

# Projet n°12

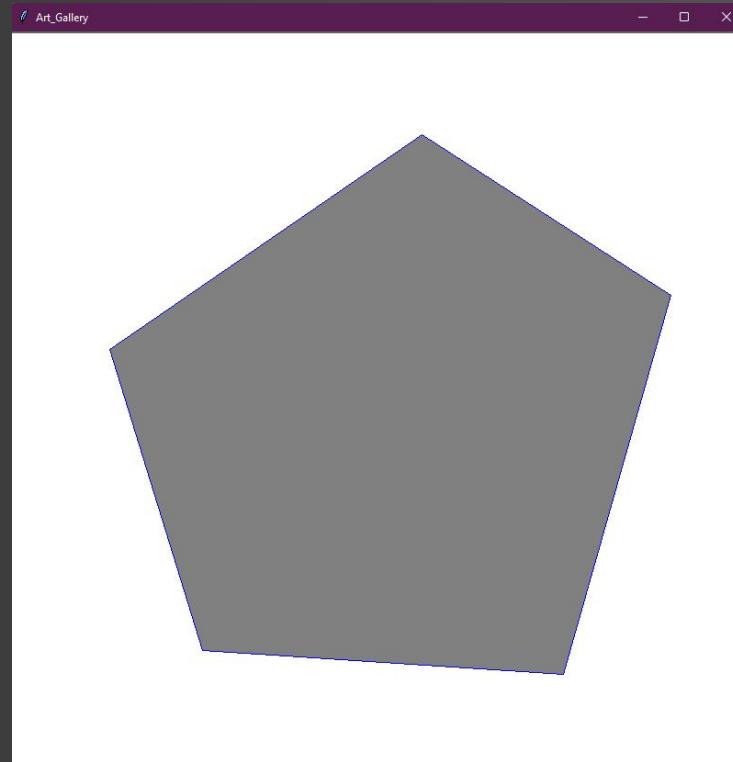
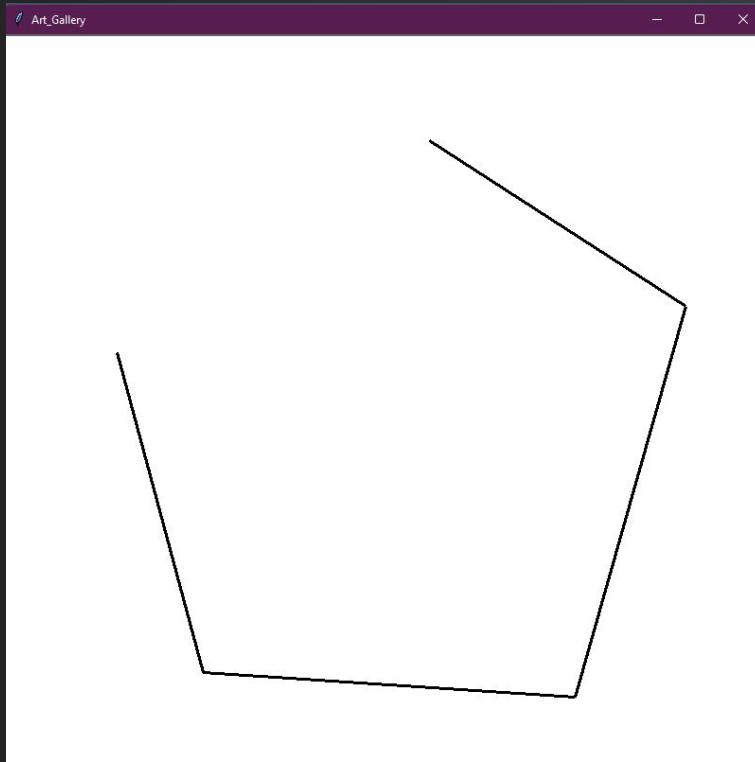
# Galerie d'Art

## 3 méthodes:

- Détection Overlapping
- Coins du polynômes
- Barycentres



# Création polygone



# Détection emplacement point

Vérification de l'emplacement du gardien

```
carreselect=len(cnva.find_overlapping(x0-5,y0-5,x0+5,y0+5)) #mesure sur le nombre d'élément sous le point
print(carreselect)
if carreselect == 0 : #Détermination si point est à l'intérieur du polygone ou à l'extérieur
    '''si la variable carreselect==0 il n'y a aucun d'élément à l'emplacement du clic donc le clic est à l'extérieur du polygone dessiné'''
    print('Placer un point à l intérieur du polygone')
    cnva.itemconfigure('effacecarre', state="normal",fill='black')#Gestion de tag pour" faire faire raparaître ancien rayon"
    cnva.itemconfigure('effaceligne', state="normal") #Gestion de tag pour" faire faire raparaître ancien rayon"
```

# Code création polygone

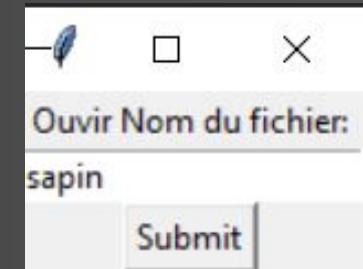
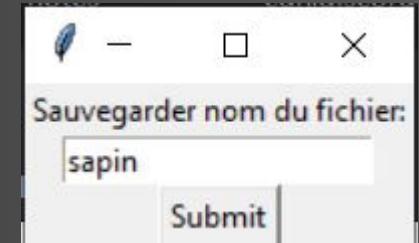
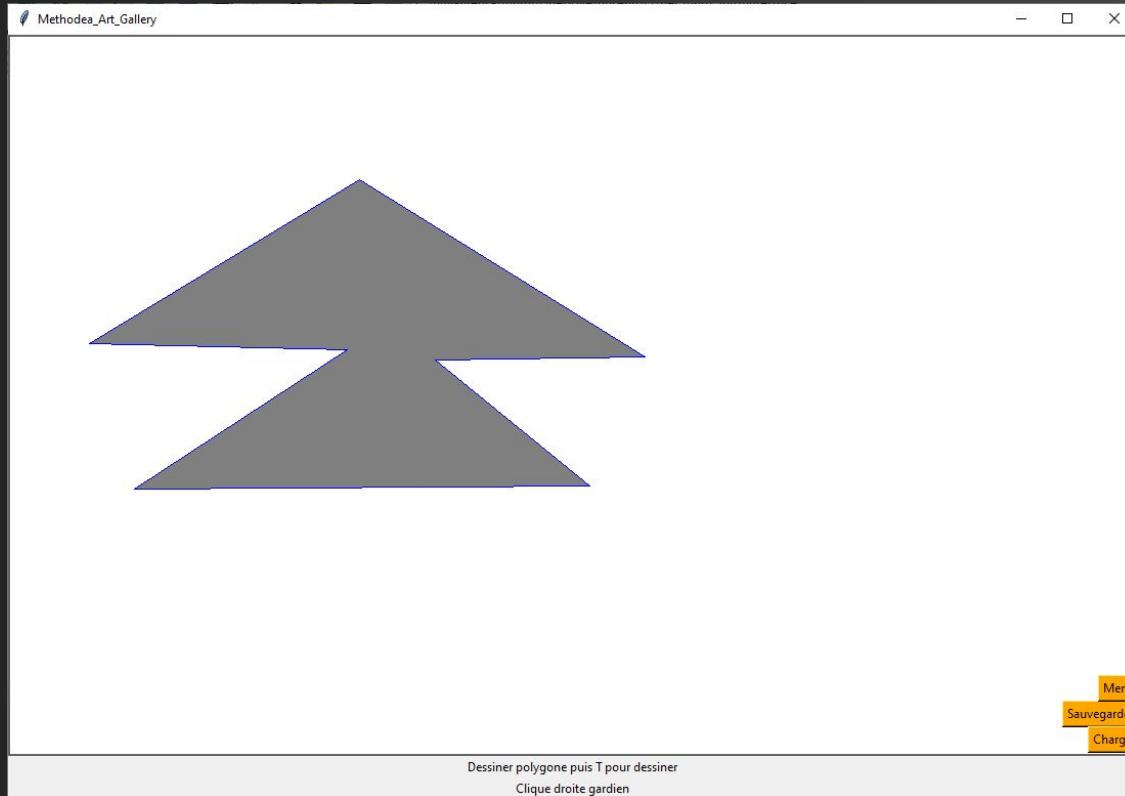
```
def creer_polya(event):
    """Gestion d'un clic : créer un rectangle réduit au point du clic."""
    global x0, y0, id_rect,xdepart,ydepart
    x0, y0 = event.x, event.y
    Lsave.append([x0,y0])
    id_rect = cnva.create_line(x0, y0, x0, y0,fill='black', width=3)
    cnva.bind('<Motion>', redessiner_polya)
    """cnv.focus_set()"""
    cnva.bind('<t>', fixer_polya)

    def redessiner_polya(event):
        """Gestion des mouvements de la souris : redessiner le rectangle."""
        cnva.coords(id_rect, x0, y0, event.x, event.y)
        coordonne=x0,y0
        L.append(coordonne)

#--Quand l'utilisateur a finit de tracer par points son polygone il appuie sur la touche 'T'-----
def fixer_polya(event):
    """Au relâchement du bouton arrêter de suivre les mouvements."""
    print("Le polygone est dessiné")

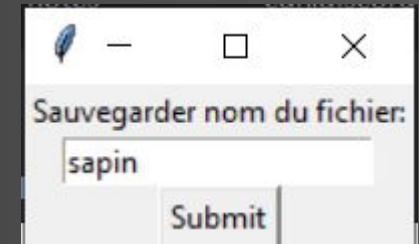
    xdepart,ydepart=L[0] #récupération du premier point
    cnva.create_line(event.x, event.y, xdepart,ydepart,fill='black', width=3) #création de la dernière ligne entre le point de départ et le dernier point
    cnva.unbind('<Motion>')
    Lsave.append([event.x,event.y])
    cnva.delete('all')
    cnva.create_polygon(Lsave, fill='grey', width=epaisseur, outline='blue')
    del L[:]
```

