Introduction aux GUI (Graphical User Interface) et à la programmation événementielle

I. Une (trop) courte introduction

`tkinter` est un module de « gadgets graphiques » (widgets - contraction de window et gadget) intégré par défaut dans Python et d'une grande simplicité.

Il existe d'autres bibliothèques de ce type (`wxPython, pyQT, pygame`...).

Un manuel de référence sur `tkinter` (168 pages en anglais !) est disponible sur Internet (google : tkinter reference guide).

Le principe général est :

- de créer une fenêtre graphique ;
- puis de placer et organiser des éléments (boutons, textes, zones de dessin, etc...) à l'intérieur de cette fenêtre.

Tous les éléments (fenêtre graphique comprise) sont considérés comme des **objets** que l'on peut modifier à l'aide de **méthodes**. C'est le principe de la programmation orientée objet (POO) que nous n'aborderons pas ici. La syntaxe est la suivante :

```
objet.methode(parametres_de_la_methode)
```

`tkinter` permet de jongler entre 15 widgets standards ! Nous nous focaliserons sur 4 widgets :

☐ **`Label`** ☐ **`Entry`** ☐ **`Button`** ☐ **`Canvas`**

II. Premières instructions

1) Création de notre première fenêtre :

☐ dans Geany, créer un fichier appelé gui01.py ;
☐ importer toutes les fonctions du module `tkinter` à l'aide de la commande import ;
☐ créer la fenêtre graphique parent à l'aide de l'instruction `fen = Tk()` ;
☐ demander l'affichage de la fenêtre graphique à l'aide de l'instruction `fen.mainloop()` .

Questions :

Que fait `fen = Tk()`?
Que fait `fen.mainloop`?

2) Ajouter du contenu :

Notre première fenêtre, bien que très jolie dans le style Windows 98, ne fait absolument rien. Nous allons modifier gui01.py .

☐ utiliser la méthode `title` sur l'objet `fen` . La méthode `title` prend comme paramètres une chaine de caractères: `title('Vive Windows 98')`

☐ ajoutons un texte à l'aide du widget **`Label`** :
Syntaxe : `tex1 = Label(fen, text= 'Hello, world!')`

Question : Le texte s'affiche-t-il ? _____

☐ faire apparaitre le texte en appelant : `tex1.pack()`

3) Ajouter un peu plus de contenu :

☐ ajouter un bouton « Ne sers à rien » à l'aide du widget **`Button`**.
Ce widget fonctionne de la même matière que le widget **`Label`**.
N'oubliez pas le `.pack()` !
Syntaxe : `but1 = Button(fen, text = 'Ne sera à rien')`

☐ ajouter un bouton « Quitter » à l'aide du widget **`Button`**.

Question : Que se passe-t-il quand on clique sur le bouton « Quitter »? Était-ce prévisible ?

Certains widgets ont des fonctionnalités accessibles grâce au paramètre **`command=`** .

☐ au sein du widget **`Button`** de "Quitter", ajoutez `command=fen.destroy`.

Question : Que se passe-t-il quand on clique sur le bouton ?

