Université Paris Ouest Nanterre
Licence d'Info-Com
2015-2016

Prise en main interface graphique Tkinter (Python)

Le langage Python dispose d'un module tkinter qui permet d'afficher des fenêtres, des boutons, des images ...

1 Créer et gérer une fenêtre

1.1 Créer la fenêtre

```
form tkinter import *          # Charge le module Tkinter
fenetre=Tk()                   # Creer de la fentere nommee "fenetre"
fenetre.mainloop               # Lance la surveillance de la fenetre
```

Recopier le code et le lancer.

1.2 Des méthodes

Les fenêtres sont des objets sur lesquels sont définies des fonctions ou des procédures appelées " méthodes". Pour redimensionner le fenêtre, on utilise la méthode `geometry` :

```
fenetre.geometry("500x150")
```

Pour mettre un titre sur la fenêtre, on utilise la méthode `title` :

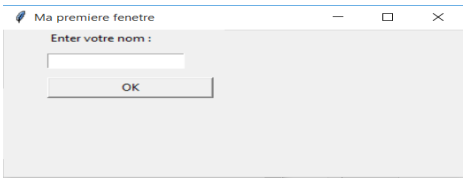
```
fenetre.title("Ma premiere fenetre")
```

2 Premiers widgets

On appelle "widgets" les différents objets graphiques que l'on va placer dans la fenêtre :

- Le widget Button permet d'afficher un bouton avec un texte (OUI, NON, Confirmer)
- Le widget Label permet afficher un texte sur votre fenêtre
- Le widget Entry permet à l'utilisateur d'entrer un texte.

```
1  from tkinter import *
2  fenetre=Tk()
3  fenetre.geometry("200x150")
4  fenetre.title("Ma premiere fenetre")
5  bouton=Button(fenetre, text='OK',width=20)
6  texte=Label(fenetre, text='Entrer votre nom :')
7  retour=Entry(fenetre)
8  bouton.place(x=40,y=60)
9  texte.place (x=40,y=0 )
10 retour.place(x=40,y=30)
11 fenetre.mainloop()
```



Remarques :

- Les lignes 1 à 4 initialisent la fenêtre.
- La ligne 5 crée un Button appelé `bouton`. On a passé en argument le nom de la fenêtre sur laquelle il est déposé, le texte qui apparaitra et sa largeur.
- La ligne 6 crée un Label appelé `texte`. On a passé en argument le nom de la fenêtre sur laquelle il est déposé, le texte qui apparaitra et sa largeur.

– La ligne 7 crée un `Entry` appelé `retour`. On a passé en argument le nom de la fenêtre sur laquelle il est déposé.
– Les lignes 8, 9 et 10 demandent à Python de placer les trois widgets en précisant leurs coordonnées.
– La ligne 8 peut être remplacée par :
`bouton.pack()` *# affiche le widget dans la fenêtre*
si on ne veut pas spécifier l'emplacement du widget (idem pour les lignes 9 et 10).

2.1 Label : Afficher un texte

Le widget Label permet d'afficher un texte à l'écran. On peut préciser un certain nombre de paramètres :

- `fg` et `bg` précisent la couleur du texte et la couleur du fond
  Les couleurs disponibles de base sont :
  "blue" "red" "green" "yellow" "brown" "black" "white" "pink" "orange" "purple" "grey"
  Sinon il faut préciser la couleur en format RGB par exemple `"#c71585"` .
  Voir `https://pythonhosted.org/ete2/reference/reference_svgcolors.html` pour une palette de couleurs.
- `Font(famille, taille, decoration)` préciseb :
  – la famille c'est-à-dire la police utilisée (arial, verdana, times . . .)
  – la taille c'est-à-dire un entier qui precise la hauteur de la fonte, en 11 pour ce texte.
  – La décoration c'est-à-dire normal, bold, italic, underline, overstrike . . .

2.2 Button

Le widget Button est donc un bouton sur lequel l'utilisateur va cliquer pour déclencher une action.