# Sauvegarder/Charger polygone



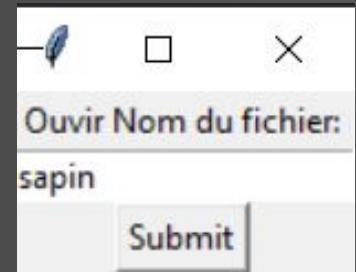
# Sauvegarder polygone

```
def sauvegardePointbutton():
    global fenetrePrincipale, e
    fenetrePrincipale = tk.Tk()
    l = tk.Label(fenetrePrincipale, text = "Sauvegarder nom du fichier:")
    e = tk.Entry(fenetrePrincipale)
    b = tk.Button(fenetrePrincipale ,text="Submit", command=Save)
    l.pack()
    e.pack()
    b.pack()
    fenetrePrincipale.mainloop()
```

A screenshot of a Windows Notepad window titled "sapin.txt - Bloc-notes". The content is a list of coordinate pairs separated by spaces:  
531 159  
226 392  
524 368  
323 508  
896 508  
670 369  
951 364

```
def Save():
    global Lsave
    print(Lsave)
    fichier = e.get()
    file = open(f"{fichier}.txt", "w")
    for i in Lsave:
        print(f"{i[0]} {i[1]}")
        file.write(f"{i[0]} {i[1]} \n")
    file.close()
    fenetrePrincipale.destroy()
```

# Charger polygone



```
def loadbuttona():
    global fenetrePrincipale, e
    fenetrePrincipale = tk.Tk()
    l = tk.Label(fenetrePrincipale, text = "Ouvir Nom du fichier:")
    e = tk.Entry(fenetrePrincipale)
    b = tk.Button(fenetrePrincipale ,text="Submit", command=Drawa)
    l.pack()
    e.pack()
    b.pack()
    fenetrePrincipale.mainloop()
```

```
#-----Création du polygone enregistré-----
def Drawa():
    global Lsave
    del Lsave[:]
    fichier=e.get()

    Lsave=RecuperationPoint(fichier)#Recupération des coordonnées des cotées dans le fichier enregistré

    cnva.delete('all') #Supprime les éléments déjà sur la fenêtre
    cnva.create_polygon(Lsave, fill='grey', width=epaisseur, outline='blue') #Création du polygone
    fenetrePrincipale.destroy()
```

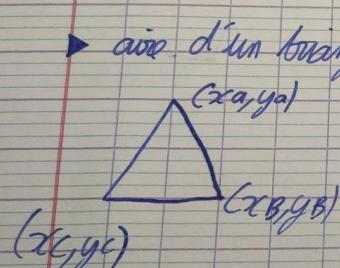
# Fonction Reset

```
#--Fonction Reset -----
def reseta(event):

    cnva.unbind('<Motion>')
    cnva.delete('all') #Supprimer les éléments de la fenêtre et tout les éléments dans les listes
    del L[:]
    del Lsave[ :]
    del listeevent[ :]
    del Lcote[ :]
```

# Démonstration

► aire d'un triangle avec celui d'un déterminant:



$$\begin{vmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} x_A - x_C & x_B - x_C & x_C \\ y_A - y_C & y_B - y_C & y_C \\ 0 & 0 & 1 \end{vmatrix}$$

$$\det(ABC) = (x_A - x_C)(y_B - y_C) - (x_B - x_C)(y_A - y_C)$$

$$= x_A \cdot y_B - x_A \cdot y_C - x_C \cdot y_B + x_C \cdot y_C - x_B \cdot y_A + x_B \cdot y_C + x_C \cdot y_A - x_C \cdot y_B$$

On lie l'aire d'un triangle avec celui d'un déterminant:

$$\text{Aire}(ABC) = \frac{1}{2} * \det(ABC)$$

$$(x_A + x_B)(y_B - y_A) + (x_B + x_C)(y_C - y_B) + (x_C + x_A)(y_A - y_C) = (x_A y_B - x_B y_A) + (x_B y_C - x_C y_B) + (x_C y_A - x_A y_C)$$

$$\left[ \begin{array}{l} (1) (x_A + x_B)(y_B - y_A) + (x_B + x_C)(y_C - y_B) + (x_C + x_A)(y_C - y_A) = \\ (2) (x_A y_B - x_B y_A) + (x_B y_C - x_C y_B) + (x_C y_A - x_A y_C) \end{array} \right]$$

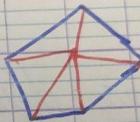
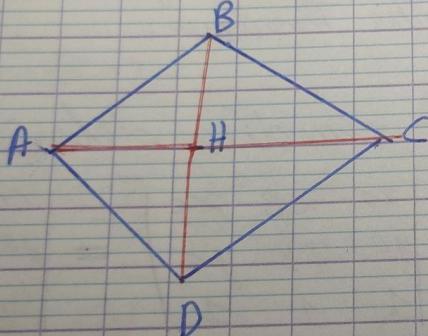
$$(2) \quad \underbrace{(x_A + x_B)(y_B - y_A)}_{\text{I}} + \underbrace{(x_B + x_C)(y_C - y_B)}_{\text{II}} + \underbrace{(x_C + x_A)(y_C - y_A)}_{\text{III}}$$

on retrouve bien (1) en développant:

$$\begin{aligned} & x_A y_B - x_B y_A + x_B y_B - x_B y_A + x_B y_C - x_B y_B + x_C y_C - x_C y_B \\ & + x_C y_A - x_C y_C + x_B y_A - x_A y_C = (1) \end{aligned}$$

$$(x_A + x_B)(y_B - y_A) + (x_B + x_C)(y_C - y_B) + (x_C + x_A)(y_C - y_A) = (x_A y_B - x_B y_A) + (x_B y_C - x_C y_B) + (x_C y_A - x_A y_C)$$

► quadrilatère convexe (avec angle saillant  $\angle ABCD$ )



Premier triangle  $[ABH]$ :  $(x_A \cdot y_B - x_B \cdot y_A) + (x_B \cdot y_H - x_H \cdot y_B)$   
 $(x_H \cdot y_A - x_A \cdot y_H)$

Deuxième triangle  $[BCH]$ :  $(x_B \cdot y_C - x_C \cdot y_B) + (x_C \cdot y_H - x_H \cdot y_C)$   
 $(x_H \cdot y_B - x_B \cdot y_H)$

Troisième triangle  $[DCH]$ :  $(x_C \cdot y_D - x_D \cdot y_C) + (x_D \cdot y_H - x_H \cdot y_D)$   
 $(x_H \cdot y_C - x_C \cdot y_H)$

Quatrième triangle  $[DAH]$ :  $(x_D \cdot y_A - x_A \cdot y_D) + (x_A \cdot y_H - x_H \cdot y_A)$   
 $(x_H \cdot y_D - x_D \cdot y_H)$

Bisector triangle  $[ABH]$ :  $(x_A \cdot y_B - x_B \cdot y_A) + (x_B \cdot y_H - x_H \cdot y_B)$   
 $(x_H \cdot y_A - x_A \cdot y_H)$

Decvisor triangle  $[BCH]$ :  $(x_B \cdot y_C - x_C \cdot y_B) + (x_C \cdot y_H - x_H \cdot y_C) + (x_H \cdot y_B - x_B \cdot y_H)$

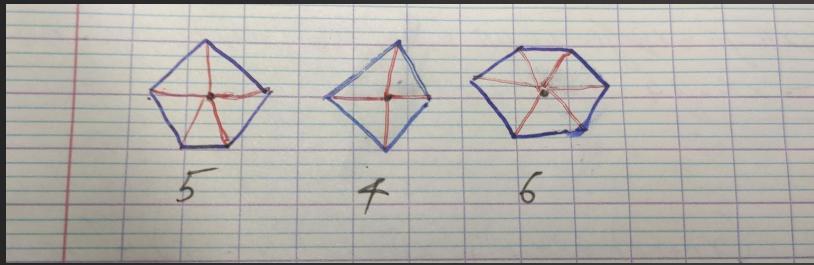
Torsor triangle  $[CDH]$ :  $(x_C \cdot y_D - x_D \cdot y_C) + (x_D \cdot y_H - x_H \cdot y_D) + (x_H \cdot y_C - x_C \cdot y_H)$

Quatrivisor triangle  $[DAH]$ :  $(x_D \cdot y_A - y_A \cdot y_D) + (x_A \cdot y_H - x_H \cdot y_A) + (x_H \cdot y_D - x_D \cdot y_H)$

On appelle le trait

axis  $[ABCD]$ :  $(x_A \cdot y_B - x_B \cdot y_A) + (x_B \cdot y_C - x_C \cdot y_B) + (x_C \cdot y_D - x_D \cdot y_C)$   
 $= (x_D \cdot y_A - y_A \cdot y_D)$

# De manière générale



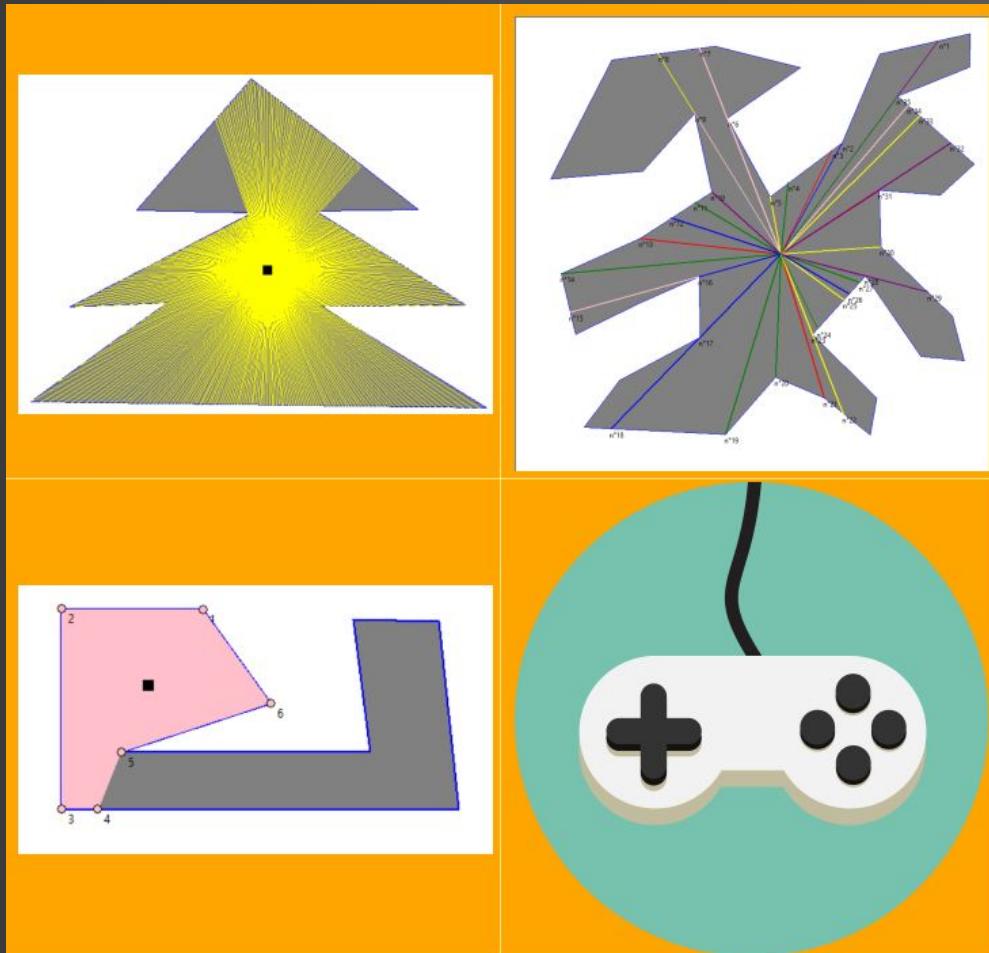
$$\frac{1}{2} \sum_{i=0}^{n-1} (x_i + x_{i+1})(y_{i+1} - y_i) = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)$$