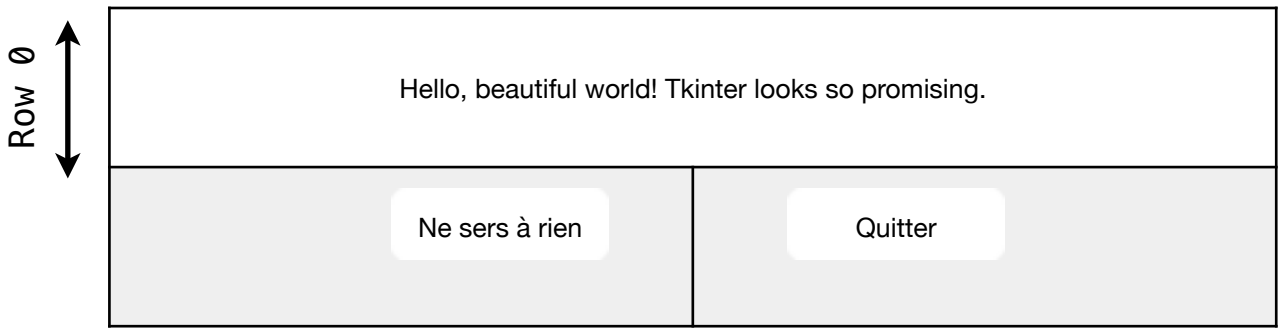
III. Gestion graphique de la fenêtre

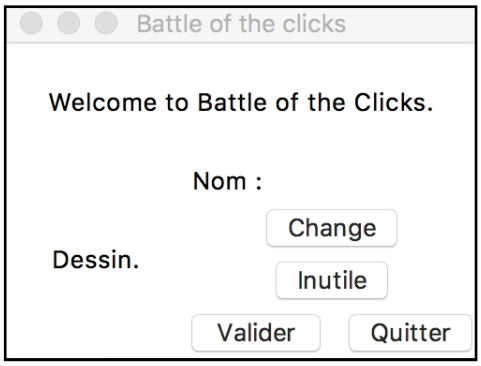
`.pack()` nous permet de placer **automatiquement** des objets et de les afficher.
Toutefois, nous souhaitons sans doute pouvoir disposer d'un certain contrôle dans le design de nos fenêtres graphiques. C'est là que la méthode **`.grid(……)`** entre en scène.

1) Organisation de fenêtre 1 :

☐ télécharger le fichier grid_example.py que vous trouverez sur github.com/bouillotvincent/coursNSI et l'enregistrer sur votre répertoire de travail;

☐ dans Geany, copier/modifier/supprimer/commenter certaines fonctions inconnues de **`.grid(……)`** et essayer de deviner leur rôles en remplissant le tableau ci-dessous.





2) Organisation de fenêtre 2

- ☐ dans Geany, créer un fichier appelé gui_02.py ;
- ☐ en effectuant des copier/coller des programmes précédents, créer une fenêtre graphique qui ressemble à la fenêtre ci-contre.

3) Création de command(e) spécifique :

La fonction `destroy` permet de fermer une fenêtre suite à un clic de la souris. Nous pouvons également créer des fonctions **sans paramètres d'entrée** que nous pouvons intégrer dans la majorité des widgets (voir documentation pour en savoir plus).

- ☐ dans le ficher gui_02.py, avant la partie graphique, créer une fonction `change()`
- ☐ `change()` va permettre de changer le bouton « inutile » en bouton « utile ».
  Syntaxe : utiliser la méthode **`.configure(text = '.....')`** sur le bouton but1 qui controle "Change"

Prolongement : Boucler sur le bouton « change » : le bouton « utile » redevient « inutile » lors d'un second clic etc. Vous aurez besoin de définir une variable globale.

4) Création de formulaire avec le widget Entry :

- ☐ télécharger le fichier gui_03.py que vous trouverez sur github.com/bouillotvincent/coursNSI et l'enregistrer sur votre répertoire de travail ;
- ☐ dans Geany, ouvrir le fichier et l'exécuter.

Questions :

Que remarquez-vous ?
Que devez-vous faire pour valider votre formulaire ?
Quelle méthode devez-vous appeler pour récupérer le contenu du formulaire ?

Prolongement : **`do_something(test, x)`** est une fonction appelée après **`command=`** mais elle contient un argument d'entrée ! Comment est-ce possible ?
Faites la même chose en simplifiant le programme et en utilisant une fonction sans argument.

5) Exercice d'application :

- ☐ télécharger le fichier lancer_des_enonce.py que vous trouverez sur github.com/bouillotvincent/coursNSI et l'enregistrer sur votre répertoire de travail ;
- ☐ modifier le pour simuler un lancer de dés à 6 faces sur la gauche (facile) et un compteur de lancers sur la droite (plus dur).

Prolongement : Ajouter un affichage permettant d'avoir le nombre de fois où vous avez tiré un 1, un 2, un 3 etc.