- Quelques méthodes :
  – `bouton.config` : Permet de modifier les paramètres du widget, par exemple : *bouton.config(text='JOUR')* va modifier le texte du bouton.
  – `bouton.cget` : renvoi la valeur de l'option demandée sous la forme d'une chaine. Par exemple *bouton.cget(text)* va retouner le texte du bouton.
- `Bouton.place_forget` : Cette méthode permet d'enlever un bouton quand on en a plus besoin.
- Le paramètre `command` : Lors de la création du bouton on rajoute le paramètre `command` afin de préciser l'action à déclencher lors du clic.

```
1  from tkinter import *
2
3  def change():
4      if bouton.cget('text')=='JOUR':
5          bouton.config(text='NUIT')
6      else:
7          bouton.config(text='JOUR')
8
9  fenetre=Tk()
10 fenetre.geometry("280x50")
11 fenetre.title("Ma fenetre JOUR NUIT")
12 bouton=Button(fenetre, text='JOUR',width=30, command=change)
13 bouton.place(x=20,y=10)
14
15 fenetre.mainloop()
```

2.3 Entry

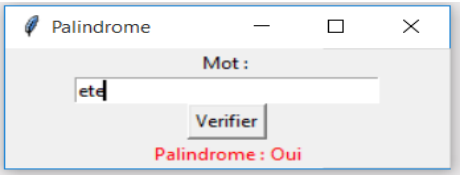
Ce widget permet au joueur de saisir un texte court sur une ligne.
Pour accéder au contenu du texte, on utilise la méthode `retour.get`
On peut forcer le curseur à se placer sur l'Entry en exécutant la méthode `retour.focus`

3 Exemples

a) **Palindrome :**

Un palindrome est un mot qui peut-être lue de gauche à droite ou de droite à gauche. Par exemple :

- Été- Kayak
- Réifier- Ressasser
- Selles- Sèves
- Pop - Rêver
- Têt



Lorsque l'on clique sur le bouton " *Verifier*", on affichera si le mot entré est un palindrome.
Si l'on actionne le bouton "*Autre couleur* ", une nouvelle couleur est tirée au hasard dans une série limitée. Cette couleur est celle qui s'appliquera aux tracés suivants.
Le bouton " *Quitter* " sert bien évidemment à terminer l'application en refermant la fenêtre.

Script :

```
from tkinter import *
fenetre = Tk()
fenetre.title("Palindrome")
# On definit la fonction appelee par le bouton
def palindrome() :
    m=maZone.get()
    n=len(m)
    vrai=1
    i=0
    while((i¡=(n//2)) and (vrai==1)) :
        if (m[i] !=m[n-1-i]) :
            vrai=0
        i=i+1
    if vrai==1 :
        Affichage.config(text = "Palindrome : " + 'Oui',fg='red')
    else :
        Affichage.config(text = "Palindrome : " + 'Non',fg='red')

# On cree un Label
champLabel = Label(fenetre, text="Mot : ")
champLabel.pack()
# On cree un Entry
maZone = Entry(fenetre, width=30)
maZone.insert(0, "Entrez votre mot ")
maZone.pack()
# On cree un Button
monBouton = Button(fenetre, text="Verifier", command=palindrome)
# On affiche le Button dans la fenetre
monBouton.pack()
# On cree un Label pour l'affichage
Affichage = Label(fenetre, text="Palindrome : ")
Affichage.pack()
fenetre.mainloop()
```

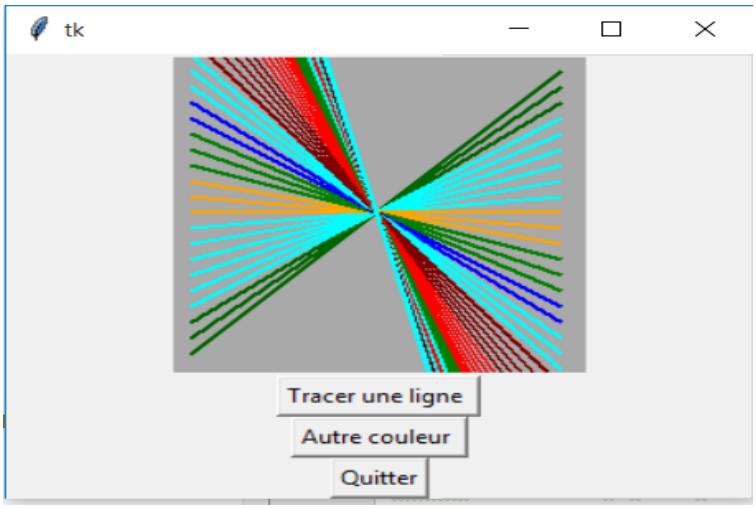
b) **Palindrome :**

Écrivez une fonction palindrome qui détermine si la chaine (phrase) passée en argument est un palindrome sans tenir compte de la ponctuation, de la casse ou des espaces. La fonction palindrome renvoie un booléen.