# Calcul de l'aire

```
#Méthode 1 par l'appui de la touche B
def airepolygomméthode1(event):
    global Lsave
    print(Lsave)
    duree=0
    somme=0
    second1 = time.time()
    for k in range(len(Lsave)):
        xi,yi=Lsave[k-1]
        yi=hauteur-yi
        xil,yil=Lsave[k]
        yil=hauteur-yil
        somme+=(xi+xil)*(yil-yi)
    somme=abs((0.5)*somme)
    print('l aire du polygon est:')
    print(somme)
    second2 = time.time()
    duree=second2-second1
    print("Temps d'exécution méthode 1:")
    print(duree)
    print(second1)
    print(second2)
```

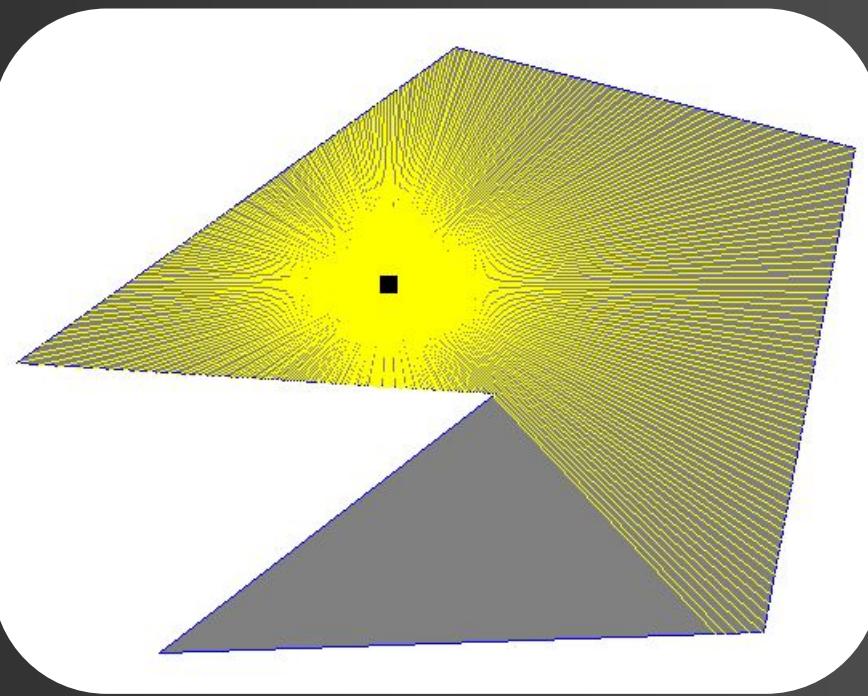
```
#Méthode 2 par l'appui de la touche N
def airepolygomméthode2(event):
    global Lsave
    print(Lsave)
    duree=0
    somme=0
    second1 = time.time()
    for k in range(len(Lsave)):
        xi,yi=Lsave[k-1]
        yi=hauteur-yi
        xil,yil=Lsave[k]
        yil=hauteur-yil
        somme+=(xi*yil-xil*yi)
    somme=abs((0.5)*somme)
    print('l aire du polygon est:')
    print(somme)
    second2 = time.time()
    duree=second2-second1
    print("Temps d'exécution méthode 2:")
    print(duree)
    print(second1)
    print(second2)
```

# Méthode A



# Méthode A

Partie création du polygone et le gardien par plusieurs droites suivants 360 degrés



# Détection des points de collisions

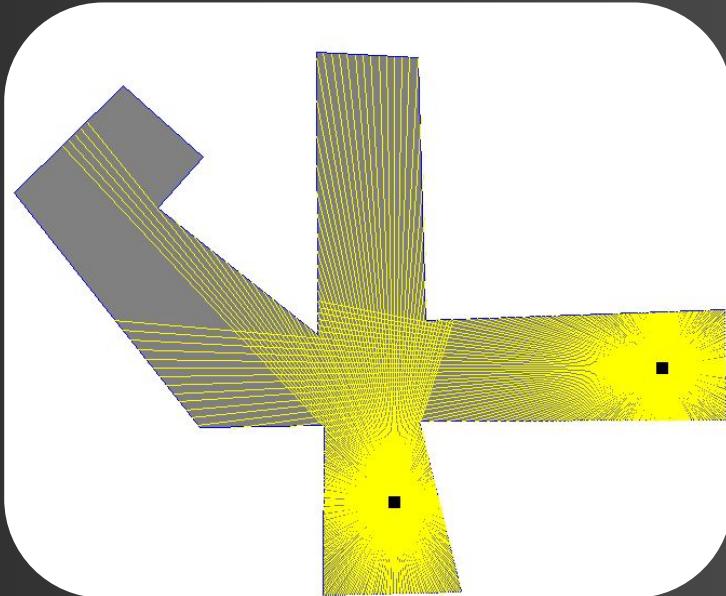
```
C=[] #liste contenant tout les coordonées des points de la bordure des rayons
for k in range (360): # Pour faire un tour
    x=math.radians(k) #On convertie en radians pour l'utilisation de cosinus et sinus
    a=math.cos(x)
    b=math.sin(x)
    R=1
    collision=len(cnva.find_overlapping(x0+R*a-1,y0+R*b-1,x0+R*a,y0+R*b)) #mesure sur le nombre d'élément sous le point
    while collision==1: #On déplace le point suivant le rayon jusqu'à ce qu'il sorte du polygone
        R+=1
        collision=len(cnva.find_overlapping(x0+R*a-1,y0+R*b-1,x0+R*a,y0+R*b))

    pointx=x0+(R-epaisseur)*a #position du point suivant x - la taille de la bordure
    pointy=y0+(R-epaisseur)*b #position du point suivant y - la taille de la bordure
    coordonne=pointx,pointy
    C.append(coordonne) #on l'ajoute dans notre liste contenant tout les coordonées des points de la bordure des rayons
```

# Plusieurs gardiens

- Tags

```
cnva.itemconfigure('effaceligne', state="hidden") #Gestion de tag pour "faire disparaître ancien rayon"  
cnva.itemconfigure('effacecarre', state="hidden") #Gestion de tag pour "cacher ancien carré"
```

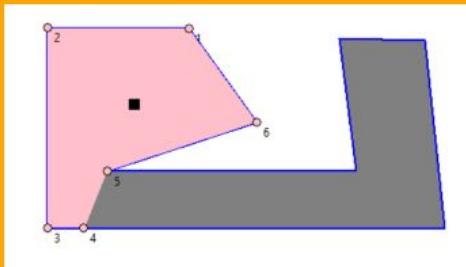
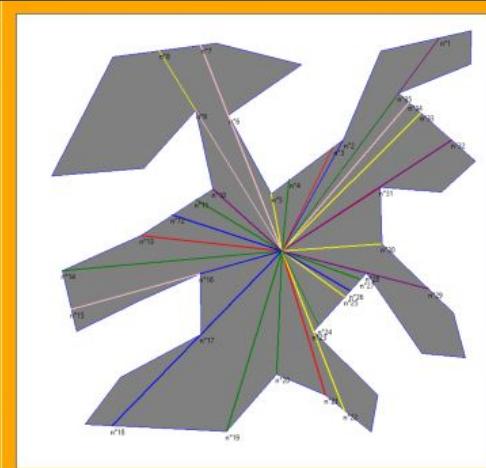
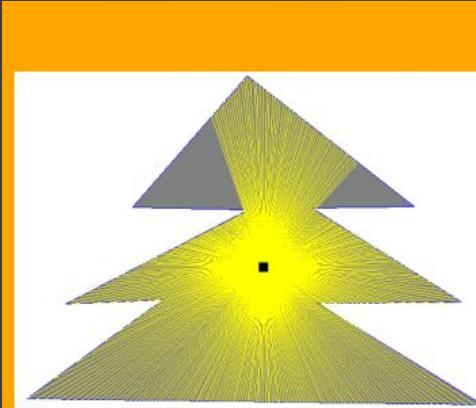


