1 Découverte de quelques widgets

1.1 What is a GUI ?

L'exécution d'un programme ne se limite pas à des calculs, elle peut nécessiter des échanges avec les périphériques (et donc l'utilisateur) sous la forme d'Entrés/Sorties (E/S). Jusqu'à présent les E/S de nos programmes s'effectuaient en mode texte. En 1984, le Macintosh d'Apple était le premier ordinateur personnel où l'interface entre l'utilisateur et les programmes (dont le système d'exploitation) se faisait à travers une **interface graphique**.Windows a emboîté le pas avec Windows et des différentes évolutions et les machines Linux utilisent en général X window développé par le MIT (plus une surcouche du type Gnome ou KDE).
L'acronyme anglais pour interface graphique est **GUI**, je vous laisse traduire ...
Pour réaliser, nous aussi, des interfaces d'E/S graphiques, nous allons utiliser le module `tkinter` de la bibliothèque standard de Python. Ce module s'interface avec la bibliothèque graphique TK qui est multiplateformes. Il existe d'autres modules d'interfaces graphiques comme `pyGTK`, `wxPython` mais `tkinter` est le module livré clef en mains et il est bien documenté sur le web.
Pour importer le module `tkinter` sous Python 3 :

```
1  # -*- coding: utf-8 -*-
2 from tkinter import*
```

Dans ce document, je vous propose un rapide survol des possibilités de `tkinter` et des principes fondamentaux. Gardez à l'esprit que je ne suis pas un expert, donc pour des informations plus précises je vous renvoie aux sites suivants :

- Excellent site d'un professeur avec des exemples :
  `http://fsincere.free.fr/isn/python/cours_python_tkinter.php`
- Site de référence pour la documentation (en Anglais) :
  `http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html`
- Le classique tutoriel du site du zéro :
  `http://www.siteduzero.com/informatique/tutoriels/apprenez-a-programmer-en-python/presentation-de-tkinte`
- La documentation officielle Python, pas très fournie :
  `http://docs.python.org/3.3/library/tkinter.html`

Je me suis aussi servi de ouvrages suivants :

- *Apprendre Python 3* de Gérard Swinnen
- *Apprendre la programmation par le jeu, à la découverte du langage Python* de Vincent Maille.

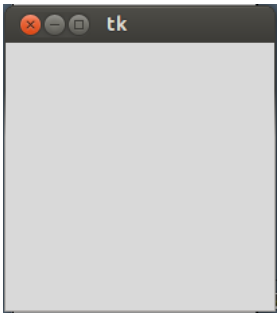
Dernier point, si vous utilisez un environnement comme `Pyzo`, commencez par ouvrir la fenêtre de configuration du `shell` puis choisissez `Tk` dans le menu déroulant des `Gui`.

1.2 Premier exemple et glossaire

Une interface graphique doit comporter au moins une fenêtre racine dans laquelle on va disposer des éléments appelés `widgets`. On dit que ce sont les enfants de la fenêtre racine et qu'elle est leur parent. Un `widget` peut lui même contenir des `widgets` enfants. Les fenêtres et leurs `widget` sont des objets, au sens de la Programmation Orientée Objet, ils possèdent des propriétés ou attributs et des fonctions ou méthodes qui permettent des les manipuler. Lorsqu'on crée un `widget Button` avec `tkinter` on appelle le constructeur de la classe `Button` en lui passant certaines options sous la forme de paramètres nommés.
Il faut toujours préciser en premier le `widget` parent dans lequel on construit notre `widget`.
Dans le code minimal ci-dessous, on crée une fenêtre racine `fen` avec deux `widget` enfants : un `Frame` et un `Label`. Le `widget Frame` possède lui-même deux `widget` enfants : un `Button` et un `Label`.
Les `Label` affichent un texte qui est celui précisé par l'option `text`.
Les `Button` sont des boutons de commande, cliquer dessus déclenche la fonction passée en valeur de l'option `command`. Ici il s'agit de la méthode `destroy` de la fenêtre parente.

```
1 >>> from tkinter import*
2 >>> fen = Tk()
3 >>> a = Frame(fen)
4 >>> b = Label(fen,text='Une étiquette')
5 >>> c = Label(a,text='Une autre étiquette')
6 >>> d = Button(a,text='Quitter',command=fen.quit)
```

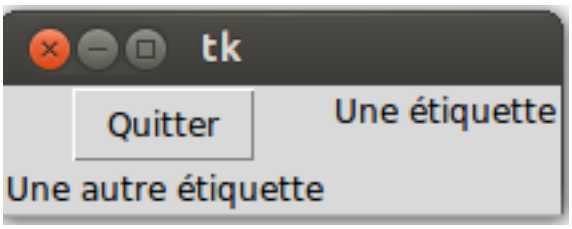
Pour l'instant, on n'obtient rien d'autre qu'une fenêtre vide.



On va d'abord faire apparaître nos `widget` enfants avec une **méthode de placement**. Il en existe trois : `place`, `pack` et `grid`. On en choisit une pour tous les `widget` mais il ne faut pas mélanger les méthodes de placement.
Testons d'abord la méthode `pack` dont l'option `side` possède quatre valeurs possibles : TOP,BOTTOM, RIGHT, LEFT.
Notons que les valeurs des options sont passées soit sous forme d'entiers s'il s'agit de dimensions (en pixels), soit sous forme de constantes `tkinter` en capitales d'imprimeries, soit sous forme de chaîne de caractères, entre quotes et en minuscules.
Ainsi, l'option side de la méthode pack peut prendre (entre autres) la valeur `TOP` ou `'top'`, mais pas `'Top'`.
Avec `pack` ou `grid`, si la géométrie du `widget` parent n'a pas été fixée, c'est le parent qui adapte ses dimensions à celles de ses enfants. En général, si on utilise `pack` ou `grid`, on ne fixe pas les dimensions de la fenêtre parent. On peut néanmoins le faire avec la méthode `geometry`, par exemple avec `fen.geometry('300x100')` où 300 est la largeur en pixels de la fenêtre et 100 sa hauteur.

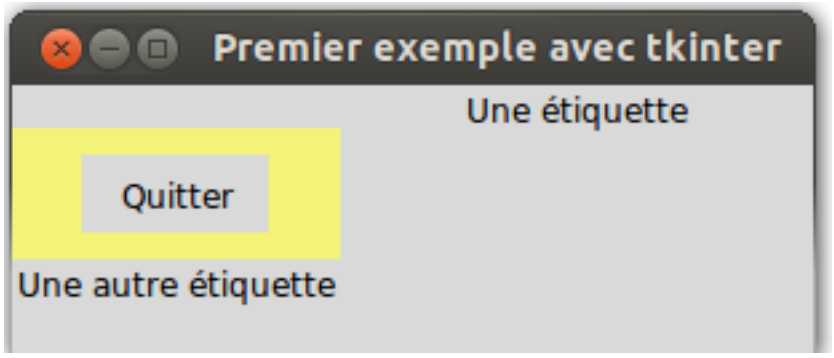
```
>>> a.pack(side=LEFT)
>>> b.pack(side='top')
>>> d.pack(side=TOP)
>>> c.pack(side=TOP)
```

On obtient à peu-près ceci :



On peut mettre un peu d'espace autour des `widget`, par exemple autour du `Button` , avec les options `padx` et `pady` de la méthode `pack`.
On peut aussi modifier des options par défaut de certains `widget` avec la méthode `config` ou avec l'opérateur crochet. Les valeurs des options sont stockées dans un dictionnaire dont les clefs sont les options sous forme de chaîne de caractères.
Pour les couleurs, on peut utiliser les couleurs prédéfinies ('red','black', 'green') ou utiliser la notation hexadécimale (Geany propose un outil palette de couleurs).
Pour avoir la liste des options d'un `widget`, on utilise la méthode `keys`
On peut aussi donner un titre à notre fenêtre racine avec la méthode `title`.
Pour redimensionner la fenêtre racine, on peut utiliser la méthode `geometry` :

```
>>> d.pack(side='top',padx=10,pady=10)
>>> fen.title('Premier exemple avec tkinter')
>>> a.keys()
['bd', 'borderwidth', 'class', 'relief', 'background', 'bg', 'colormap', 'container', '
      cursor', 'height', 'highlightbackground', 'highlightcolor', 'highlightthickness', '
      padx', 'pady', 'takefocus', 'visual', 'width']
>>> d.config(relief='flat')
>>> a['background']='#F3F378'
>>> fen.title('Premier exemple avec tkinter')
>>> fen.geometry('300x100')
```



Si on recopie les commandes précédentes dans un script Python et qu'on l'exécute, contrairement au test dans la console, aucune fenêtre n'apparaît. En fait dans la console, lorsqu'on crée une fenêtre racine avec `Tk()`, **le réceptionnaire d'événements** est automatiquement déclenché.
Le réceptionnaire d'événements est la fonction de `tkinter` qui met en place tous les éléments graphiques et qui scrutent tous les événements qui peuvent se produire dans l'interface graphique (par exemple un clic sur le bouton Quitter). On peut le comparer à une boucle infinie, qui s'exécute en tâche de fonds et qui réagit aux événements déclenchés par l'utilisateur.
Le réceptionnaire d'événements est en général appelé en fin de script, une fois que tous les widgets et toutes les actions liées à des événements ont été définis. L'appel se fait par la méthode `mainloop` de la fenêtre racine.

```
1 >>> fen.mainloop()
```

Méthode *Synthèse des `widget` et des méthodes déjà rencontrés*

- Une fenêtre racine (Top-Level window) est créée avec `fen = Tk()`.