IV. Canvas, sweet Canvas

Les canevas **`Canvas()`** sont les widgets les plus « souples » du module `tkinter`. Ce sont aussi les widgets que vous allez utiliser le plus souvent.

Ce sont des rectangles orientés par un repère ayant pour origine le coin supérieur gauche et dans lesquels on peut :

- afficher des images et du texte ;
- tracer des dessins (pixelisés et en 2D !) ;
- animer des figures.

La syntaxe du widget **`Canvas()`** est la suivante :

```
canvas_nom = Canvas(fenetre parent, width = a, height = b)
```

Les figures affichées dans un canevas s'appellent des items. Il existe de très nombreux items disponibles… On s'intéressera pour l'instant uniquement aux : Segments ; Rectangles ; Ovales ; Textes et Images.

Pour visualiser les effets des différentes instructions proposées dans cette partie, on part du programme canvas_template.py que vous trouverez sur Github.

Au fur et à mesure de la lecture de cette partie, **modifiez et ajoutez** les instructions proposées à la suite de la ligne 17, sous le commentaire : `##---- Dessiner dans le canevas ----##`.

1) Segment :

```
canvas_nom.create_line(x0, y0, x1, y1)
```

- (x0 ; y0) et (x1 ; y1) sont les coordonnées des points extrémités du segment tracé.
- width _____
- fill _____

Exemple : ligne1 = canvas_nom.create_line(250, 175, 250, 225, width=4, fill='#d05e82')

2) Rectangle :

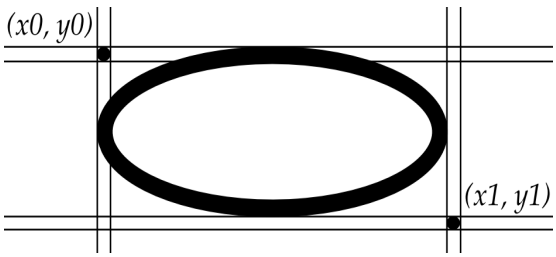
```
canvas_nom.create_rectangle(x0, y0, x1, y1)
```

- (x0 ; y0) et (x1 ; y1) sont les coordonnées des sommets opposés du rectangle.
- La colonne de pixels d'abscisses x1 n'est pas atteinte et la ligne de pixels d'ordonnées y1 non plus. Il faut ajouter *à la main* quelques pixels…
- width _____
- outline _____
- fill _____

Exemple : rect1 = canvas_nom.create_rectangle(175, 250, 325, 327, outline='#d05e82'

3) Ovale, Cercle :

```
canvas_nom.create_oval(x0, y0, x1, y1)
```

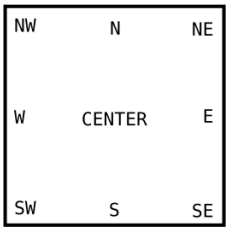


- (x0 ; y0) et (x1 ; y1) sont les coordonnées des sommets opposés du rectangle dans lequel est tracée l'ellipse.
- La colonne de pixels d'abscisse x1 et la ligne de pixels d'ordonnée y1 ne sont pas atteintes.
- width, outline et fill sont des attributs optionnels (voir 1) et 2))

Exemple :
ovale1 = canvas_nom.create_oval(75, 25, 425, 375, width=2, outline='#fb8007')
ovale2 = canvas_nom.create_oval(180, 140, 230, 190, width=0, fill='#d05e82')
ovale3 = canvas_nom.create_oval(270, 140, 320, 190, width=0, fill='#d05e82')

4) Texte :

```
canvas_nom.create_text(x0, y0, text = 'texte')
```



- (x0 ; y0) sont les coordonnées de l'ancre du rectangle dans lequel sera écrit la chaîne de caractères 'texte'.
- anchor est un attribut optionnel donnant la position du texte par rapport au point d'ancrage. Défaut : CENTER.
- font est un attribut optionnel donnant la police de caractères suivie de sa taille.