# Affichage champs de vision

```
for k in range (360): #On dessine ici tout les rayons avec les positions enregistrées dans la liste C
    pointx,pointy=C[k]
    cnva.create_line(x0, y0,pointx,pointy, fill='yellow', width=1, tag='effaceligne') #création d'un rayon avec deux coordonnées
    cnva.itemconfigure('effaceligne', state="normal") #Gestion de tag pour" faire faire raparaître ancien rayon"

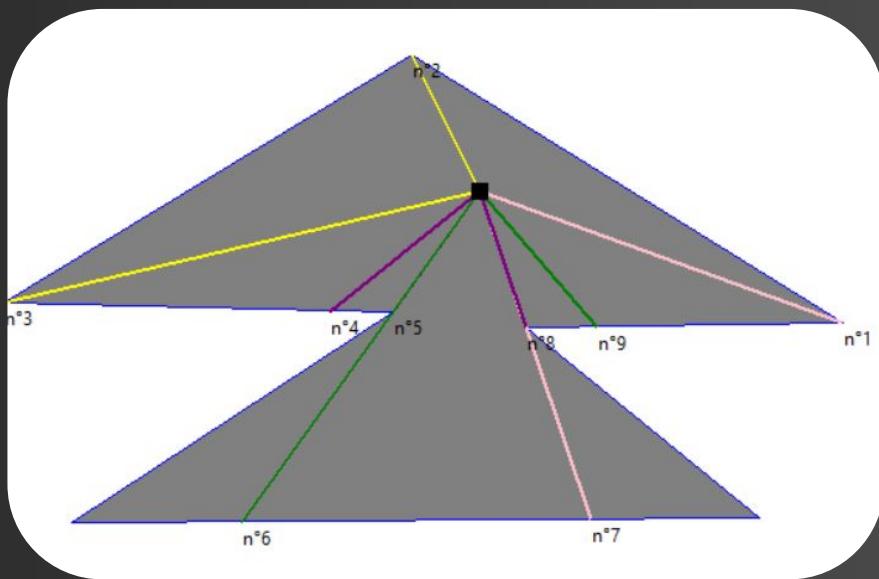
for k in range (len(listeevent)):#On dessine ici tout les carrés de gardiant event mémoriser en premier plan
    x0,y0=listeevent[k]
    cnva.create_rectangle(x0-5, y0-5,x0+5,y0+5, fill='black', tag='effacecarre') #creation d'un carré noir metant en lumière la position du clic soit du gardiant
```

# Méthode B



# Méthode B

Partie création du polygone et du gardien suivant les points des côtés du polygone avec la méthode overlapping



# Fonctions utilisés

```
#Fonction qui permet de déterminer si un point (x,y)
#est compris sur la droite de points (x0,y0) et ((x1,y1))

def equation(x,y,x0,y0,x1,y1):
    if (x1-x0)==0:
        if y1-y0>0:
            if y>y0 and y<y1:
                return True
            else:
                return False
        else:
            if y>y1 and y<y0:
                return True
            else:
                return False

    if (abs(y-((y1-y0)/(x1-x0))*x+y0-((y1-y0)/(x1-x0))*x0))<=20:
        return True
    else:
        print('supprimer')
        print(x)
        print(y)
    return False
```

```
def verification(x,y,xl,yl,xa,ya):
    # xl<x<xa et yl<y<ya on vérifie si le point x,y est compris entre les deux extrémités d'une droite

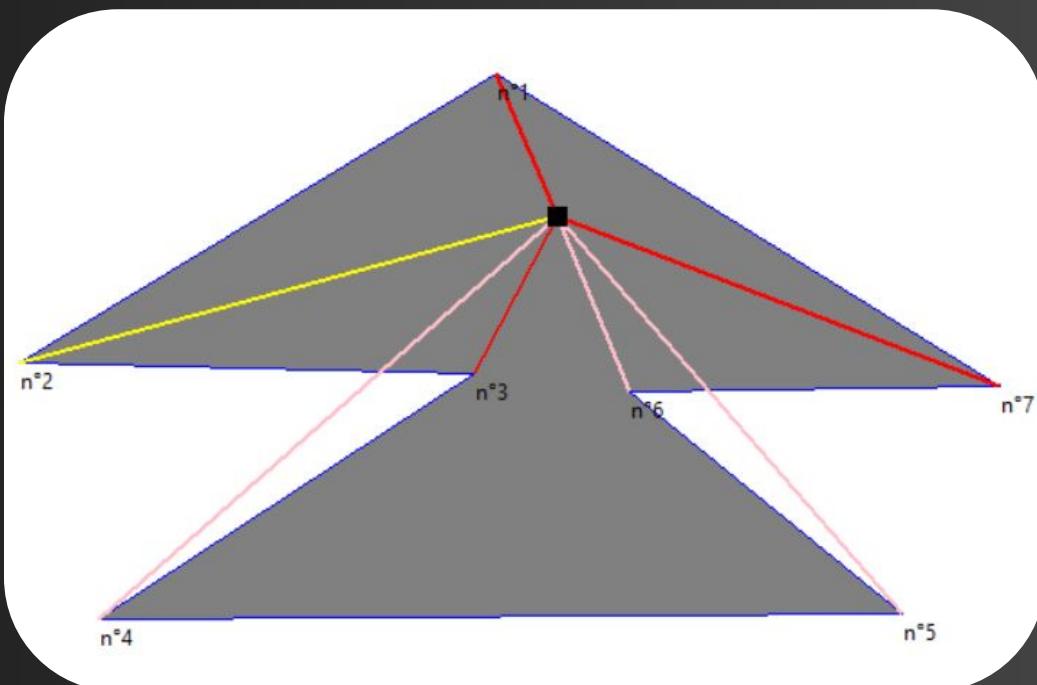
    if (xl>xa):
        xmax=xl
        xmin=xa
    else:
        xmax=xa
        xmin=xl

    if (yl>ya):
        ymax=yl
        ymin=ya
    else:
        ymax=ya
        ymin=yl

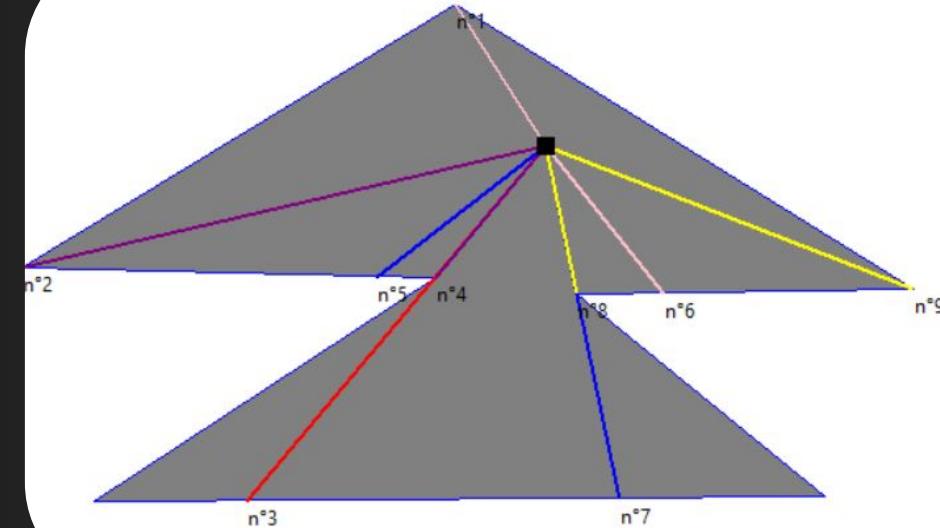
    if (xmin-x<=1 and x-xmax<=1 and ymin-y<=1 and y-ymax<=1):
        return True
    else:
        return False
```

```
def distance(x1,y1,x2,y2): #Fonction qui permet le calcul d'un module avec deux points en paramètre
    dist = math.sqrt((x2-x1)**2 + (y2-y1)**2)
    return dist
```

# Détection des points de collisions



Numérotation des côtés du polygone (liste Lsave)



```

for k in range (len(Lsave)):
    pointx,pointy=Lsave[k]
    angle=math.atan2(-y0+pointy,-x0+pointx)
    a=math.cos(angle)
    b=math.sin(angle)
    R=1
    collision=len(cnvb.find_overlapping(x0+R*a-1,y0+R*b-1,x0+R*a,y0+R*b))

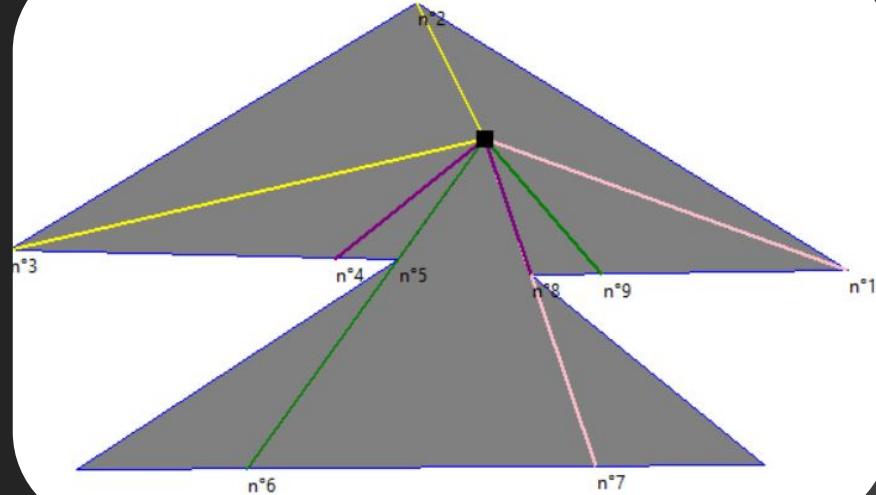
    while collision==1:
        R+=1
        collision=len(cnvb.find_overlapping(x0+R*a-1,y0+R*b-1,x0+R*a,y0+R*b))

        newpointx=int(x0+R*(a) - a*epaisseur)
        newpointy=int(y0+R*(b) - b*epaisseur)
        coordonne=newpointx,newpointy
        C.append(coordonne)

        if (distance(x0,y0,pointx,pointy)-distance(x0,y0,newpointx,newpointy))<-5:
            coordonnee=pointx,pointy
            C.append(coordonnee)

C=reemplace(Lsave,C)
print(' ')
print("Coordonnees de tous les points sans tri")
print(C)

```



```
for i in range (-1,len(Lsave)-1):
    xa,ya=Lsave[i+1]
    x1,y1=Lsave[i]
    for j in range (len(C)):
        x,y=C[j]
        if equation(x,y,x1,y1,xa,ya)==True and verification(x,y,x1,y1,xa,ya)==True:
            P.append([x,y])
```

```
stop=1
if (len(P)>1):
    time=1
    for j in range (len(P)):
        Px,Py=P[j]
        Lsavex,Lsavey=Lsave[i]
        if Px==Lsavex and Py==Lsavey and time==1:
            valeurx,valeury=P[0]
            P[0]=[Lsavex,Lsavey]
            P[j]=[valeurx,valeury]
            time=0
            print("1")
```

```
[P1.append(x) for x in P if x not in P1]
del P[:]
```

