#un canvas pour notre jeu
fond = tk.Canvas(ecran, width=largeurFond,height=hauteurFond,bg="white")
fond.grid(row=0,column=2, rowspan=3, padx = 5, pady = 5)
#----------------------------------------
#Création des serpents : le premier mince avec tête noire et anneaux rouge
serpent1=Serpent(4,4,"black")
serpent1.ajouterAnneaux("red")
#le second serpent épais, avec tête bleue et anneaux verts
serpent2=Serpent(10,10,"blue")
serpent2.ajouterAnneaux("aquamarine")
#créer des ensembles de pommes : 6 petites pommes bleues
lesPommes1=Pommes(6,6,"blue")
lesPommes1.creerPomme(6)
#4 grosses pommes violettes
lesPommes2=Pommes(10,10,"plum1")
```

```
lesPommes2.creerPomme(4)
#----------------------------------------
#Enregistrement des touches que le jeu doit utiliser
#pour guider le premier serpent
ecran.bind("<Up>",serpent_up)
ecran.bind("<Down>",serpent_down)
ecran.bind("<Right>",serpent_right)
ecran.bind("<Left>",serpent_left)
#pour guider le deuxième serpent
ecran.bind("<a>",serpent_a)
ecran.bind("<e>",serpent_e)
ecran.bind("<c>",serpent_c)
ecran.bind("<w>",serpent_w)
ecran.mainloop()
```

Les fonctions Call back

```
def serpent_up(event):
    serpent1.setDirection([0,-1])
def serpent_down(event):
    serpent1.setDirection([0,1])
def serpent_right(event):
    serpent1.setDirection([1,0])
def serpent_left(event):
    serpent1.setDirection([-1,0])
def serpent_a(event):
    serpent2.setDirection([0,-1])
def serpent_e(event):
    serpent2.setDirection([0,1])
def serpent_c(event):
    serpent2.setDirection([1,0])
```

```
    def serpent_w(event):
        serpent2.setDirection([-1,0])

    def debutPartie():
        gestionSerpent()

    def finPartie() :
        ecran.destroy()
```

La fonction principale du jeu

```
#-----------------------------------------------------
    def gestionSerpent():
        global scA, scB
        # déplacer les serpents
        serpent1.deplacerSerpent()
        serpent2.deplacerSerpent()
        #tester collision serpent1-pommes1
        test= lesPommes1.detectCollision(serpent1)
        if test:
            serpent1.ajouterAnneaux("red")
            scA+=3
            textA.set(str(scA))
        #tester collision serpent2-pommes1
        test= lesPommes1.detectCollision(serpent2)
        if test:
            serpent2.ajouterAnneaux("aquamarine")
            scB+=3
            textB.set(str(scB))
```

```
#tester collision serpent1-pommes2
test= lesPommes2.detectCollision(serpent1)
if test:
    serpent1.ajouterAnneaux("red")
    scA+=1
    textA.set(str(scA))
#tester collision serpent2-pommes2
test= lesPommes2.detectCollision(serpent2)
if test:
    serpent2.ajouterAnneaux("aquamarine")
    scB+=1
    textB.set(str(scB))
#tester si les serpents se mordent la queue
test = serpent1.detectSelfCollision()
if test:
    scA-=3
    textA.set(str(scA))
test = serpent2.detectSelfCollision()
if test:
    scB-=3
    textB.set(str(scB))

fond.after(50,gestionSerpent)
```