Méthode	Effet
`fen = Tk()`	Crée une fenêtre racine
`fen.title('Titre')`	Donne un titre
`fen.geometry('400x600')`	Redimensionne la fenêtre en 400 pixels de large sur 600 de haut

- Une étiquette de texte est créée avec `a = Label(fen)`.

  On n'est pas obligé d'affecter le `widget` à une variable mais ce peut être utile de disposer d'une référence si on veut y accéder par la suite.

  Lors de la création du widget, on peut préciser dans le constructeur `Label` les valeurs d'un certain nombre d'options.

Options	Effet	Syntaxe
`font`	mise en forme	`font=('Arial',10,'bold','italic')`
`fg`	couleur du texte	`fg='white'` ou `fg='#ffffff'`
`bg`	couleur du fonds	`bg='white'` ou `bg='#ffffff'`
`width, height`	largeur et hauteur	`width=100` ou `height='2cm'`

- Un bouton de commande est créé avec `b = Button(fen)`. Les options `width, height, fg, bg` sont les mêmes que pour un `widget Label` mais on peut lui ajouter l'option `command` pour qu'un clic sur le bouton déclenche une fonction.

Options	Effet	Syntaxe
`command`	précise la fonction à déclencher lors du clic	`command=fonction`

  Attention, il ne faut pas mettre de parenthèse après le nom de la `fonction`.

- Un cadre rectangulaire est créée avec `a = Frame(fen)`. Un `Frame` est surtout utilisé comme conteneur pour d'autres `widget`.
- Méthodes universelles pour les `widget`.

Méthode	Effet
`widget.cget('option')` ou `widget['option']`	Retourne la valeur courante de l'option
`widget.config(option=value)` ou `widget['option']=value`	Change la valeur courante de l'option
`widget.pack(side=value,fill=value, padx=10,pady=15)` où `side=LEFT,TOP,RIGHT,BOTTOM` et `fill='x','y','both','none'`	Méthode de placement géométrique (indispensable pour le rendre visible)

1.3 Méthode de placement `grid` et `widget Canvas`

Méthode *Méthode de placement `grid`*

La méthode `grid` permet de placer les `widget` dans la fenêtre racine en la quadrillant avec une grille. Chaque `widget` est repéré par sa ligne `row` et sa colonne `col` .
Les lignes sont numérotées dans l'ordre croissant de haut en bas et les colonnes de gauche à droite.

Options	Effet
`row=...et column=...`	Numéro de ligne et le numéro de colonne
`rowspan=...et columnspan=...`	Nombre de lignes et de colonnes occupées par le `widget`
`padx=...et pady=...`	Ecart en pixels entre le `widget` et les bords de la grille
`sticky=...`	Côté duquel le `widget` est collé à la grille, valeurs : 'nw','n','ne','w','center','e','sw','s','se'

Méthode *`widget Canvas`*

Un `widget Canvas` est une aire rectangulaire destinée à recevoir des dessins (lignes, polygones), des images ... Un `Canvas` peut contenir d'autres `widget` mais aussi d'autres types d'objets graphiques (ou **items**) comme des lignes, des rectangles, des ovales, des images, des textes ....
Les items sont repérés par leurs coordonnées dans un repère dont l'origine est le coin supérieur gauche du `Canvas`. L'axe des abscisse orienté vers la droite est le côté supérieur et l'axe des ordonnées orienté vers le bas est le côté gauche. Autrement dit le point $(0;0)$ est en haut à gauche, et les abscisses toujours positives et comptées en pixels augmentent en allant vers la droite du `Canvas` tandis que les ordonnées positives augmentent en allant vers le bas.
Les items successivement créés dans un `Canvas` sont rangés dans une pile, c'est-à-dire que le dernier item créé est placé au-dessus de la pile et peut éventuellement dissimulé les items en dessous. En bas de la pile il y a le `Canvas` qui est visible par son fonds (option `'bg'`)

- On crée un `Canvas` de fenêtre parent `fen` avec avec :

  `c = Canvas(fen)`

  On peut de plus préciser des options comme `width,height,bg` ...

- Dans le `Canvas c`, on crée une **ligne** reliant le point $(x_1\,;\,y_1)$ au point $(x_2\,;\,y_2)$ avec

  `ligne = c.create_line(`x_1`,`y_1`,`x_2`,`y_2`,options)`

Options	Effet
`width`	Epaisseur de la ligne en pixels
`fill`	Couleur de la ligne

- Dans le `Canvas c`, on crée un **rectangle** dont le sommet supérieur gauche et le sommet inférieur droit ont pour coordonnées $(x_1\,;y_1)$ et $(x_2\,;y_2)$ avec

`rectangle = c.create_rectangle(`x_1`,`y_1`,`x_2`,`y_2`,options)`

Options	Effet
`width`	Epaisseur de la ligne en pixels
`fill`	Couleur de remplissage du rectangle
`outline`	Couleur du bord

- Dans le `Canvas c`, on crée un **ovale** inscrit dans un rectangle dont le sommet supérieur gauche et le sommet inférieur droit ont pour coordonnées $(x_1\,;y_1)$ et $(x_2\,;y_2)$ avec

`ovale= c.create_oval(`x_1`,`y_1`,`x_2`,`y_2`,options)`

Les options sont les mêmes que pour les rectangles.

Pour créer un cercle il suffit de choisir un carré comme rectangle exinscrit.

- Dans le `Canvas c`, on crée un **texte** au point de coordonnées $(x\,;y)$ avec

`texte = c.create_text(x,y,options)`

Options	Effet
`anchor`	Précise la position d'attache du texte par rapport à (x,y). Valeurs possibles : 'n','e','s' ...comme pour l'option sticky de la méthode grid
`fill`	Couleur du texte
`font`	Paramétrage de la police de caractères (Voir `Label`)
`text`	Chaîne de caractères contenant le texte à afficher

- Dans le `Canvas c`, on peut insérer une image de format `gif` (*Pour d'autres formats d'image il faut utiliser le module ImageTk de PIL*) selon cette procédure :
  - on charge l'image dans une **variable globale** :
    `fichierimg = PhotoImage(file='chemin')`
  - on place l'image dans `c` au point de coordonnées $(x\,;y)$ en remplissant au moins l'option image :
    `texte = c.create_image(x,y,image=fichierimg,anchor='nw')`
- Méthodes des `widget Canvas` :

Méthode	Effet
`c.itemconfig(item,option='valeur')`	Change la valeur d'une option d'un item du `Canvas c`
`c.itemcget(item,option='valeur')`	Accède à la valeur d'une option d'un item du `Canvas c`
`c.tag_raise(item)` `c.tag_lower(item)`	Fait monter (ou descendre) l'item dans la hiérarchie d'affichage du `Canvas`

- `state` est une option commune à tous les items précédents, elle peut prendre trois valeurs NORMAL par défaut, DISABLED qui désactive les réactions de l'item aux clics de souris et HIDDEN qui rend invisible l'item ce qui peut être intéressant.
- Par ailleurs on peut faire référence aux items soit par le nom de variable qui leur a été assigné, soit par leur identifiant, un entier retourné lors de leur création, soit par un tag, une chaîne de caractère identifiant l'item. Pour avoir un tuple de tous les identifiants d'items créés dans l'ordre on peut appeler la méthode `find_all` du `widget Canvas`.