Exemple :
canvas_nom.create_text(254, 90, text='OK Computer', fill='#fb8007', font='Arial 20')

5) Image :

```
im = PhotoImage(file = 'mon_image.gif', master=fen_tk_nom)
canvas_nom.create_image(x0, y0, image = im)
```

- L'image (au format .gif) est chargée grâce à la fonction PhotoImage et un lien relatif.
- (x0 ; y0) sont les coordonnées de l'ancre de l'image.

Exemple : *image.gif doit être téléchargé à partir de GitHub*
im = PhotoImage(file = 'image.gif', master=fen)
logo1 = canvas_nom.create_image(200, 200, image = im)

6) Autres fonctions utiles à appliquer au Canvas :

delete(nom)	Supprime l'item donné en paramètre (*par son **nom**, son identifiant **id** ou son **tag***).
delete(ALL)	Supprime tous les items du canevas.
itemconfigure(nom, para)	Modifie le (*ou les*) paramètre(*s*) désigné(*s*) de l'item **nom**.
itemcget(nom, para)	Retourne la valeur actuelle du paramètre **para** de l'item **nom**.

V. Des clics (et des claques)

Dans une fenêtre programmée à l'aide du module `tkinter`, les actions de l'utilisateur vont pouvoir modifier ou agir sur les widgets. Ces actions, appelées **événements,** sont nécessaires au déclenchement de certaines fonctions.

La syntaxe pour associer des événements à des widgets est la suivante :

```
widget_nom.bind( evenement_nom, fonction_exec )
```

Celle-ci associe l'événement `evenement_nom` au widget (souvent un Canvas) stocké dans la variable `widget_nom`. Au déclenchement de l'événement, la fonction **`fonction_exec`** est exécutée.

La fonction **`fonction_exec`** ne doit avoir qu'un seul argument: **`event`** :

```
def fonction_exec(event):
    """ La fonction « fonction_exec » est associée à un événement event """
```

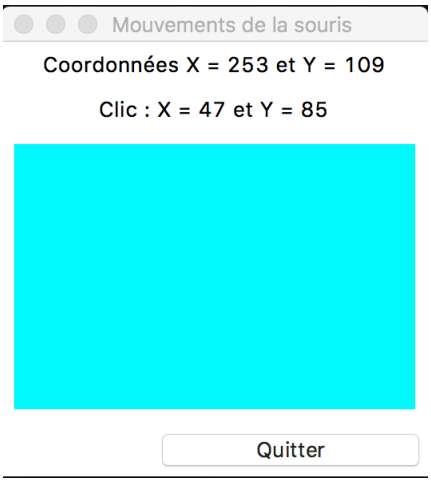
Voici une liste des événements:

	Événement	Description
Souris	'<Motion>'	Mouvement de la souris à l'intérieur du widget.
	'<Button-1>'	Clic (enfoncement) du bouton gauche (1) ou droit (3).
	'<ButtonRelease-3>'	Relâchement du bouton gauche (1) ou droit (3).
	'<Enter>'	La souris passe au-dessus du widget.
	'<Leave>'	La souris «sort» du widget.
Clavier	'<a>'	Appui sur la touche a (par exemple).
	<KeyRelease-h>'	Relâchement de la touche h (par exemple)

Pour les événements clavier, on lie l'événement à la fenêtre elle-même.

1) Position de la souris, position du clic :

Dans cette section, on veut lier les mouvements de la souris au Canvas et afficher en temps réel la position de la souris dans le Canvas



- ☐ télécharger le fichier gui_04.py que vous trouverez sur github.com/bouillotvincent/cours_isn et l'enregistrer sur votre répertoire de travail ;
- ☐ Dans le « programme principal » (*ligne **34***), appliquer la méthode **`.bind()`** au canvas **`dessin`**. Associer l'événement de mouvement de la souris à la fonction **`afficher().`**

Question :

Que se passe-t-il ?

