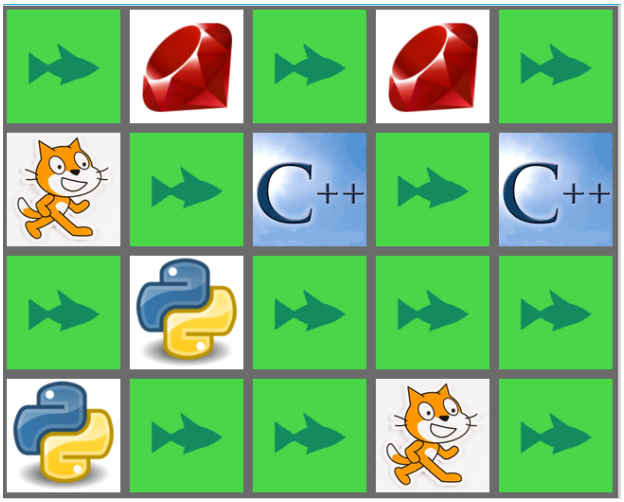
Pascal ORTIZ



Tutoriel Tkinter : jeu memory

Version du 17 janvier 2020

Table des matières

Présentation du jeu memory

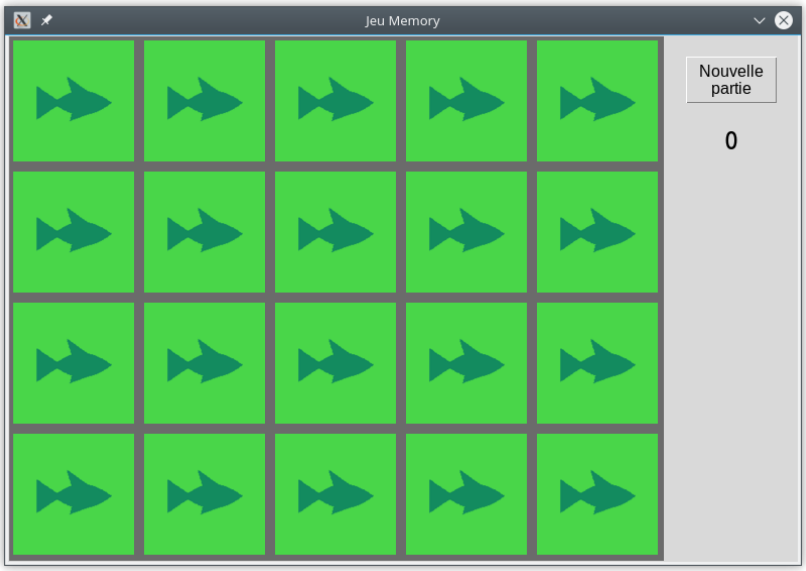
Si vous n'avez jamais joué au *jeu memory* ou si vous ne voyez pas de quoi je parle, le plus simple est d'essayer le jeu. Sur Internet, on pourra y jouer, par exemple, aux liens ci-dessous :