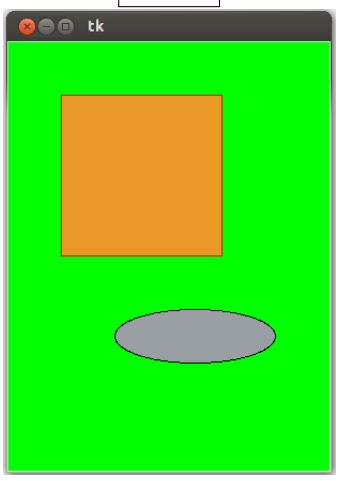
Exercice 1

Tester les commandes suivantes, en console ou dans un script,pour découvrir différentes méthodes des `widget Canvas` :

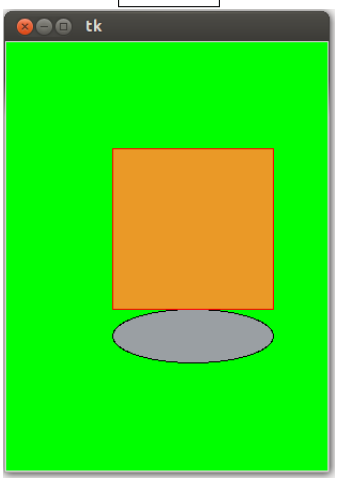
```
1 >>> from tkinter import*
```

```
>>> fen = Tk()  #fenetre racine
>>> c = Canvas(fen,width=300,height=400) #canevas
>>> c.pack(side=LEFT) #placement du canevas
>>> c['bg'] = 'green' #changement de la couleur de fonds
>>> r = c.create_rectangle(50,50,200,200,fill='blue',outline='red') #un rectangle
1
>>> c.create_oval(100,250,250,300,fill='#E74BC8') #un ovale
2
>>> c.find_all() #tuple avec les items créés
(1, 2)
>>> c.itemconfig(r,fill='#EA9927') #modification d'une option d'item
>>> c.itemcget(r,'fill')        #accès à une option d'item
'#EA9927'
>>> c.itemconfig(2,fill='#9A9FA3')
>>> c.itemconfig(r,state=HIDDEN) #rendre un item invisible
>>> c.itemconfig(r,state=NORMAL)
>>> c.coords(r,100,100,250,250) #modifier les coordonnées d'un item
[]
>>> c.delete(r)  #détruire un item
>>> c.delete(2)
>>> img = PhotoImage(file='/home/fjunier/ISN/Cours/CoursJunier/GUI/Tkinter/
images/coccinelle.gif') #référence à un fichier image
>>> c.create_image(50,50,image=img,anchor='nw') #image dans le canevas
3
```

Etape 1



Etape 2



Etape 3

Exercice 2

Réaliser un simulateur de lanceur de dé à 6 faces dont l'interface graphique devra ressembler au modèle ci-dessous.
En plus de la fenêtre racine, il faudra créer :

- un `widget Button` qui commandera une fonction lancer
- un `widget Button` qui permettra de quitter l'application
- un `widget Label` qui affichera le résultat du lancer de dé
- un `widget Canvas` qui contiendra une image de dé (on pourra prendra l'image `de.gif`)

1.4 `widget Entry` et variables de contrôle

Méthode *widget Entry*

On peut créer un champ de saisie de texte sur une ligne avec un `widget Entry` :

```
e = Entry(fen)
```

Ce `widget` possède un certain nombre d'options classiques comme `width,height,font` plus une option `justify` pour la justification du texte qui peut prendre les valeurs LEFT, RIGHT, CENTER.
Evidemment, le texte saisi dans un `Entry` doit pouvoir être récupéré. On, peut alors distinguer deux stratégies :

- on peut le récupérer avec la méthode `get` qu'on applique ainsi : `saisie = e.get()`.
- on peut aussi le capturer dans une variable de contrôle tkinter et la relier à l'`Entry` par son option `textvariable`. De plus cette variable peut être reliée à plusieurs `widget`. Attention, il ne s'agit pas de variables Python comme les autres et il faut les définir avec des commandes spéciales. De plus pour manipuler une variable de contrôle `var` il faut employer des méthodes spéciales :

  `var.get()` pour récupérer sa valeur

  `var.set(valeur)` pour la modifier

```
saisie = StringVar()
e = Entry(fen,textvariable=saisie)
```

variable de controle StringVar()

```
entier = IntVar()
e = Entry(fen,textvariable=saisie)
```

variable de controle IntVar()

Méthodes applicables sur un `widget e = Entry(fen)`

Méthode	Effet
`e.get()`	Retourne le texte saisi

`e.insert(i,texte)`
`e.insert(INSERT,texte)`
`e.insert(END,texte)`
	Insère le texte en position i
	ou à la position du curseur
	ou à la fin

`e.delete(i)`
`e.delete(deb,fin)`
`e.delete(0,END)`
	Efface un morceau du texte saisi en position i,
	la partie entre les positions i et j,
	tout le texte

Exercice 3

Analyser le code du script suivant qui crée une interface graphique de vérification de mot de passe.
Quels sont les `widget` employés ? Comment la variable de contrôle lié au `widget Entry` est-elle utilisée ?

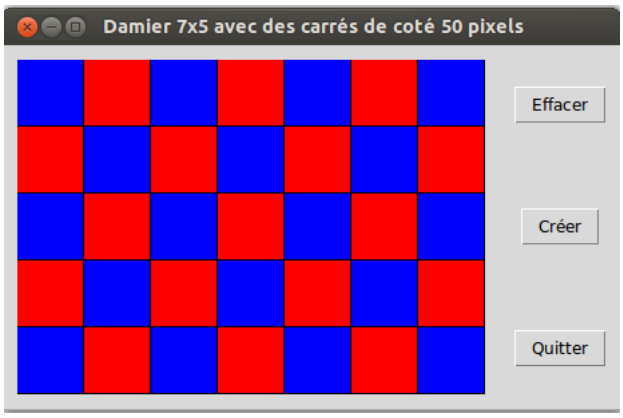
```
# -*- coding: utf-8 -*-
from tkinter import*

def verif():
    """Fonction de vérification du mot de passe"""
    if proposition.get() == password:
        r['text'] = 'Mot de passe correct'
    else:
        r['text'] = 'Mot de passe incorrect'


fen = Tk()

password = 'ISN'

proposition = StringVar()

p = Entry(fen,textvariable=proposition,justify='center')
p.grid(row=1,column=1,padx=10,pady=10)

r = Label(fen,text='Réponse')
r.grid(row=2,column=1,padx=10,pady=10)

b = Button(fen,text='Valider',command=verif)
b.grid(row=1,column=2,rowspan=2,padx=10,pady=10)

fen.mainloop()
```

Exercice 4

Modifier le script de simulateur de dé à 6 faces en rajoutant un `widget Entry` qui permet de saisir un nombre de lancers, le champ de saisie étant rempli par défaut à 1 000.
La valeur retournée après avoir appuyer sur le bouton `simulation` est cette fois la moyenne de tous les lancers obtenus.
On pourra utiliser une variable de contrôle pour le nombre de lancers.