- le sel
- Engage le jeu que je le gagne
- Élu par cette crapule.
- Zeus a été à Suez.
- Eh ! ça va la vache ?
- Tu l'as trop écrasé, César, ce Port-Salut !

c) **Tracé de lignes dans un canvas**

Le premier exemple de cette section consiste à créer une fenêtre comportant trois boutons et un canvas. Suivant la terminologie de tkinter, ***un canvas est une surface rectangulaire délimitée, dans laquelle on peut installer ensuite divers dessins et images***



Lorsque l'on clique sur le bouton " *Tracer une ligne* ", une nouvelle ligne colorée apparaît sur le canvas, avec à chaque fois une inclinaison différente de la précédente.
Si l'on actionne le bouton "*Autre couleur* ", une nouvelle couleur est tirée au hasard dans une série limitée. Cette couleur est celle qui s'appliquera aux tracés suivants.
Le bouton " *Quitter* " sert bien évidemment à terminer l'application en refermant la fenêtre.

Description du script :

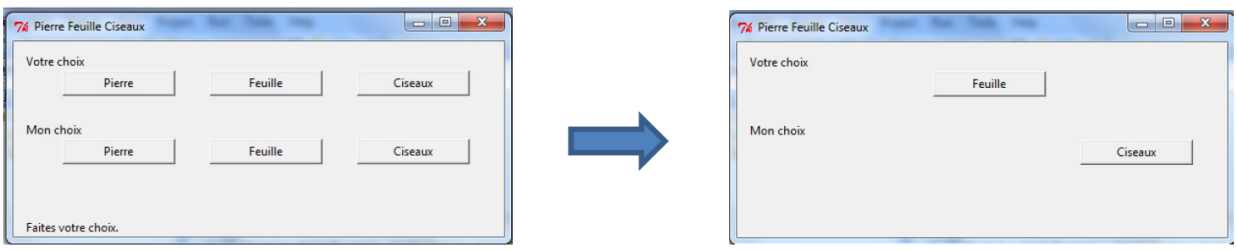
Le script décrit ci-dessous implémente notre première application en recourant à des variables globales. Comme nous pouvons le constater, les fonctions de ce script peuvent modifier les valeurs de certaines variables qui sont définies au niveau principal du programme, grâce à l'instruction global utilisée dans leur définition. Nous procédons ainsi afin de pouvoir distinguer les différents composants d'un programme piloté par des événements. Néanmoins, sachez que cette pratique n'est vraiment pas recommandable surtout lorsqu'il s'agit d'écrire de grands programmes !

```
from tkinter import *
from random import randrange

def DessinLigne () :
    # Trace d'une ligne dans le canvas can1
    global x1 , y1 , x2 , y2 , coul
    can1.create_line ( x1 , y1 , x2 , y2 , width = 2 , fill = coul )
    # modification des cordonnees pour la ligne suivante
    y2 =y2 + 10
    y1 =y1 - 10
```

```
def ChangerCouleur () :
    # Changement aleatoire de la couleur du trace
    global coul
    palette = ['purple','cyan', 'maroon', 'green', 'red', 'blue', 'orange', 'yellow']
    c = randrange ( len ( palette )) # nombre aleatoire parmi les index possibles de la palette
    coul = palette [ c ]

# ------ Programme Principal ---------
# variables globales
x1 , y1 , x2 , y2 =10 , 190 , 190 , 10          # coordonnees de la ligne
coul = 'dark green'          # couleur de la ligne
fen1 = Tk ()          # widget de la fenetre principale
# Canvas :
can1 = Canvas(fen1, bg ='dark grey', height = 200, width = 200)
can1.pack ()
# Boutons :
bouton1 = Button ( fen1 , text = 'Quitter', command = fen1 . destroy )
bouton1 . pack ( side = BOTTOM )          # positionnement du bouton1
bouton2 = Button ( fen1 , text = 'Tracer une ligne ', command = DessinLigne )
bouton2 . pack ()          # positionnement du bouton2
bouton3 = Button ( fen1 , text = 'Autre couleur ', command = ChangerCouleur )
bouton3 . pack ()          # positionnement du bouton3
fen1 . mainloop ()          # lancement de la boucle d'evenements
```