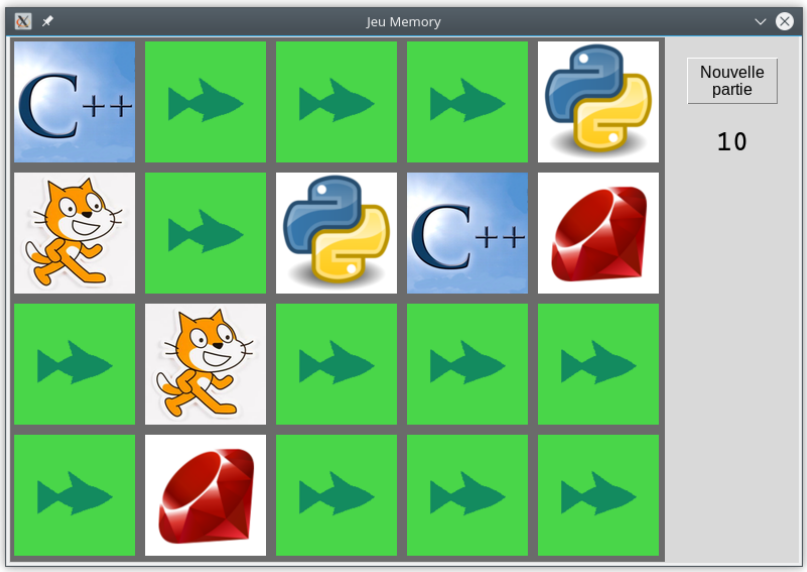
— Jeu memory : animaux

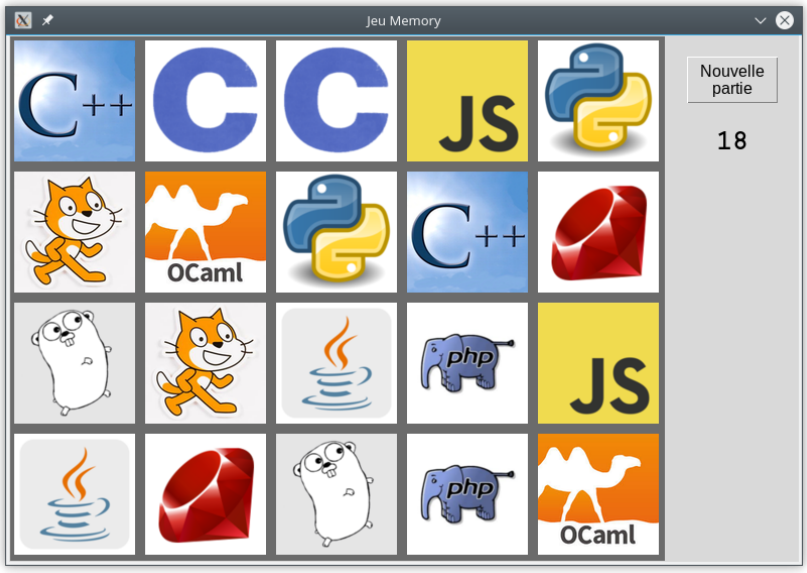
— Jeu memory : fruits

Sous Android, on peut y jouer avec Picture Match.

Tkinter est la bibliothèque officielle du langage Python pour construire des interfaces graphiques. Dans ce tutoriel, je vous propose de découvrir Tkinter en implémentant un memory game. Le code fonctionne pour Python 3.5 et au-delà. Les images seront des logos de langages de programmation. Ci-dessous, on peut voir le jeu :







Au cas où vous ne connaîtriez pas le jeu, en voici une description. Le jeu memory se joue avec un plateau de quelques dizaines de cartes représentant des motifs, par exemple des fruits. Au début du jeu, les cartes sont faces cachées. Le joueur retourne une paire de cartes. Si les deux cartes représentent la même image, elles restent faces visibles. Sinon, le joueur les retourne pour que leurs faces restent cachées. Le joueur retire alors une autre paire et il continue ainsi jusqu'à ce que toutes les cartes aient été découvertes. Le jeu existe bien sûr en version numérique.

Il peut y avoir des variantes :

— les cartes sont montrées pendant quelques secondes avant le début du jeu
— le temps de recherche est plafonné
— certaines images peuvent être présentes 3, 4, etc fois au lieu de juste 2
— etc.

Le jeu numérique

Le joueur choisit (ie retourne) successivement deux cartes en cliquant dessus. Si la 2e carte est identique à la première, elles restent définitivement dans cette position, faces visibles.

La première choisie reste visible tant que la 2e n'a pas été découverte.

Si la 2e carte est différente de la 1re, les deux cartes restent visibles environ une demi seconde puis se replacent automatiquement face cachée.

Remarque importante. On conviendra dans notre implémentation que *tant que ces deux faces sont en attente d'être retournées, les autres cartes sont bloquées.* Cette contrainte simplifie la formulation des règles et le codage du jeu.

Le code

La difficulté du codage du jeu numérique provient essentiellement de l'animation à gérer : il faut arriver à distinguer les deux clics de souris puis à effectuer le retournement de cartes (s'il doit se faire), le tout sans bug !

Je suppose que vous êtes déjà initié à Python, et que vous avez un niveau au moins équivalent à celui que propose mon cours d'introduction et donc que vous avez pratiqué Python quelques mois. J'utiliserai parfois des notions plus avancées que celles de mon cours telles que des listes en compréhension, imbriquées éventuellement.