Exercice 5

Réaliser une application avec interface graphique qui dessine un damier de *colonnesxlignes* cases carrées de couleurs alternées. Le nombre de colonnes, de lignes et la taille des cases seront des variables globales du programme.
L'interface pourra ressembler à celle ci-dessous :



1.5 Un exemple avec des menus déroulants

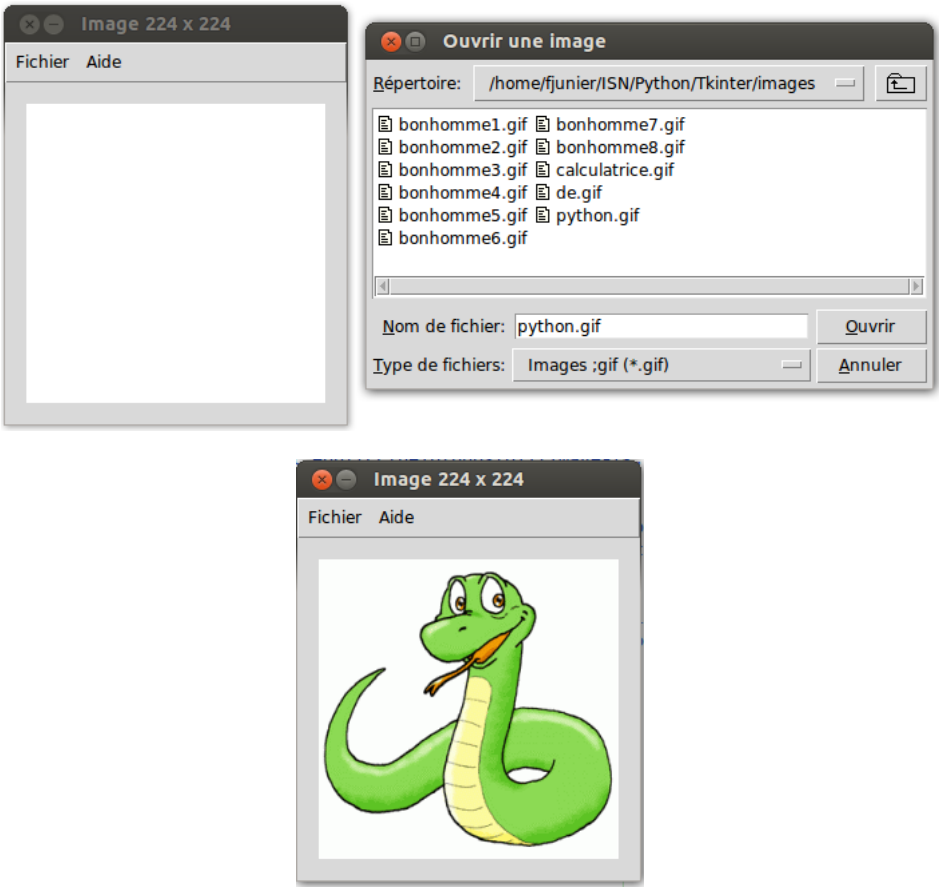
Exercice 6

Tester puis analyser le script suivant (tiré du site de Fabrice Sincère) qui réalise une petite application de `browser` d'images `gif`.
On s'intéressera en particulier à la façon dont on peut créer des menus déroulants (en cascade) et à l'utilisation des sous-modules de tkinter que sont `tkinter.messagebox` pour les boîtes de dialogue pop-up et `tkinter.filedialog` pour la gestion des fichiers.
Autre point intéressant l'utilisation du mot clef `global` devant la variable picture en ligne 15. La variable `picture` reçoit la référence au fichier image ouvert, or cette variable est défini dans l'espace de nommage local de la fonction `ouvrir`. Lorsque l'action de cette fonction est terminée, on retourne dans l'espace de nommage global où une variable locale comme `picture` est à priori inconnue. Dans ce cas la référence au fichier image est perdue et l'image ne peut être affichée dans le `Canvas`. On a donc besoin de transférer la varaible locale picture dans l'espace de nommage global, c'est justement la signification de la déclaration `global picture` au début du corps de la fonction `ouvrir`.

```
1 from tkinter import*
2 #pour l'affichage de fenetres pop up de messages
3 from tkinter.messagebox import*
4 #pour l'affichage de boites de dialogue de gestion de fichiers
5 from tkinter.filedialog import*
6
7 #definition des fonctions commandées par le menu
8
9 def ouvrir():
```

```
    """fonction d'ouverture de fichier
    Utilisation de la fonction askopenfilename du sous-module tkFileDialog
    Il faut charger l'image dans une variable globale afin qu'elle soit référencée  lorsqu'
        on revient au programme principal
    """
    global picture
    fond.delete(ALL) # on efface l'image précédente affichée dans le canevas
    filename= askopenfilename(title='Ouvrir une image',filetypes=[('Images ;gif','.gif'),('
        Tous fichiers','.*')])
    picture= PhotoImage(file=filename)
    fond['width'], fond['height'] = picture.width(),picture.height()
    fond.create_image(0,0,anchor='nw',image=picture)
    racine.title('Image %d x %d'%(picture.width(),picture.height()))

def fermer():
    """fonction de fermeture de fichier"""
    fond.delete(ALL)
    racine.title('Image')

def quitter():
    """Quitter l'application"""
    racine.quit()

def aide():
    """Affichage d'une fenetre pop-up  avec l'adresse d'un site de référence sur Tkinter"""
    showinfo('Site à consulter','http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.
        html')

#création du widget maitre (fenetre racine)
racine = Tk()
#création d'un titre
racine.title('Image')
#autoriser ou non que la fenetre racine soit redimensionnée
racine.resizable(width=False,height=False)


#création de la barre de menu avec le widget Menu
menubar = Menu(racine)
#création d'un menu découlant fichier
menufichier = Menu(menubar,tearoff=0)
menufichier.add_command(label='Ouvrir une image',command=ouvrir)
menufichier.add_command(label='Fermer une image',command=fermer)
menufichier.add_command(label='Quitter',command=quitter)
menubar.add_cascade(label='Fichier',menu=menufichier)
#création d'un menu découlant aide
menuaide = Menu(menubar,tearoff=0)
menuaide.add_command(label='Aide',command=aide)
menubar.add_cascade(label='Aide',menu=menuaide)


#création d'un canevas pour afficher l'image
fond = Canvas(racine,bg='white')
fond.pack(padx=15,pady=15)

#affichage du menu
racine['menu']=menubar

#lancement du réceptionnaire d'événements
racine.mainloop()
racine.destroy()
```

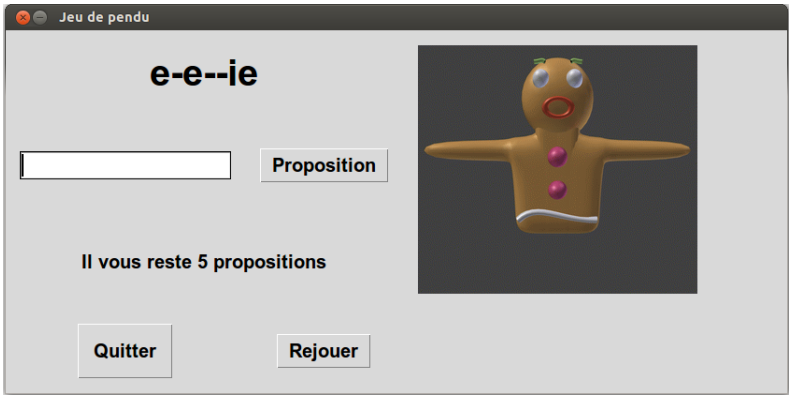
browser d'images



1.6 Mini-projet

Exercice 7 *Mini-Projet 1*

Réaliser une interface graphique pour le jeu de pendu réalisé dans l'exercice 14 du Chapitre Programmation 2.
A chaque nouvelle proposition d'une lettre qui n'est pas dans le mot secret, on affiche une nouvelle image de bonhomme avec une partie du corps en moins. Au bout de sept mauvaises propositions, le joueur a perdu. Les huit images `bonhommex.gif` seront disponibles dans la Dropbox.
L'interface graphique devra ressembler à l'exemple ci-dessous :



2 Gestion du clavier et de la souris, animations

2.1 Exemples de gestion du clavier

Méthode *Evénements et gestion du clavier*

Dans de nombreuses applications comme des jeux, on peut avoir besoin de piloter le déplacement d'un objet au clavier.
Pour celà il faut que le réceptionnaire d'événement (lancé par `fen.mainloop()`) surveille le `widget` contenant l'objet qu'on souhaite piloter et qu'il déclenche une fonction lorsque **l'événement** appui sur un touche de direction se produit.

Le réceptionnaire surveille un `widget` lorsque celui-ci a le `focus`. Dans une fenêtre graphique, on peut sélectionner le `widget` qui a le `focus` en utilisant la touche TAB. Le `widget` avec le `focus` est entouré d'un mince cadre noir.
Par ailleurs on peut forcer le `focus` sur un `widget` avec `widget.focus_force()`.