La fonctionnalité de ce programme est essentiellement assurée par les deux fonctions **DessinLigne()** et **ChangerCouleur()**, qui seront activées par des événements, ceux-ci étant eux-mêmes définis dans la phase d'initialisation.
Dans la phase d'initialisation, située dans le corps principal du programme, on commence par importer l'intégralité du module tkinter ainsi que la fonction randrange du module random. On crée ensuite les différents widgets par instanciation à partir des classes **Tk()**, **Canvas()** et **Button()**. L'initialisation se termine avec l'instruction **fen.mainloop()** qui démarre le réceptionnaire d'événements. Les instructions qui suivent ne seront exécutées qu'à la sortie de cette boucle.
L'option **command** utilisée dans l'instruction d'instanciation des boutons permet de désigner le nom de la fonction qui devra être appelée lorsqu'un événement "*clic gauche de la souris sur le widget*" se produira. Dans notre fonction **ChangerCouleur()**, une couleur est choisie au hasard dans une liste. Nous utilisons pour ce faire la fonction **randrange()** importée du module **random**. Appelée avec un argument *n*, cette fonction renvoie un nombre entier, tiré au hasard entre *0* et *n-1*.
La commande liée au bouton " Quitter " appelle la méthode **quit()** de la fenêtre fen1. Cette méthode sert à fermer (quitter) le réceptionnaire d'événements (mainloop) associé à cette fenêtre. Lorsque cette méthode est activée, l'exécution du programme se poursuit avec les instructions qui suivent l'appel de mainloop.

d) **Réaliser un jeu "Pierre Feuille Ciseaux"**



Réaliser une interface graphique avec une fenêtre dans laquelle le joueur choisit l'un des boutons "Pierre" "Feuille" "Ciseaux " alors les autres choix disparaissent laissant le choix du joueur et indiquant celui de la machine

- Améliorer le programme afin qu'il affiche quel est le gagnant et sinon s'il y a égalité.
- Améliorer le programme afin que le joueur puisse rejouer.

Tutoriel

Bibliothèque graphique Tkinter / WIDGET CANVAS

Le module Tkinter dispose d'un widget nommé Canvas destiné à dessiner sur la fenêtre.

Si on veut créer un canvas nommé *canv*, l'appel se fait comme pour tous les widgets, de la manière suivante :

canv = **Canvas**(*emplacement*, *paramètres*)

où *emplacement* est le nom de la fenêtre sur lequel il est posé et *paramètres* indique les différentes propriétés du Canvas comme :

Paramètres	Effet
bg	Précise la couleur de fond du Canvas
height	Précise la hauteur du Canvas
width	Précise la largeur du Canvas

Il faudra bien sûr placer dans un 2ème temps le widget sur la fenêtre à l'aide des méthodes **place** ou **grid**.

I/ Dessiner :

- ✓ Sur un Canvas noté ici C, on peut dessiner **une ligne** :

ligne = C**.create_line**(x1,y1,x2,y2,*options*)

Cette instruction dessine un segment reliant le point de coordonnées (x1 ; y1) inclus au point de coordonnées (x2 ; y2) exclu.

Options	Effet
fill	Précise la couleur de la ligne
width	Précise l'épaisseur de la ligne en pixels

- ✓ On peut dessiner **un rectangle** :

rect = C**.create_rectangle**(x1,y1,x2,y2,*options*)

Cette instruction dessine un rectangle dont deux sommets opposés ont pour coordonnées (x1 ; y1) et (x2 ; y2).

Options	Effet
fill	Précise la couleur de l'intérieur du rectangle
outline	Précise la couleur du trait
width	Précise l'épaisseur du trait en pixels

- ✓ On peut dessiner **une ellipse** :

el = C**.create_oval**(x1,y1,x2,y2,*options*)

Cette instruction dessine une ellipse inscrite dans un rectangle dont deux sommets opposés ont pour coordonnées (x1 ; y1) et (x2 ; y2). Il y a les mêmes options que pour **create_rectangle**.

Pour obtenir un cercle, il suffit d'inscrire l'ellipse dans un carré : ce sera alors un cercle.