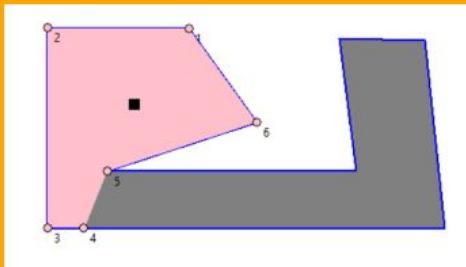
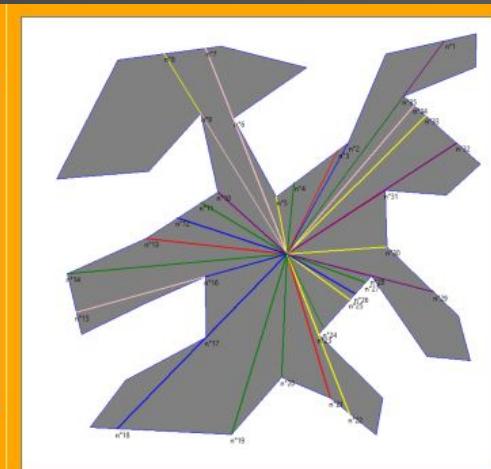
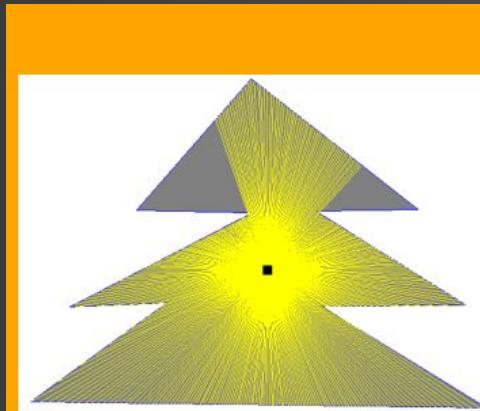
```
while(stop==1):
    stop=0
    for m in range (len(P)-1):
        point1x,point1y=P[m]
        pointx,pointy=Lsave[i]
        point2x,point2y=P[m+1]
        if distance(pointx,pointy,point1x,point1y)>distance(pointx,pointy,point2x,point2y):
            a=P[m]
            P[m]=P[m+1]
            P[m+1]=a
            stop=1
            print("trié croissant")
            print(P)
```

# Affichage droite numéroté

```
print("Ordre affiché Coordonne tout les points avec deuxieme tri sans doublons")
print(P1)
compteurbis=0
for k in range(len(P1)):
    pointx,pointy=P1[k]
    compteurbis+=1
    couleur=random.choice(color)
    cnvb.create_line(x0, y0,pointx,pointy, fill=couleur, width=2, tag='effaceligne')#création d'un rayon avec deux coordonnées
    cnvb.create_text(pointx+10, pointy+10, text= f'n°{compteurbis}', tag='effaceligne')
```

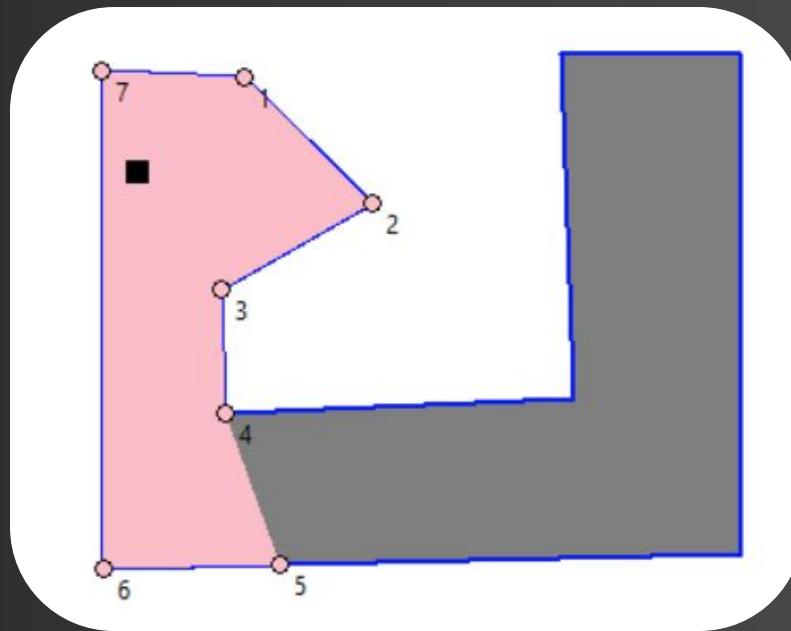
#creation d'un carré noir mettant en lumière la position du clic soit du gardien  
cnvb.create\_rectangle(x0-5, y0-5,x0+5,y0+5, fill='black', tag='effacecarre')

# Méthode C

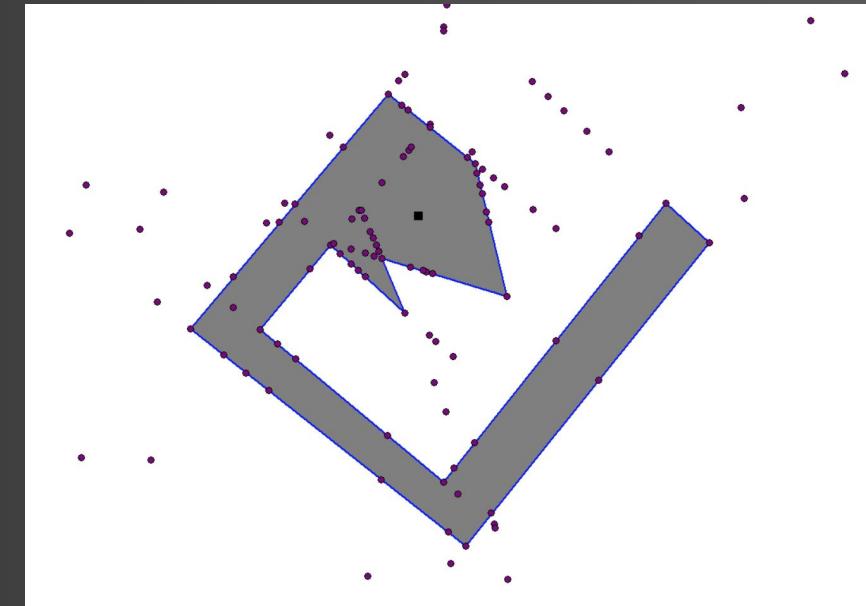


# Méthode C

Partie création du polygone et du gardien suivant les points des côtés du polygone en utilisant les barycentres

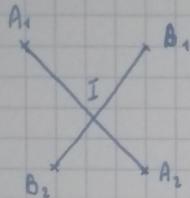


```
Lsave.append(Lsave[0])
for j in range (len(Lsave)):
    for i in range (len(Lsave)-1):
        #print(empGar)
        I= Barycentre(empGar,Lsave[j],Lsave[i],Lsave[i+1])
        pointBar=I.barycentre()
```



# Système d'équation

1)

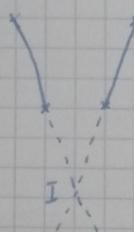


$$A_1(x_{A_1}, y_{A_1})$$

$$A_2(x_{A_2}, y_{A_2})$$

$$B_1(x_{B_1}, y_{B_1})$$

$$B_2(x_{B_2}, y_{B_2})$$



$$y_1(t) = A_1 A_2 = \frac{y_{A_2} - y_{A_1}}{x_{A_1} - x_{A_2}} t + y_{A_1} - \frac{y_{A_1} - y_{A_2}}{x_{A_1} - x_{A_2}} x_{A_1}$$

$$y_2(t) = B_1 B_2 = \frac{y_{B_2} - y_{B_1}}{x_{B_1} - x_{B_2}} t + y_{B_1} - \frac{y_{B_1} - y_{B_2}}{x_{B_1} - x_{B_2}} x_{B_1}$$

$$A_1 A_2 = B_1 B_2$$

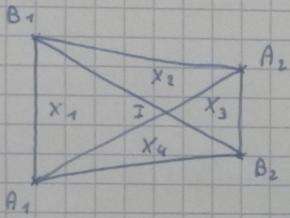
$$\Leftrightarrow a_1 t + y_{A_1} - a_1 x_{A_1} = a_2 t + y_{B_1} - a_2 x_{B_1}$$

$$\Leftrightarrow t = \frac{(y_{B_1} - y_{A_1}) + (a_1 x_{A_1} - a_2 x_{B_1})}{a_1 - a_2}$$

Preuve Barycentre 2):



$$\text{M Barycentre de } ((A, \alpha), (B, \beta), (C, \gamma))$$
$$(\alpha + \beta + \gamma) M = \alpha A + \beta B + \gamma C$$