Les événements claviers peuvent être de différents types, les principaux sont `'<Key>'` abréviation de `'<KeyPress>'`, et `'<KeyRelease>'`. On peut préciser la touche avec la syntaxe `'<Key-Up>'` par exemple s'il s'agit de la flèche de direction vers le haut.
En général on capture un événement `'<Key>'` ou `'<KeyRelease>'` puis on fait le tri entre les différentes touches dans la fonction déclenchée. Pour celà, on peut utiliser l'attribut `keysym` d'un événement

Attribut	Signification
`evt.char`	Caractère correspondant à la touche enfoncée
`evt.keysym`	Symbole correspondant à la touche enfoncée s'il s'agit d'une symbole qui n'est pas un caractère
`evt.keycode`	Entier correspondant au code de la touche enfoncée

Pour un `widget` donné, on peut relier un événement clavier (appui ou relâchement d'une touche) au déclenchement d'une fonction avec la méthode `bind`. La syntaxe est du type suivant :

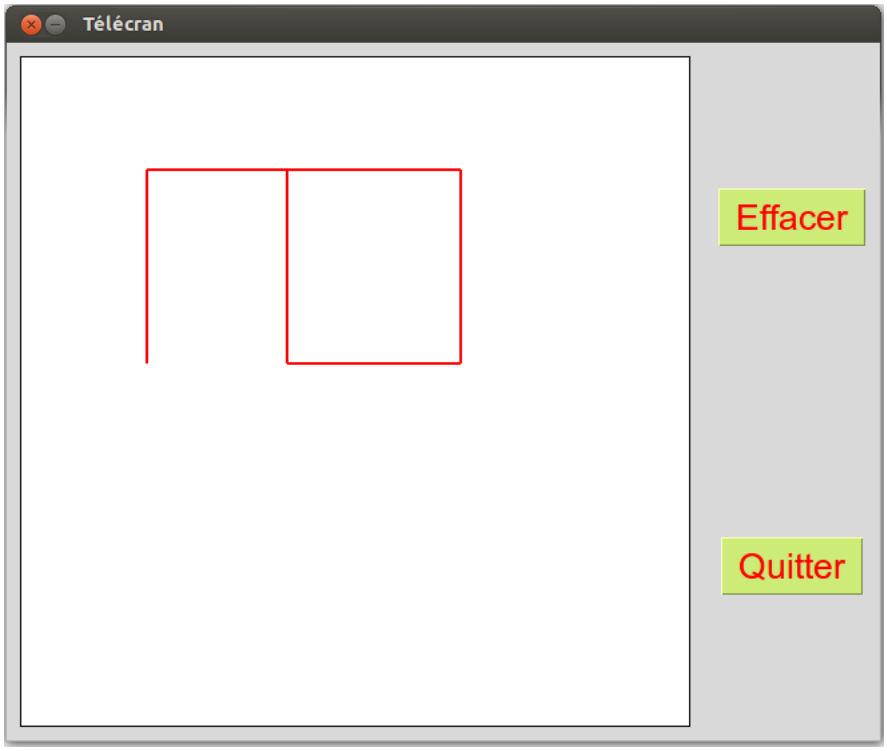
```
1 widget.bind('<Key>',fonction)
2
3 def fonction(event):
4     if event.keysym == 'Up':
5             pass
6     .....
```

On peut aussi lier un événement et une fonction quel que soit le `widget` où l'événement se produit avec la méthode `bind_all` appliquée à la fenêtre racine.

Méthode	Effet
`w.bind('<Key>',fonction))`	L'événement appui sur une touche déclenche fonction si le `widget w` a le focus
`fen.bind_all('<Key>',fonction)`	L'événement appui sur une touche déclenche fonction quel que soit le `widget` de `fen` qui a le `focus`

Exercice 8 *Télécran*

Analyser le script d'un application télécran basique qui permet de piloter un curseur avec juste quatre touches et qui propose un bouton Effacer et un bouton Quitter.




```
# -*- coding: utf-8 -*-
"""Télécran avec Tkinter"""

from tkinter import*

#fonction déclenchée par le réceptionnaire d'événement
def deplacement(event):
    """Déplacement du curseur dans le canevas"""
    global x,y
    print(x,y)
    x0,y0 = x,y
    #on passe n majuscule
    event.keysym = event.keysym.upper()
    if event.keysym=='UP':
        #deplacement vers le haut
        y= max(0,y-dy)
    elif event.keysym=='DOWN':
        #deplacement vers le bas
        y= min(400,y+dy)
    elif event.keysym=='LEFT':
        #deplacement vers la gauche
        x= max(0,x-dx)
    elif event.keysym=='RIGHT':
        #deplacement vers la droite
        x= min(400,x+dx)
    fond.create_line(x0,y0,x,y,width=2,fill='red')

def clear():
    """Efface le canevas"""
    fond.delete(ALL)

#Variables globales
#x ey y sont les coordonnées du point courant sur le canevas
#dx et dy sont les déplacements élémentaires
x,y = 200,200
dx,dy = 5,5
```

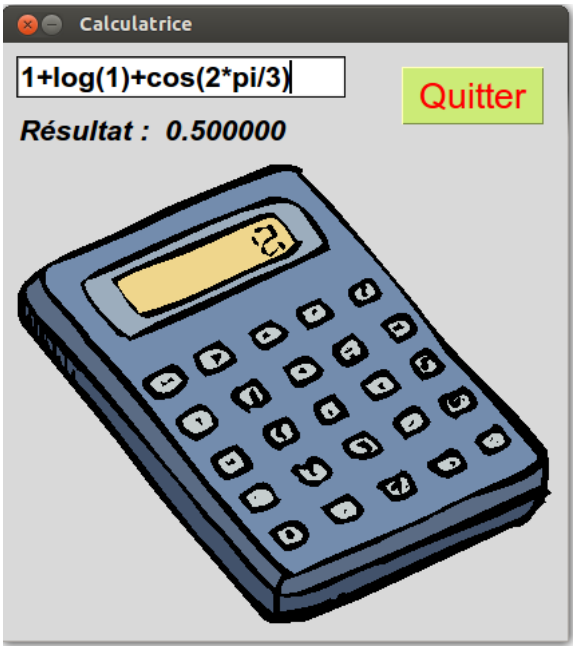
```
37
38 #création du widget maitre (fenetre racine)
39 fen = Tk()
40 #création d'un titre
41 fen.title('Télécran')
42 #autoriser ou non que la fenetre racine soit redimensionnée
43 fen.resizable(width=False,height=False)
44
45 #création de widgets esclaves
46 #Widget Canvas où se feront les dessins
47 fond = Canvas(fen,width=500,height=500,bg='white')
48 fond.grid(row=1,column=1,rowspan=2,pady=10,padx=10)
49 #on force le focus sur le widget Canvas, on peut aussi changer le focus avec la touche tab
50 fond.focus_force()
51 #si on appuie sur une touche, le gestionnaire d'événement deplacement est déclenché
52 fond.bind('<Key>',deplacement)
53
54 #widget Button pour effacer le Canevas
55 effacer = Button(fen,fg='red',bg='#CCEB77',text='Effacer',font=('Arial',20),command=clear)
56 effacer.grid(row=1,column=2,padx=10,pady=10)
57
58 #widget Button pour quitter
59 quitter = Button(fen,fg='red',bg='#CCEB77',text='Quitter',font=('Arial',20),command=fen.quit
       )
60 quitter.grid(row=2,column=2,padx=10,pady=10)
61
62 #lancement du réceptionnaire d'événements
63 fen.mainloop()
64 #lorsqu'on sort de la boucle du réceptionnaire d'événement
65 #on détruit la fenetre
66 fen.destroy()
```

Télécran

2.2 Mini-Projet

Exercice 9 *Mini-Projet 2*

Réaliser une application calculatrice qui affiche le résultat du calcul saisi dans un `widget Entry`, lorsqu'on appuie sur la touche `Return`.
On utilisera le `module math` et la fonction `eval` .
L'interface graphique devra ressembler au modèle ci-dessous (image `calculatrice.gif` disponible dans la Dropbox) :



2.3 Exemple de gestion de la souris

Méthode *Gestion de la souris*

Comme pour le clavier, on peut définir pour certains `widget` des fonctions qui se déclenchent lorsqu'ils ont le focus et qu'un certain événement souris se produit. On utilise de même la méthode `bind` du `widget`.
Quelques événements souris (bouton gauche codé par 1, bouton central par 2 et le droit par 3) :