Schéma :

II/ Afficher un texte :

txt = C**.create_text**(*x*,*y*,*options*) affiche un texte au point de coordonnées (*x* ; *y*).

Options	Effet
fill	Couleur du texte
font	Police de caractères utilisée
text	Chaîne de caractères : texte à afficher
anchor	Précise la position d'attache du texte par rapport au point de coordonnées (*x* ; *y*) : *'center'* par défaut (texte centré autour du point indiqué) ou : *'n','e','s','w','nw','ne','sw','se'*

III/ Insérer des images :

On travaillera très bien avec des images au format GIF : images en 256 couleurs, possédant éventuellement une couleur de transparence (à voir).

Etape 1 : on charge l'image dans une variable globale, par exemple *fichierimg*, à l'aide de l'instruction :

fichierimg = **PhotoImage**(**file =** ´´ image.gif ´´)

Etape 2 : on place l'image au point de coordonnées(*x* ; *y*) sur le Canvas C avec la méthode **create_image** :

img = C**.create_image**(*x* , *y* , *options*)

Le paramètre *options* permet de renseigner les propriétés :

Propriétés	Effet
image	Indique l'image à afficher : *fichierimg* dans l'exemple.
anchor	Précise la position du point de coordonnées (*x* ; *y*) de référence par rapport au reste de l'image : *'center'* par défaut ou : *'n','e','s','w','nw','ne','sw'* ou *'se'*

IV/ Modifications en cours de programme :

1. Changer les propriétés :

Méthode	Effet
C**.delete**(*item*)	Efface l'item *item* du Canvas C
C**.delete**(ALL)	Efface tout ce qui se trouve sur le Canvas C
C**.coords**(*item*,x0,y0)	Modifie les coordonnées de l'item *item*
C**.coords**(*item*,x1,y1,x2,y2)	Modifie les coordonnées de l'item *item*
C**.itemconfig**(*item*,*options*)	Permet de modifier une ou plusieurs options de l'item *item*
C**.itemcget**(*item*,*prop*)	Renvoie la valeur de la propriété *prop* de l'item *item.*

Exemple : C.itemconfig(**titre** , text = 'Gagné', fill = 'red') transforme le contenu du texte **titre** placé sur le Canvas C en 'Gagné' écrit en rouge.

2. Profondeur des objets :

Les objets images se superposent dans l'ordre dans lequel ils sont créés. Ainsi si un personnage est créé puis un décor, on ne verra pas le personnage car il sera placé derrière le décor, élément créé en dernier !

Pour modifier cet ordre, on peut utiliser les méthodes suivantes :

Méthode	Effet
C.**tag_raise**(*obj*)	Place l'objet *obj* en premier plan du Canvas C
C.**tag_lower**(*obj*)	Place l'objet *obj* en arrière plan du Canvas C

Gestion du clavier

Piloter un objet avec des boutons n'est pas aisée. Utiliser les touches serait préférable.

A chaque fois qu'une touche est enfoncée ou relâchée, on dit qu'un événement est déclenché. Il existe une multitude d'événements : souris qui se déplace, fenêtre que l'on redimensionne, que l'on ferme … La méthode mainloop() que l'on place toujours en fin de programme déclenche justement le gestionnaire d'événements qui surveille tous ces événements.

I/ Appui sur une touche classique :

Pour le moment, on s'intéresse à l'appui d'une touche classique représentant un caractère alphanumérique.

On peut vouloir surveiller un événement clavier :

- sur un Entry : E.**bind**('<Key>', *fonct*)
- sur la fenêtre entière : *fenetre*.**bind_all**('<Key>', *fonct*)

E est le nom du Widget, *fenetre* est le nom de la fenêtre et *fonct* la fonction à appeler lorsqu'une touche est enfoncée.