$$(x_1 + x_2) I = x_1 A_2 + x_2 A_1 \quad (1)$$

$$(x_3 + x_4) I = x_3 A_1 + x_4 A_2 \quad (2)$$

$|B_1B_2A_2| + |B_2B_1A_1| \rightarrow$  Aire du polygone  $B_1A_2B_2A_1$

$$(1) + (2) : (x_1 + x_2 + x_3 + x_4) I = (x_1 + x_4) A_2 + (x_2 + x_3) A_1$$

$$(x_1 + x_2 + x_3 + x_4) = |B_1B_2A_2| + |B_2B_1A_1|$$

$$(x_1 + x_4) = |B_2B_1A_1|$$

$$(x_2 + x_3) = |B_1B_2A_2|$$

$$I = \frac{|B_1B_2A_2| A_1 + |B_2B_1A_1| A_2}{|B_1B_2A_2| + |B_2B_1A_1|}$$

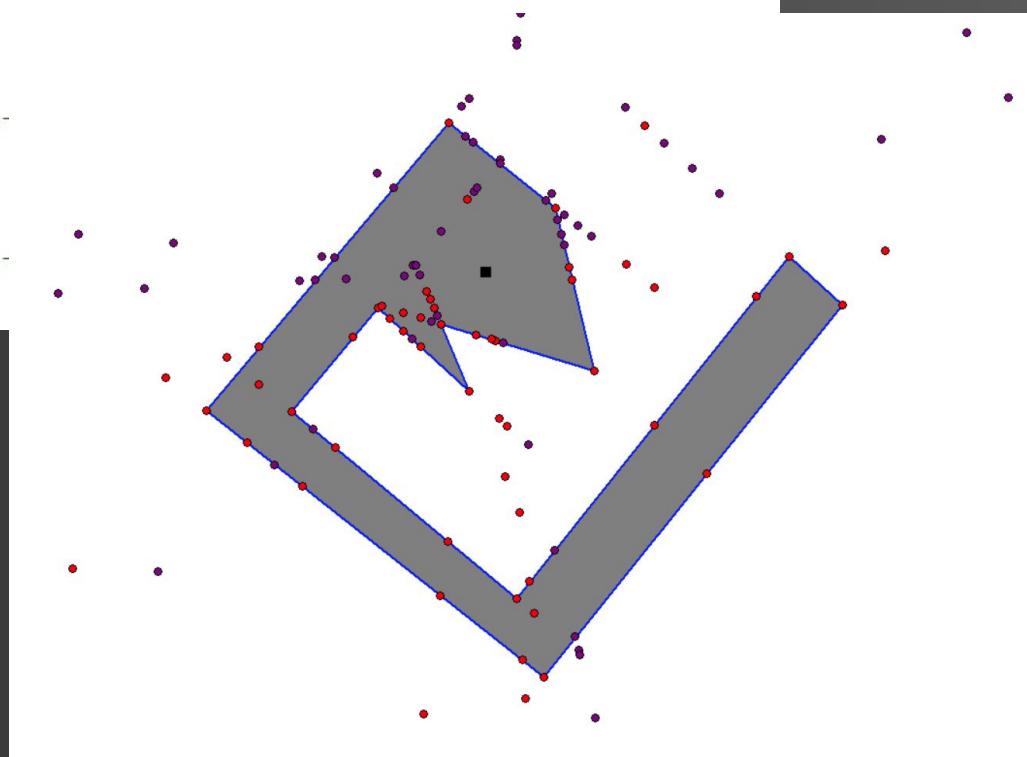
```

def barycentre(self):
    I=[0,0]
    if self.A1[0]-self.A2[0] == 0 or self.B1[0]-self.B2[0] == 0 :
        if (self.determinant(self.B1,self.B2,self.A2)+self.determinant(self.B2,self.B1,self.A1)) != 0 :
            I[0]=float("{:.4f}".format((self.determinant(self.B1,self.B2,self.A2)*self.A1[0]+self.determinant(self.B2,self.B1,self.A1)*self.A2[0])/(self.determinant(self.B1,self.B2,self.A2)+self.determinant(self.B2,self.B1,self.A1))))
            I[1]=float("{:.4f}".format((self.determinant(self.B1,self.B2,self.A2)*self.A1[1]+self.determinant(self.B2,self.B1,self.A1)*self.A2[1])/(self.determinant(self.B1,self.B2,self.A2)+self.determinant(self.B2,self.B1,self.A1))))
        else:
            I=[99999,99999]
    else:
        a=(self.A1[1]-self.A2[1])/(self.A1[0]-self.A2[0])
        b=(self.B1[1]-self.B2[1])/(self.B1[0]-self.B2[0])
        if a==b:
            I=[99999,99999]
        else:
            x=float("{:.4f}".format((self.B1[1]-self.A1[1]+a*self.A1[0]-b*self.B1[0])/(a-b)))
            y=float("{:.4f}".format(a*x+self.A1[1]-a*self.A1[0]))
            I=[x,y]
    return I

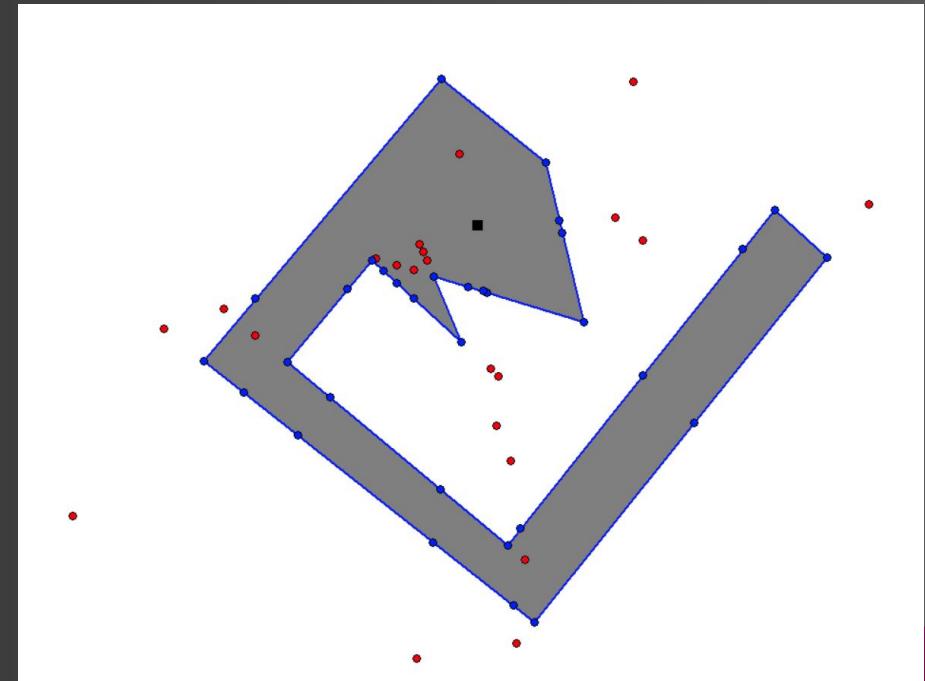
```

$$I = \frac{|B_1B_2A_2| A_1 + |B_2B_1A_1| A_2}{|B_1B_2A_2| + |B_2B_1A_1|}$$

```
if (empGar[0]-Lsave[j][0]>0 and empGar[1]-Lsave[j][1]>0):
    if (pointBar[0]<=empGar[0] and pointBar[1]<=empGar[1]):
        #cnvc.create_oval(int(pointBar[0])-4,int(pointBar[1])-4,int(pointBar[0])+4,int(pointBar[1])+4,fill="red")
        barVal.append(pointBar)
elif (empGar[0]-Lsave[j][0]<0 and empGar[1]-Lsave[j][1]<0):
    if (pointBar[0]>=empGar[0] and pointBar[1]>=empGar[1]):
        #cnvc.create_oval(int(pointBar[0])-4,int(pointBar[1])-4,int(pointBar[0])+4,int(pointBar[1])+4,fill="red")
        barVal.append(pointBar)
elif (empGar[0]-Lsave[j][0]<0 and empGar[1]-Lsave[j][1]>0):
    if (pointBar[0]>=empGar[0] and pointBar[1]<=empGar[1]):
        #cnvc.create_oval(int(pointBar[0])-4,int(pointBar[1])-4,int(pointBar[0])+4,int(pointBar[1])+4,fill="red")
        barVal.append(pointBar)
else:
    if (pointBar[0]<=empGar[0] and pointBar[1]>=empGar[1]):
        #cnvc.create_oval(int(pointBar[0])-4,int(pointBar[1])-4,int(pointBar[0])+4,int(pointBar[1])+4,fill="red")
        barVal.append(pointBar)
```