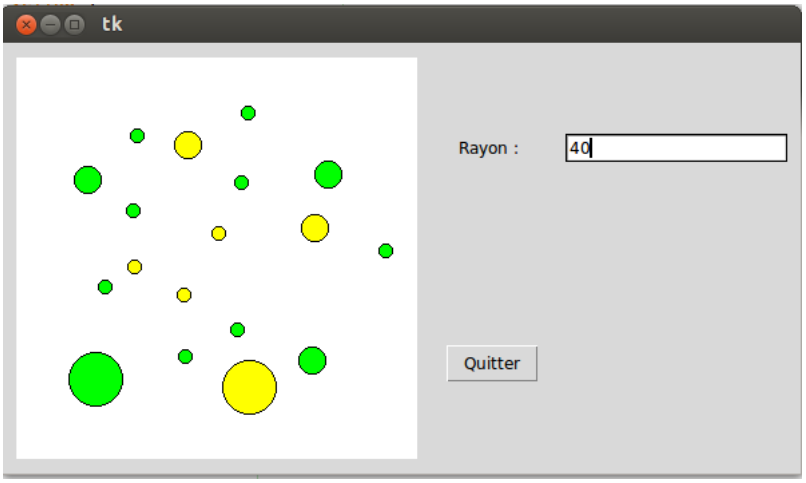
Evénement	Déclenché par
`'<Motion>'`	Le déplacement de la souris'
`'<ButtonPress-1>'`	Une pression sur le bouton gauche
`'<ButtonRelease-1>'`	Un relâchement du bouton gauche
`'<Enter>'`	L'entrée de la souris sur le `widget`
`'<Leave>'`	La sortie de la souris du `widget`

Un événement déclenché par la souris ne possède pas d'attributs `event.keysym` ou `event.keycode` mais les attributs suivants :

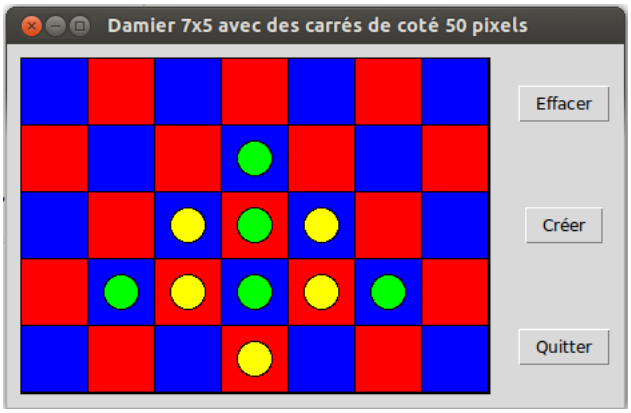
- `event.x` et `event.y` pour la position de la souris lors du clic.
- `event.num` pour le numéro du bouton de souris responsable du clic.

Exercice 10

1. Ecrire un script Python pour une interface graphique avec un `widget Canvas` où s'affiche un disque jaune (bouton gauche 1) ou vert (bouton droit 3) de centre le point cliqué avec la souris et de rayon la valeur saisie dans un `widget Entry`.



2. Adapter le script précédent et celui du damier (exercice 5) pour afficher un disque jaune (bouton gauche 1) ou vert (bouton droit 3) centré dans la case où l'on clique.



2.4 Exemple d'animation sans controle

Méthode *Méthode* `after`

On va exposer deux méthodes d'animation à partir de l'exemple d'une boule qui rebondit sur les bords d'un `Canvas fond` selon la loi des chocs élastiques. Il s'agit d'une animation sans contrôle une fois qu'elle est lancée.

- un `widget scalebox` permet de choisir la vitesse horizontale avant de démarrer l'animation
- un `widget spinbox` permet de choisir la vitesse verticale avant de démarrer l'animation
- un bouton `Démarrer` permet de démarrer l'animation
- un bouton `Pause` permet de la mettre en pause
- un bouton `Reprendre` permet de la reprendre après une pause
- un bouton `Quitter` permet de quitter l'application

Pour réaliser cette animation on peut procéder de deux façons :

1. On peut écrire une fonction `animation2` avec une boucle qui met à jour les coordonnées de la boule, puis qui met à jour le `Canvas fond` avec sa méthode `update`.

```
fond.update()
```

L'inconvénient majeur de cette méthode est qu'elle rend inactifs tous les autres `widget` de la fenêtre qui n'ont pas d'effet direct sur la boucle d'animation. Par exemple, le bouton `Quitter` qui déclenche la méthode quit de la fenêtre `racine`,

sera inactif pendant toute l'animation. Ce n'est pas le cas du bouton `Pause` capable de changer la valeur de la variable `enpause` qui contrôle le test de la boucle d'animation.

Le code de cette fonction est inactif dans notre exemple. Pour l'activer il suffit de remplacer `animation()` par `animation2()` dans les fonctions `demarrer` et `reprendre`.

2. 0n peut écrire une fonction `animation` qui met à jour les coordoonnées de la boule, puis qui se rappelle elle-même si la variable `enpause` est à `False` avec la méthode `after` de la fenêtre `racine`.

```
racine.after(20,animation)
```

Notons qu'`after` est une méthode de la fenêtre racine et non du `widget Canvas` comme `update`.

Le paramètre 20 représente le **délai de rafraichissement** : la fonction `animation` est rappelée au bout de 20 millisecondes. `animation` est donc une **fonction récursive**.

Le délai de rafraichissement joue sur la vitesse d'animation puisque s'il est réglé sur 20 ms et que le déplacement horizontal élémentaire est `dx = 5` (en pixels) alors la vitesse de déplacement horizontal est de :

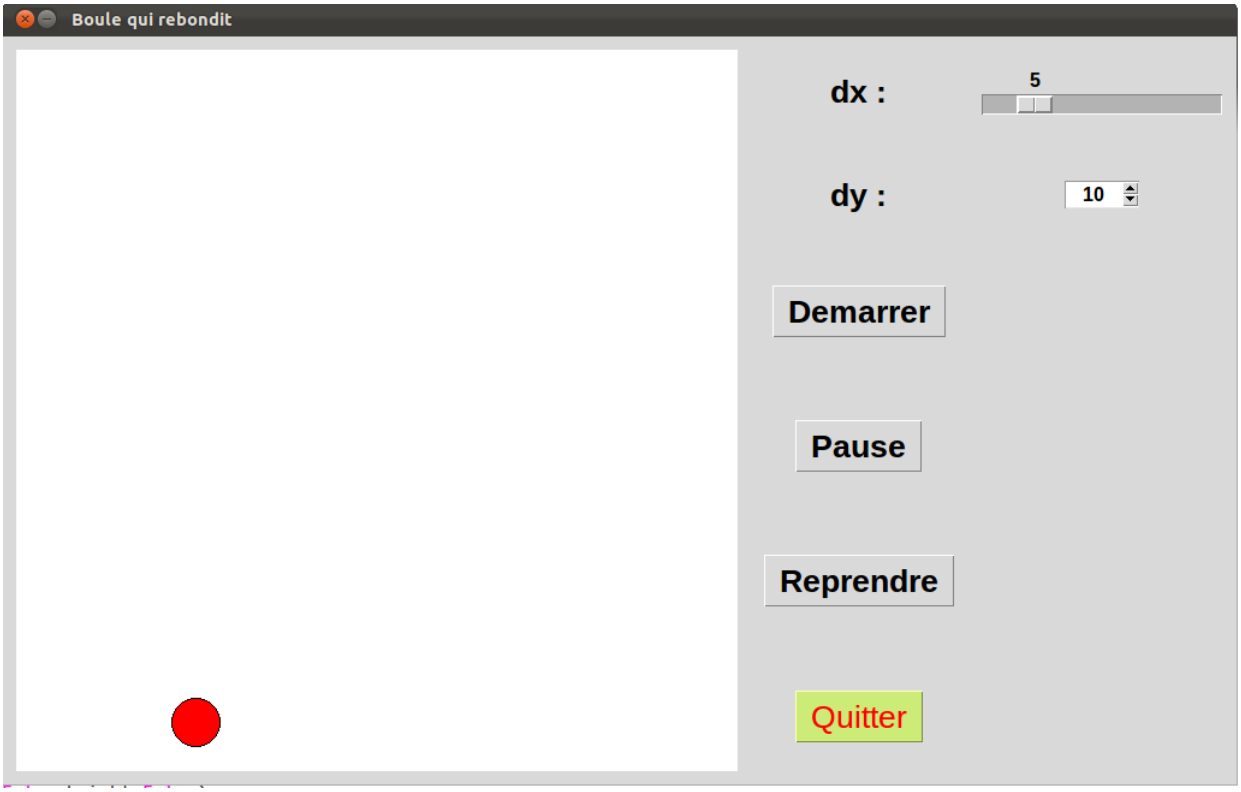
$$v = \frac{5}{20 \times 10^{-3}} = 2,5 \times 10^2 pixels/s$$

Mais si ce paramètre est de 50 ms alors, pour le même déplacement horizontal élémentaire, la vitesse de déplacement horizontal est de :

$$v = \frac{5}{50 \times 10^{-3}} = 1 \times 10^2 pixels/s$$

L'avantage de la méthode `after` est qu'elle ne désactive pas les autres `widget`. Par exemple on peut cliquer sur le bouton `Quitter` pour quitter l'application pendant l'animation.