La fonction *fonct* va devoir être définie d'une manière spéciale :

```
def fonct(evt) :
    ……..
    ……..
```

Le paramètre *evt* qui est obligatoire (on peut changer son nom) va permettre de récupérer des informations sur l'événement. Par exemple, à l'intérieur de la fonction, *evt*.**char** renvoie le caractère sur lequel on a appuyé.

```
from tkinter import *
def touche(evt):
    print('Vous avez appuyé sur la touche', evt.char)
fenetre = Tk()
fenetre.bind_all('<Key>', touche)
fenetre.mainloop()
```

II/ Appui sur une touche spéciale :

Certaines touches comme F1 ne correspondent à aucun caractère. L'objet *evt* que l'on récupère lors d'une action au clavier peut renvoyer d'autres informations :

evt.char : renvoie le caractère correspondant à la touche enfoncée

evt.keysym : renvoie une chaîne de caractères contenant le symbole correspondant à la touche enfoncée

evt.keycode : renvoie un entier contenant le code correspondant à la touche enfoncée

evt.widget : renvoie le widget sur lequel l'action a été déclenchée

Par exemple :

Touche	keysym	keycode
	space	32
↑	Up	38
↓	Down	40
←	Left	37
→	Right	39
↲	Return	13

Pour les autres touches classiques (A, 0, f, …) : le keycode est le code ASCII du caractère : il s'agit d'un entier.

III/ Surveiller une touche particulière ou une combinaison de touches :

Le gestionnaire d'événements ne surveille plus toutes les touches possibles. On va lui demander de ne surveiller que les touches Haut, Bas, Gauche, Droite, Espace.

Exemple : **fenetre.bind_all('<Up>',haut)**

fenetre.bind_all('<Down>',bas)

La fonction haut est exécutée quand le joueur enfonce la flèche du haut, la fonction bas est exécutée si le joueur enfonce la flèche du bas.

IV/ Gestion des touches multiples :

Problème si jeu à 2 joueurs où une seule touche peut être enfoncée à la fois !

Penser aux méthodes :

Fen.bind_all('<KeyPress>',action) Exécute la fonction action lorsqu'une touche est enfoncée

Fen.bind_all('<KeyRelease>',action) Exécute la fonction action lorsqu'une touche est relâchée.

Gestion de la souris

I/ Les différents types d'événements :

Comme pour le clavier, on peut surveiller différents événements. La syntaxe est identique :

widget.**bind**(*evenement* , *action*)

où :

- *widget* est un widget sur lequel on surveille l'événement. Cela peut être aussi la fenêtre principale. Dans ce cas, l'événement se déclenche sur tous les widgets déposés sur celle-ci.
- *evenement* est une chaîne de caractères représentant le type d'événements à surveiller : clic, déplacement …
- *action* est le nom de la fonction à exécuter lorsque l'événement se déclenche. Cette fonction doit être déclarée comme demandant un paramètre en entrée.

Par exemple : **def** action(evt) :

……

La variable evt donnera diverses informations complémentaires sur l'action réalisée par le joueur.

Evènement	Est déclenché lorsque :
'<Motion>'	la souris est déplacée.
'<ButtonPress-1>'	le bouton gauche est enfoncé.
'<ButtonRelease-1>'	le bouton gauche est relâché.
'<Double-Button-1>'	le joueur fait un double-clic avec le bouton gauche.
'<B1-Motion>'	la souris est déplacée alors que le bouton gauche est maintenu enfoncé.
'<Enter>'	la souris entre sur le widget.
'<Leave>'	la souris sort du widget.

Le bouton gauche est identifié par le chiffre 1. Le bouton central par 2, le bouton droit par 3.

Si on veut surveiller un clic droit, on utilisera l'événement : '<ButtonPress-3>'.

II/ Information sur l'événement en cours :

On peut récupérer des informations sur la souris lorsque la fonction est exécutée grâce au paramètre obligatoire *evt*.

Paramètre	Information renvoyée
evt.x	Renvoie l'abscisse de la souris lors du déclenchement de l'événement.
evt.y	Renvoie l'ordonnée de la souris lors du déclenchement de l'événement. Ces coordonnées sont relatives au coin nord-ouest du widget situé sous le curseur.
evt.widget	Renvoie le widget pour lequel l'événement est déclenché : premier widget situé sous le curseur de la souris.
evt.num	Renvoie le numéro du bouton enfoncé ou relâché.
evt.delta	Renvoie un différentiel lors de l'utilisation de la molette de la souris.

Par exemple :

```
from tkinter import *
def cliquer(evt):
    if evt.widget==C:
        print("clic gauche sur le canvas")
    else :
        print("clic gauche sur la fenêtre")
    print("position : x=",evt.x,"y=",evt.y)
fen=Tk()
C = Canvas(fen,width=50,height=50,bg="white")
C.place(x=30,y=30)
fen.bind('<ButtonPress-1>',cliquer)
fen.mainloop()
```