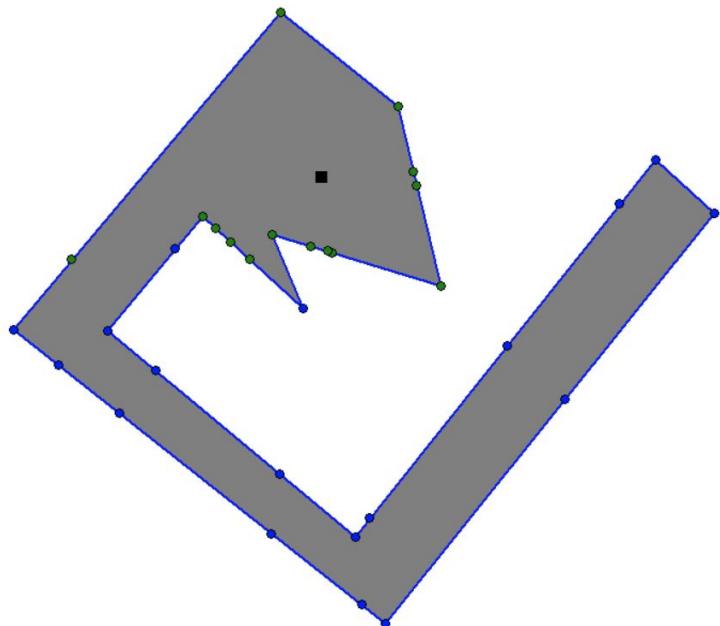


```
for y in range (len(Lsave)-1):
    for z in range (len(barValTri1)):
        if equationc(barValTri1[z],Lsave[y],Lsave[y+1]) == True :
            barValTri2numerote.append([barValTri1[z][0],barValTri1[z][1],y])
            barValTri2.append([barValTri1[z][0],barValTri1[z][1]])
            #cnvc.create_oval(int(barValTri2[-1][0])-4,int(barValTri2[-1][1])-4,int(barValTri2[-1][0])+4,int(barValTri2[-1][1])+4,fill="blue")
```



```
for droite in aligne:  
    triModule=[ ]  
    [triModule.append([module(point,empGar),point]) for point in droite]  
    triModule.sort()  
    lumiere(triModule,0)
```

```
for xxx in pointSegment:  
    combo=[xxx]  
    for yyy in barValTri2Uni:  
        if equation2(yyy,xxx,empGar) == True:  
            if yyy not in combo:  
                combo.append(yyy)  
                #print("aaaaaaaaaaaa")  
                if yyy in pointSegment:  
                    pointSegment.remove(yyy)  
    aligne.append(combo)
```



```

def lumiere(rayonL,n):
    """fonctionnement d'un rayon de lumiere qui passe par un angle

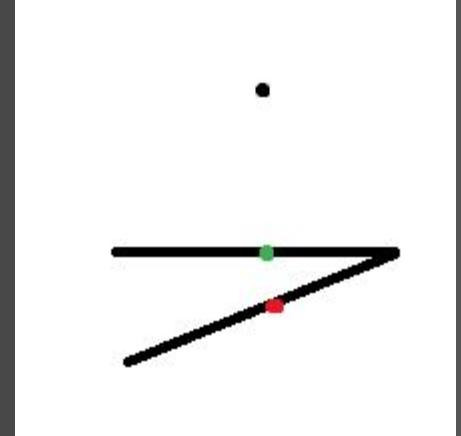
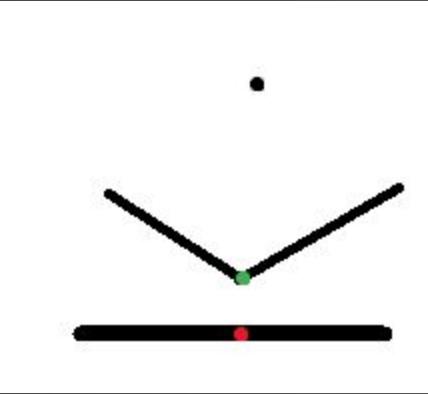
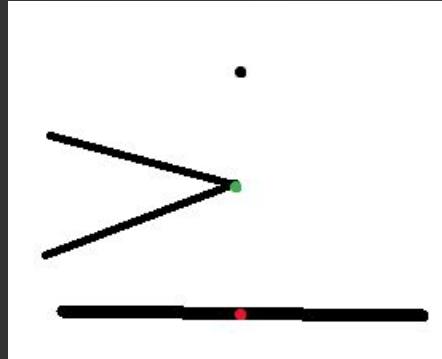
Args:
    rayonL (list): coordonne de point qui sont aligné
    n (int): numeros de la répétition

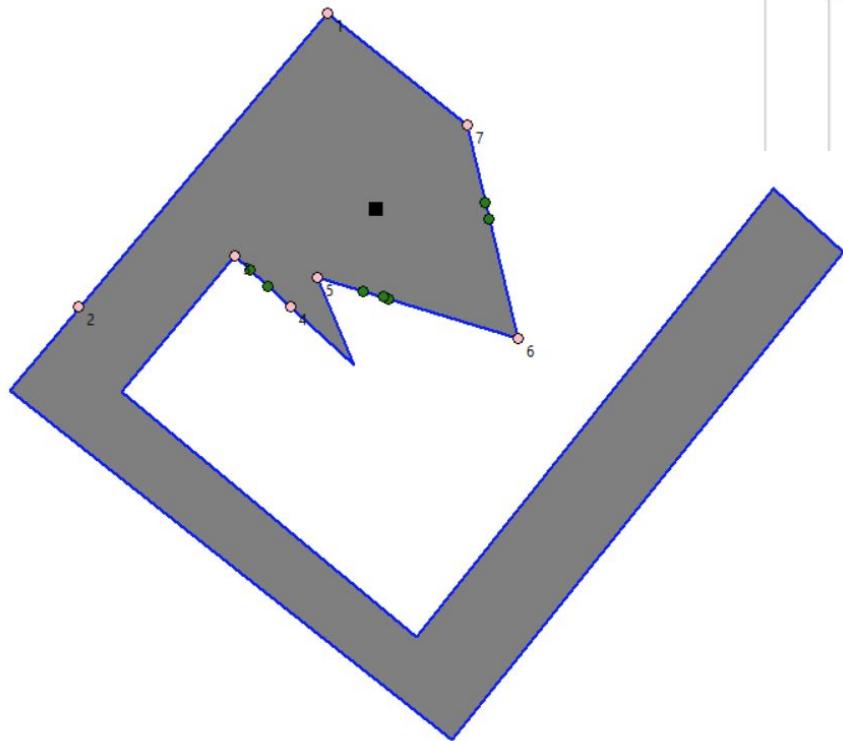
"""

global Lsave, barValTri2Sup
#print(rayonL[n][1])
if rayonL[n][1] not in Lsave :
    #print("aaaa")
    for i in range (n+1,len(rayonL)):
        barValTri2Sup.append(rayonL[i][1])
else :
    positionAngle = Lsave.index(rayonL[n][1])
    if positionAngle == 0 :
        if posDroite(empGar , Lsave[positionAngle], Lsave[2], Lsave[-2]) == True :
            n+=1
            return lumiere(rayonL,n)
    else :
        if posDroite(empGar , Lsave[positionAngle], Lsave[positionAngle+1], Lsave[positionAngle-1]) == True :
            n+=1
            return lumiere(rayonL,n)

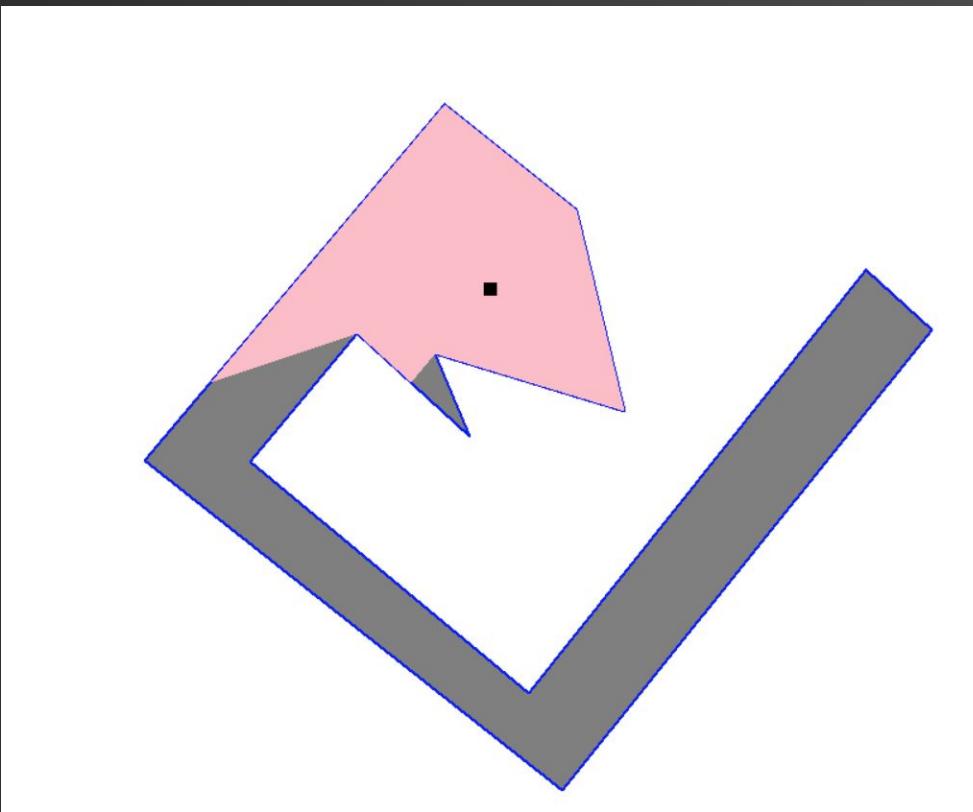
    for i in range (n+1,len(rayonL)):
        barValTri2Sup.append(rayonL[i][1])

```

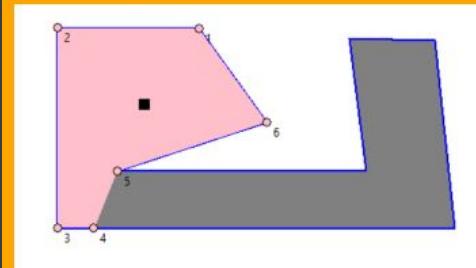
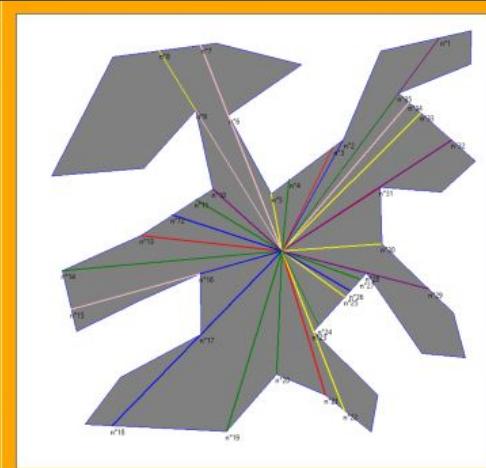
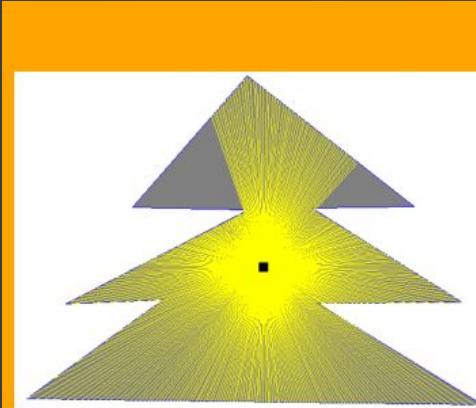




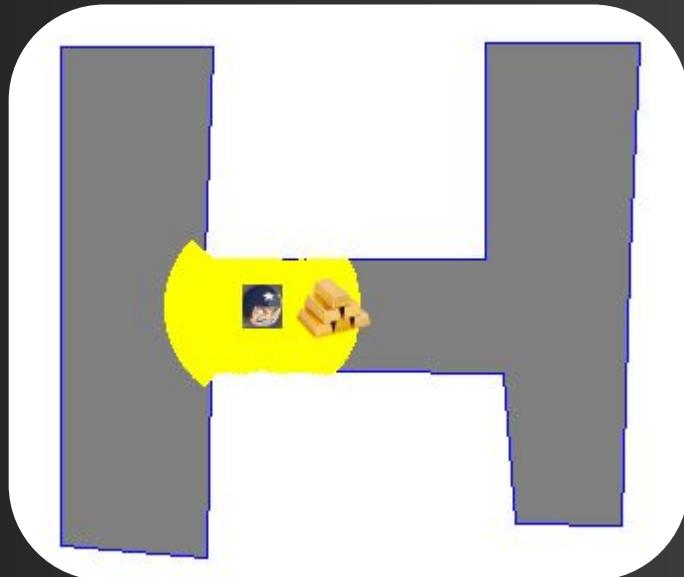
```
for u in range (0,len(Lsave) - 1) :  
    triPoint = []  
    for point in barValTri2Uni :  
        if equationc(point,Lsave[u],Lsave[u+1]) == True :  
            triPoint.append([module(Lsave[u],point),point])  
    triPoint.sort()  
    #print(triPoint)  
    if triPoint != [] :  
        if triPoint[0][1] not in polygoneVue :  
            polygoneVue.append(triPoint[0][1])  
        if triPoint[-1][1] not in polygoneVue :  
            polygoneVue.append(triPoint[-1][1])
```