Nous utiliserons donc de préférence cette méthode pour nos animations.



Exercice 11

Analyser le script de l'animation sans contrôle d'une balle qui rebondit sur les bords d'un `Canvas`.

1. Quels sont les `widget` employés ?
2. Quelles sont les variables globales ? A quoi sert la variable `enpause`, comment-est elle modifiée ?

3. Des variables de contrôle `tkinter` sont-elles utilisées ?

4. Comment la fonction animation, modifie-t-elle la vitesse de la boule ? On pourra se remémorer les principes de base de la physique des chocs élastiques.

```
# -*- coding: utf-8 -*-
from tkinter import*

#fonction déclenchée par les boutons de commande
def animation():
    """animation de la boule avec la méthode after"""
    global x,y,dx,dy
    print(dx,dy)
    #si la boule atteint le bord gauche ou le bord droite
    #son déplacement élémentaire horizontal dx est changé en son opposé
    if x+dx>W-rayon or x+dx<rayon:
        dx = -dx
    #si la boule atteint le bord supérieur ou inférieur
    #son déplacement élémentairevertical dy est changé en son opposé
    if y+dy>H-rayon or y+dy<rayon:
        dy = -dy
    x,y = x+dx,y+dy
    #on change les coordonnées de la boule
    fond.coords(boule,x-rayon,y-rayon,x+rayon,y+rayon)
    if not enpause:
        racine.after(20,animation)

def animation2():
    """animation de la boule avec la méthode update"""
    global x,y,dx,dy
    while not enpause:
        #si la boule atteint le bord gauche ou le bord droite
        #sa vitesse horizontale dx est changée en son opposé
        if x+dx>W-rayon or x+dx<rayon:
            dx = -dx
        #si la boule atteint le bord supérieur ou inférieur
        #sa vitesse verticale dy est changée en son opposé
        if y+dy>H-rayon or y+dy<rayon:
            dy = -dy
        x,y = x+dx,y+dy
        #on change les coordonnées de la boule
        fond.coords(boule,x-rayon,y-rayon,x+rayon,y+rayon)
        fond.update()

def demarrer():
    """variable enpause mise à False
    et lancement de l'animation"""a
    global enpause,x,y,dx,dy
    print(dx,dy)
    x,y = round(W/2),round(H/2)
    #initialisation des vitesses horizontale dx et verticale dy
    if tkvar_dx.get() != dx:
        dx = tkvar_dx.get()
    if tkvar_dy.get() != dy:
        dy = tkvar_dy.get()
    enpause = False
    animation2()

def pause():
    """variable enpause mise à True"""
    global enpause
```

```
    enpause = True

def reprendre():
    """variable enpause mise à False"""
    global enpause
    if enpause:
        enpause = False
    animation2()


#Variables globales toutes initialisées par défaut
#W,H et sont la largeur et la hauteur du Canevas
#x et y sont les coordonnées du centre de la boule sur le canevas
#enpause est un booleen indiquant si l'animation est en pause
#dx et dy sont les déplacements élémentaires
#rayon est le rayon par défaut, en pixels, de la boule
W,H=800,800
x,y = round(W/2),round(H/2)
enpause = True
dx,dy = 5,5
rayon = round((W+H)/60)
print(rayon)

#création du widget maitre (fenetre racine)
racine = Tk()
#création d'un titre
racine.title('Rebonds')
#autoriser ou non que la fenetre racine soit redimensionnée
racine.resizable(width=False,height=False)


#variables de controle, vitesse de la boule
tkvar_dx = IntVar()
tkvar_dy = IntVar()

#création de widgets esclaves

#Widget Canvas où se feront les dessins
fond = Canvas(racine,width=W,height=H,bg='white')
fond.grid(row=1,column=1,rowspan=6,pady=10,padx=10)
#création de la boule
boule = fond.create_oval(x-rayon,y-rayon,x+rayon,y+rayon,fill='red')

#widget Label, message pour indiquer à l'utlisateur qu'il peut modifier dx
labeldx= Label(racine,text='dx :',font=('Arial',20,'bold'),justify='center')
labeldx.grid(row=1,column=2,padx=10)

#widget Scale pour modifier dx
scaledx = Scale(from_=0,to=30,resolution=5,orient='horizontal',variable=tkvar_dx,font=('
      Arial',12,'bold'),length=200)
scaledx.grid(row=1,column=3,padx=10)

#widget Label, message pour indiquer à l'utlisateur qu'il peut modifier dy
scaledy = Label(racine,text='dy :',font=('Arial',20,'bold'),justify='center')
scaledy.grid(row=2,column=2,padx=10)

#widget Spinbox pour modifier dy
spindy = Spinbox(from_=0,to=30,increment=5,textvariable=tkvar_dy,font=('Arial',12,'bold'),
      justify='center',width=5)
spindy.grid(row=2,column=3,padx=10)

```

```
116 #widget Button pour démarrer
117 boutondemarrer = Button(racine,text='Demarrer',font=('Arial',20,'bold'),command=demarrer)
118 boutondemarrer.grid(row=3,column=2,padx=10,pady=10)
119
120 #widget Button pour faire une pause
121 boutonpause = Button(racine,text='Pause',font=('Arial',20,'bold'),command=pause)
122 boutonpause.grid(row=4,column=2,padx=10,pady=10)
123
124 #widget Button pour reprendre
125 boutonreprendre = Button(racine,text='Reprendre',font=('Arial',20,'bold'),command=reprendre)
126 boutonreprendre.grid(row=5,column=2,padx=10,pady=10)
127
128
129 #widget Button pour quitter
130 boutonquitter = Button(racine,fg='red',bg='#CCEB77',text='Quitter',font=('Arial',20),command
      =racine.quit)
131 boutonquitter.grid(row=6,column=2,padx=10,pady=10)
132
133
134 #lancement du gestionnaire d'événements
135 racine.mainloop()
136
137 racine.destroy()
```