Comment la fonction afficher récupère-t-elle les coordonnées de la souris ? *Écrire la syntaxe ci-dessous :*

- ☐ La fonction **`afficher()`** doit aussi changer la zone de texte **`message`** pour afficher en temps réel les coordonnées de la souris **dans** le Canvas. *Indice : tex.configure…*
- ☐ Finalement, à l'aide de `.grid` et en créant une fonction `afficher_clic()`, ajouter une nouvelle zone de texte où vous afficherez les coordonnées du dernier **clic** de souris *(voir image ci-contre).*

2) Battle of the Clicks.

Nous allons finir notre Battle of the Clicks.

But du « jeu » :

- compter le nombre de clics total ;
- faire apparaitre un cercle de 20 pixels de rayon lorsque l'on clique dans le Canvas et pouvoir change la couleur de ce cercle à l'aide de la bande d'espace.

Mise en place :

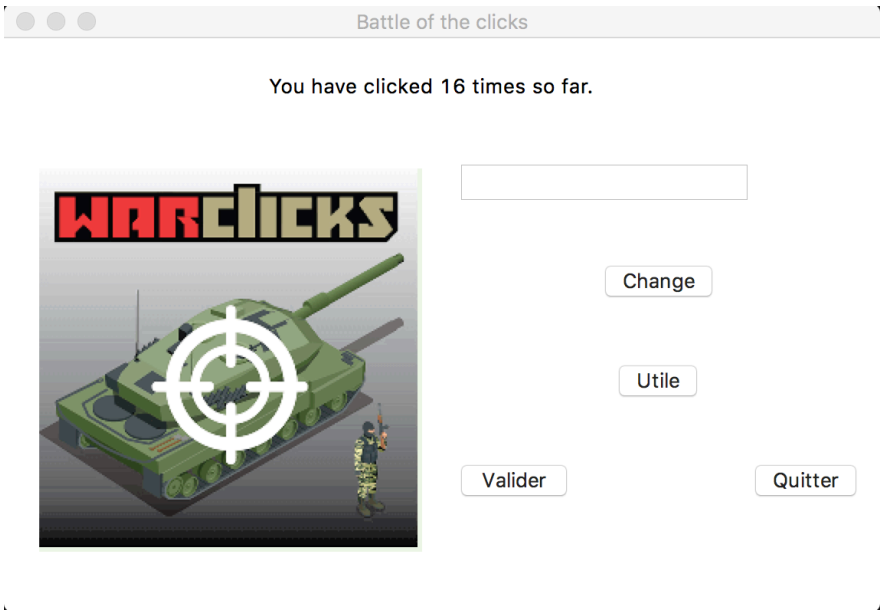
- ☐ dupliquer gui_03.py et renommer cette copie : battle_clicks.py
- ☐ créer un Canvas à la place du widget Label intitulé « Dessin ». Ce Canvas fera 250 pixels de haut et 250 pixels de large.
- ☐ importer une image .gif (chercher warclicks.gif sur GitHub…) avec `PhotoImage`, diviser sa dimension par 2 à l'aide de l'instruction `.subsample(2)` et placer cette image en x = 125, y = 125.

Comptage des clics :

- ☐ à l'aide de `.bind()` et de la fonction compte de l'exercice sur le lancer de dés, afficher dans le `Label` tex1 le nombre total de clics.
  On aura besoin de définir `compteur` comme une variable globale.
- ☐ *Aide : la fonction `do_something` a l'air assez importante…*

Apparition des cercles :

- ☐ créer une fonction `draw_cercle(x, y)` qui, à partir des coordonnées du centre d'un cercle, trace un cercle de 20 pixels de rayon dans le Canvas can. La méthode `create_oval` est votre amie !
- ☐ modifier la fonction compte pour qu'au clic de souris un cercle centré sur les coordonnées de la souris s'affiche.
- ☐ *Aide: Penser à `event.x` et `event.y` …*



Prolongement : Créer une autre fonction qui sera appelée par un appui sur la barre d'espace et qui changera la couleur de nos cercles. On pourra introduire une liste de couleurs prédéfinies.

Ceci constitue la base d'un shoot' em up, comme vous pouvez le constater sur l'exemple au tableau. Vous trouverez également ce code sur le GitHub.