# Game



# La ruée vers l'or (jeu)



Bien joué niveau complété



# Chargement du polygone par niveau

```
def niveau1():
    global Lsavepolygon,Lsaveor

    reset()

#Chargement des cotés du polygone niveau1 dans un dossier texte
fichierpolygon='map1' #nom du fichier où est stocké les coordonnées des côtés du polygone
Lsavepolygon=RecuperationPoint(fichierpolygon)#Récupération des coordonnées des cotées dans le fichier enregistré
#Création du polygone
cnvgame.create_polygon(Lsavepolygon,fill='grey',width=epaisseur, outline='blue')
print(Lsavepolygon)
```

```
"""photoor='Gold50.png'"""
fichieror='or1' #nom du fichier où est stocké les coordonnées des lingots d'or caché
Lsaveor=RecuperationPoint(fichieror)#Récupération des coordonnées des cotées dans le fichier enregistré
print(Lsaveor)
"""photo = tk.PhotoImage(file=photoor) """
```

*#Gestion des boutons et touches*

```
cnvgame.bind('<3>', premierpoint)
cnvgame.bind('<1>', position)
cnvgame.bind('<z>', mouvementhaut)
cnvgame.bind('<s>', mouvementbas)
cnvgame.bind('<d>', mouvementdroit)
cnvgame.bind('<q>', mouvementgauche)

wndgame.mainloop()
```

# Affichage champ de vision

- Méthode A :Détection Overlapping

```
C=[] #liste contenant tout les coordonées des points de la bordure des rayons
for k in range (360): # Pour faire un tour
    x=math.radians(k) #On convertie en radians pour l'utilisation de cosinus et sinus
    a=math.cos(x)
    b=math.sin(x)
    R=1
    collision=len(cnvgame.find_overlapping(x0+R*a-1,y0+R*b-1,x0+R*a,y0+R*b)) #mesure sur le nombre d'élément sous le point
    while collision==1 and R<50: #On déplace le point suivant le rayon jusqu'à qu'il sorte du polygone avec un rayon max de 50
        R+=1
        collision=len(cnvgame.find_overlapping(x0+R*a-1,y0+R*b-1,x0+R*a,y0+R*b))

    pointx=x0+(R-epaisseur)*a #position du point suivant x - la taille de la bordure
    pointy=y0+(R-epaisseur)*b #position du point suivant y - la taille de la bordure
    coordonne=pointx,pointy
    C.append(coordonne) #on l'ajoute dans notre liste contenant tout les coordonées des points de la bordure des rayons

for k in range (360): #On dessine ici tout les rayons avec les positions enregistrées dans la liste C
    pointx,pointy=C[k]
    cnvgame.create_line(x0, y0,pointx,pointy, fill='yellow', width=2, tag='effaceligne')#création d'un rayon avec deux coordonnées
```

# Affichage perso et lingot d'or trouvé

```
photoperso='perso.png'  
photocarre = tk.PhotoImage(file=photoperso)  
cnvgame.create_image(x0,y0, image=photocarre,tag='effacecarre')  
#Permet de dessiner l'or au premier plan au dessus des rayons
```

```
#Permet dessiner l'or au premier plan au dessus des rayons  
if (len(Ldecouvertor)!=0):  
    photoor='Gold50.png'  
    photo = tk.PhotoImage(file=photoor)  
    for k in range (len(Ldecouvertor)):  
        x,y=Ldecouvertor[k]  
        cnvgame.create_image(x,y, image=photo,tag='or2')  
wndgame.mainloop()
```

```
"""Gestion du premier clic gauche dans le polygone: création d'un champ de vision de l'utilisateur suivant 360 degrés"""

def premierpoint(event):
    global x0,y0
    x0, y0 = event.x, event.y # coordonnées du point du clic
    Verification() #Fonction qui calcul si un lingot d'or n'a pas été observé dans le champ de vision de l'utilisateur
    gardiangame(x0,y0) #fonction qui permet de tracer les rayons jaunes autour de l'utilisateur
```

```
"""Gestion du mouvement haut de l'utilisateur dans le polygone: touche Z"""

pas=20 #Pas de déplacement de l'utilisateur

def mouvementhaut(event):
    global x0,y0
    x0,y0=x0,y0-pas

    #Pour ne pas sortir du polygone
    #le voleur se déplace avec un pas de 20 soit suivant x ou y
    #on calcule si son prochain emplacement est en dehors du polygone
    #si oui le voleur reste sur place
    if (len(cnvgame.find_overlapping(x0-1,y0-1,x0+1,y0+1))==0):
        x0,y0=x0,y0+20

    Verification()
    gardiangame(x0,y0)
```

```

#Fonction qui permet de vérifier sur tous les emplacements de lingots d'or
#si le champ de vision du voleur de là pas découvert

def Verification():
    global Lsaveor, Ldecouvertor

#Pour tous les lingots d'or non trouvé on parcourt la liste et vérifie que le champ de vision de l'utilisateur ne survole pas un lingot d'or
if len(Lsaveor)!=0:
    for k in range (len(Lsaveor)):
        x,y=Lsaveor[k]
        if len(cnvgame.find_overlapping(x-5,y-5,x+5,y+5))>1: #on mesure le nombre d'élément à l'emplacement d'un lingot d'or
            """cnv.create_rectangle(x-10,y-10,x+10,y+10,fill='yellow')"""
            Ldecouvertor.append([x,y]) #on l'ajoute dans la liste lingot d'or découvert pour l'affiché par la suite dans la fonction guardian
            print(Ldecouvertor)
        del Lsaveor[k] #on l'ajoute dans la liste lingot d'or non trouvé
    break

#Si tous les lingots d'or ont été trouvé le jeu est terminé la liste est nul
if len(Lsaveor)==0:
    cnvgame.create_text(largeur//2,hauteur//2,text='Bien joué niveau complété !',font=("Arial", 40),fill='gold')

```

# Interface menu

```
def menu():
    global wnd,wnda,cnv,cnva

    wnd =tk.Tk()
    wnd.title("Menu_Art_Gallery")

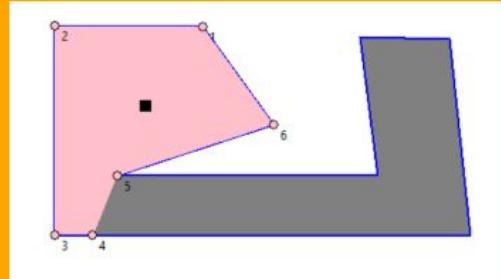
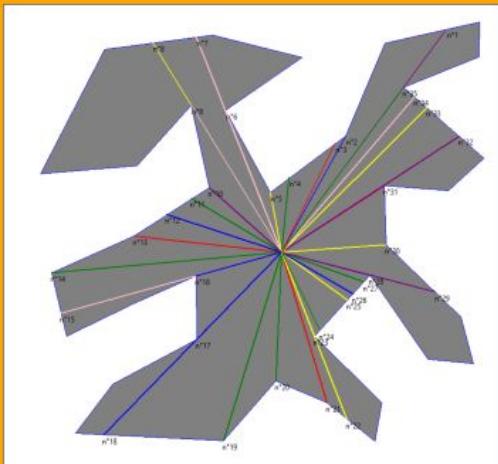
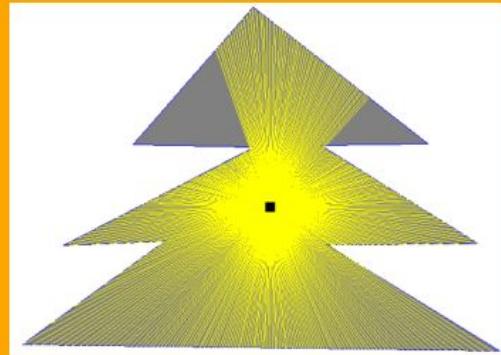
    fichier1='Ninterfacediagonale.png'
    fichier2='Nintefacecoté.png'
    fichier3='Ninterfacebarrycentre.png'
    fichier4='NVideo-Game-Controller-Icon-IDV-green.svg.png'

    cnv = tk.Canvas(wnd, width=largeur, height=hauteur, bg='white')

    photo1 = tk.PhotoImage(file=fichier1)
    photo2 = tk.PhotoImage(file=fichier2)
    photo3 = tk.PhotoImage(file=fichier3)
    photo4 = tk.PhotoImage(file=fichier4)

    btn=tk.Button(wnd,width=largeur//3,height=hauteur//2,image=photo1, bg='orange',command=methodea)
    btn.place(x=0,y=0)
    btn=tk.Button(wnd,width=largeur//3,height=hauteur//2,image=photo2,bg='orange',command=methodeb)
    btn.place(x=largeur//3,y=0)
    btn=tk.Button(wnd,width=largeur//3,height=hauteur//2,image=photo3,bg='orange',command=methodec)
    btn.place(x=0,y=hauteur//2)
    btn=tk.Button(wnd,width=largeur//3,height=hauteur//2,image=photo4,bg='orange',command=game)
    btn.place(x=largeur//3,y=hauteur//2)

    cnv.create_text(largeur*4//5,hauteur//2,text='ART GALERIE\n\n\nCLASSE: P2C \n\n\nMathis VEBER Tom TRICOIRE Arnaud HUCHON \n\n\n')
    cnv.pack()
    cnv.focus_set()
    wnd.mainloop()
```



ART GALERIE

CLASSE: P2C

Mathis VEBER Tom TRICOIRE Arnaud HUCHON

NOTICE:

Tracer polygone puis T pour dessiner

Clique droit pour afficher gardien

RACOURCIE TOUCHE:

R pour reset

M pour sauvegarder

L pour charger fichier

B pour calculer aire avec la méthode 1

N pour calculer aire avec la méthode 2

Touche mouvement jeu: ZQSD

# Retour menu

```
#fonction permettant la fermeture de la fenetre en cours et l'ouverture du menu
def retourmenua():
    global wnd,wnda,cnv,cnva
    del L[:]
    del Lsave[:]
    del listeevent[:]
    del Lcote[:]
    wnda.destroy()
    menu()
```

**MERCI DE VOTRE  
ATTENTION**