boule qui rebondit

2.5 Exemple d'animation avec controle

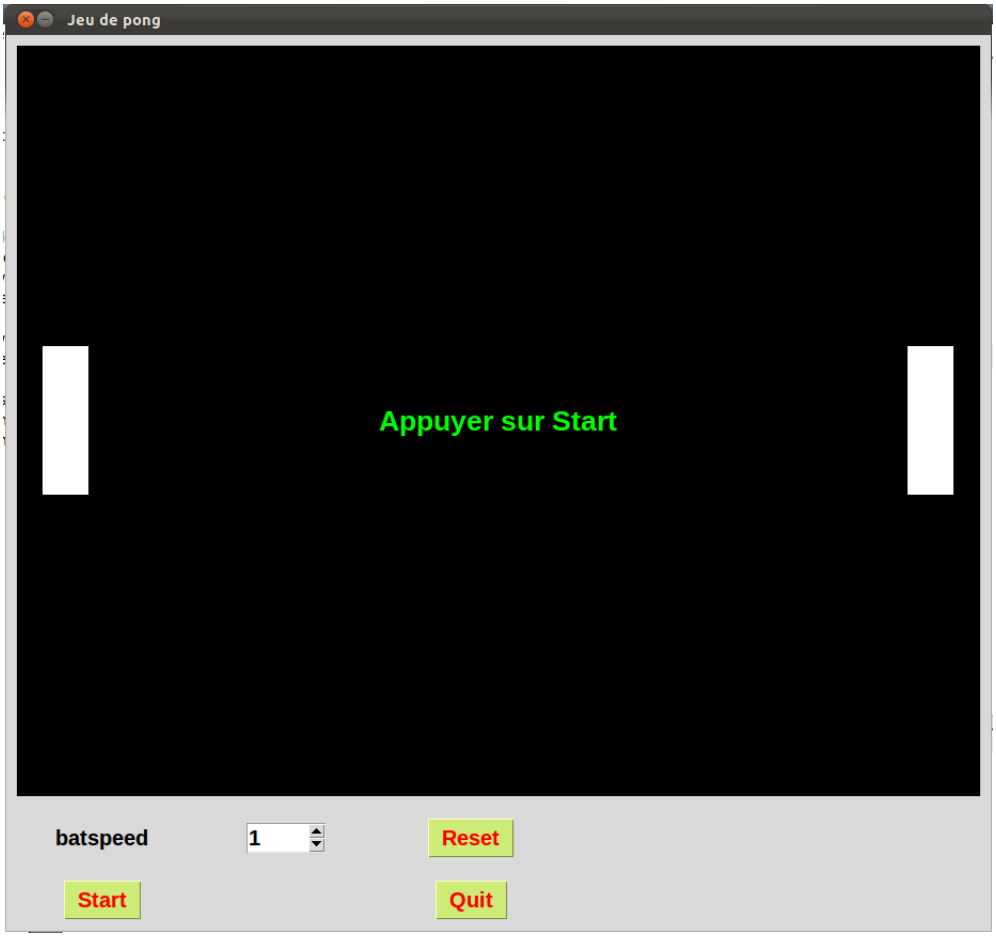
Exercice 12

On veut développer une petite application avec interface graphique qui dessine deux rectangles blans sur le fond noir d'un `Canvas` et qui permet de déplacer verticalement ces rectangles avec les touches du clavier :

- les touches 'UP' et 'DOWN' pour déplacer une raquette
- les touches 'A' et 'Q' pour déplacer l'autre raquette
- on peut de plus régler la vitesse des raquettes avec un `widget spinbox`.

1. Analyser le script ci-dessous : identifier les `widget` utilisés et représenter l'algorithme d'animation des raquettes sous la forme d'un diagramme.
2. Compléter le corps des fonctions `anim_bat1` et `anim_bat2` chargées d'animer le déplacement de chacune des raquettes.
   Les raquettes ne doivent pas sortir des limites du `Canvas`.

Un prolongement naturel de ce code, si on assimile les rectangles à des raquettes, est la programmation d'un jeu de Pong.




```
# -*- coding: utf-8 -*-
from tkinter import*
############Fonctions déclenchées par des événements############
def reset():
    """réinitialisation des variables"""
    global bath,batw,bat1x,bat1y,bat2x,bat2y
    gamezone.itemconfig(affichage,state='normal')
    #largeur de raquette
    batw = W//20
    #hauteur de raquette
    bath = H//5
    #position du centre de la raquette 1 nommée bat1
    bat1x, bat1y = batw, H//2
    #position du centre de la raquette 2 nommée bat2
    bat2x, bat2y = W-batw, H//2
    #liste des touches appuyées
    touches = []
    #on dessine les raquettes bat1 et bat2
    gamezone.coords(bat1,bat1x-batw//2,bat1y-bath//2,bat1x+batw//2,bat1y+bath//2)
    gamezone.coords(bat2,bat2x-batw//2,bat2y-bath//2,bat2x+batw//2,bat2y+bath//2)

def start():
    """on lance une nouvelle partie"""
    gamezone.itemconfig(affichage,state='hidden')
    #on lance la boucle d'animation
    animation()


def pressed(event):
    """on rajoute la touche pressée dans la liste touches
    si elle n'y est pas déjà"""
```

```
    t = event.keysym.upper()
    if not t in touches:
        touches.append(t)

def released(event):
    """on enlève la touche relachée de la liste touches
    """
    t = event.keysym.upper()
    if t in touches:
        touches.remove(t)

def anim_bat1():
    """animation de la raquette du joueur 1"""
    global bat1x,bat1y
    vitesse = int(var_batspeed.get()*H//50)
    ............

def anim_bat2():
    """animation de la raquette du joueur 2"""
    ....................;

def animation():
    """boucle d'animation"""
    #animation de la raquette 1
    anim_bat1()
    #animation de la raquette 2
    anim_bat2()
    #si la partie n'est pas terminée on continue
    window.after(30,animation)


########création des widgets de l'interface graphique##############
###############création du widget maitre (fenetre racine)##############
window = Tk()
#création d'un titre
window.title('Jeu de pong')
#autoriser ou non que la fenetre racine soit redimensionnée
window.resizable(width=False,height=False)

#############Variables de controle#######
#vitesse de déplacement des raquettes
var_batspeed = IntVar()

#####Variables globales##########
#largeur et hauteur de la gamezone en pixels
W,H = 900,700
#largeur de raquette
batw = W//20
#hauteur de raquette
bath = H//5
#position du centre de la raquette 1 nommée bat1
bat1x, bat1y = batw, H//2
#position du centre de la raquette 2 nommée bat2
bat2x, bat2y = W-batw, H//2
#liste des touches appuyées
touches = []

################################widgets esclaves#############
#Ecran widget Canvas
gamezone = Canvas(window,width=W,height=H,bg='black')
gamezone.grid(row=1,column=1,columnspan=8,padx=10,pady=10)
```

```
93
94  #raquette1 (une raquette de ping-pong s'appelle bat en anglais)
95  bat1 = gamezone.create_rectangle(bat1x-batw//2,bat1y-bath//2,bat1x+batw//2,bat1y+bath//2,
        fill='white')
96  #raquette2 (une raquette de ping-pong s'appelle bat en anglais)
97  bat2 = gamezone.create_rectangle(bat2x-batw//2,bat2y-bath//2,bat2x+batw//2,bat2y+bath//2,
        fill='white')
98
99
100 #affichage au centre du Canevas
101 affichage = gamezone.create_text(W//2,H//2,text='Appuyer sur Start',font=('Arial',20,'bold')
        ,fill='green')
102
103 #liens entre événements survenant sur le widget gamezone et des fonctions
104 gamezone.bind('<KeyPress>',pressed)
105 gamezone.bind('<KeyRelease>',released)
106
107 #Label pour le widget Spinbox suivant
108 Label(text='batspeed',font=('Arial',15,'bold')).grid(row=2,column=1,padx=5,pady=5)
109 #widget Spinbox pour controler la vitesse de déplacement vertical des raquettes
110 batspeed = Spinbox(from_=1,to=5,increment=1,textvariable=var_batspeed,font=('Arial',15,'bold
        '),width=5)
111 batspeed.grid(row=2,column=2,padx=5,pady=5)
112
113 #widget Button pour reset
114 redemarrer = Button(window,fg='red',bg='#CCEB77',text='Reset',font=('Arial',15,'bold'),
        command=reset)
115 redemarrer.grid(row=2,column=3,padx=10,pady=10)
116
117 #widget Button pour démarrer
118 demarrer = Button(window,fg='red',bg='#CCEB77',text='Start',font=('Arial',15,'bold'),command
        =start)
119 demarrer.grid(row=3,column=1,padx=10,pady=10)
120
121 #widget Button pour quitter
122 quitter = Button(window,fg='red',bg='#CCEB77',text='Quit',font=('Arial',15,'bold'),command=
        window.quit)
123 quitter.grid(row=3,column=3,padx=10,pady=10)
124
125 #lancement du gestionnaire d'événements
126 window.mainloop()
127 window.destroy()
```

Déplacement de raquettes

2.6 Mini-projet

Exercice 13 *Mini-Projet n°3*

Réaliser une application de jeu de morpion (ou Tic-Tac-Toe) avec interface graphique où deux joueurs peuvent s'affronter. Comme prolongement, on pourra proposer une version où un joueur affronte l'ordinateur.

2.7 Idées de projet

Exercice 14 *Projet n°1*

Réaliser un jeu de Pong avec interface graphique

Exercice 15 *Projet n°2*

Réaliser un jeu de Snake avec interface graphique : un serpent peut être déplacé dans un `Canvas` quadrillé en cases, où sont disposées aléatoirement de la nourriture ou des obstacles. Lorsque la tête du serpent passe sur une case de nourriture, il grandit d'une case, si la tête touche un bord du `Canvas` ou une case avec un obstacle, le serpent meurt.

Exercice 16 *Projet n°3*

Faire une recherche documentaire sur *le jeu de la vie* inventé par le mathématicien John Horton Conway. Réaliser une programme de simulation du *jeu de la vie* à partir d'une population initiale qu'on peut définir en cliquant sur les cases d'une grille.

Table des matières