Il est essentiel d'avoir bien intégré toutes les abstractions que des variables et le découpage en fonctions permettent de réaliser. Vous devez également être fluide concernant l'usage du hasard en Python et des boucles `for` (imbriquées éventuellement).

Le tutoriel ne suppose pas que vous connaissiez la programmation objet, qu'il n'utilise pas d'ailleurs, et cela pour permettre au plus grand nombre de suivre le tutoriel. Toutefois, la fin du tutoriel proposera un code de jeu en utilisant une classe, ce qui est la façon la plus appropriée d'écrire des jeux en Tkinter.

Toutes les variables et noms de fonctions sont écrits en anglais (sauf dans certains mini-programmes d'illustration).

Les images

Les images du jeu que nous allons coder représentent les logos des dix langages de programmation suivants, par ordre alphabétique :

C, CPP, GO, JAVA, JS, OCAML, PHP, PYTHON, RUBY, SCRATCH.

Il y a 11 images, au format gif, de forme carrée, de taille 120 pixels. En plus des 10 logos, il y a une image pour le dos des cartes, de couleur verte, avec un poisson au centre (mémoire oblige !). Je me référerai à cette carte en parlant de carte de « couverture » (*cover*).

J'ai choisi le format gif, pourtant ancien et limité, et non le format png car je ne suis pas certain que sous macOS, même sous Python 3.6, le format png soit complètement reconnu par Tkinter. Toutefois, à partir de la version Python 3.7, cela ne semble plus être le cas.

Un fichier Python du jeu **ne doit pas** être placé *à l'intérieur* du dossier d'images mais à l'extérieur, en « voisin » du dossier.

1 Le canevas

Création d'une fenêtre et d'un widget

Une application graphique de bureau s'exécute dans une fenêtre qui contient des widgets. Un widget est juste un composant graphique comme un menu ou un bouton. Voici un code "Hello Tkinter !" qui crée une fenêtre et un widget sous Tkinter :

fenetre_widget.py

```
1  from tkinter import *
2
3  WIDTH=600
4  HEIGHT=400
5
6  my_root=Tk()
7  cnv=Canvas(my_root, width=WIDTH, height=HEIGHT, background='ivory')
8  cnv.pack()
9
10 my_root.mainloop()
```

ce qui ouvre l'« application » suivante

- Ligne 1 : on importe toutes les fonctionnalités de Tkinter même si on n'a pas besoin de toutes celles-ci. C'est pratique pour de petits programmes. On procède légèrement autrement dans des usages plus avancés.
- Ligne 6 : on créé une fenêtre Tkinter en appelant le constructeur `Tk`, dite fenêtre « maîtresse » (*master*).
- Ligne 7 : on crée un « canevas » dans la fenêtre : c'est un widget permettant d'effectuer du graphisme, des animations, etc.
- Ligne 8 : on indique avec `pack` comment le widget doit être intégré dans le widget-maître (ici la fenêtre `my_root`).
- Ligne 10 : on lance la « boucle principale » (*mainloop*) de l'application. C'est typique de la programmation événementielle.

Important

Pour créer un widget (ligne 7), on utilise le constructeur approprié, par exemple ici `Canvas`. Ensuite, on passe des arguments à ce constructeur :

— le **premier argument** est le widget-maître qui va contenir le widget qu'on va construire. Par exemple, pour le canevas (ligne 7), le premier argument est la fenêtre `my_root`.

— les **autres arguments** sont les **options** de construction du widget. Ces options sont nommées, c'est-à-dire de la forme `option=truc`, où option est, par exemple, une dimension comme `width`, une option de couleur telle `background`, etc.

Remarques sur l'écriture et l'exécution d'une interface graphique

— Les interfaces graphiques sont construites par **emboîtement de widgets** comme des boutons, des entrées, des menus, etc. Le conteneur qui n'est contenu dans aucun autre conteneur est souvent référencé (ligne 6) par une variable appelée `root`, `master`, `window`, etc.

— Il est courant de placer en début de programme des variables écrites en majuscules (lignes 3 et 4) et qui représentent des données invariables du programme, comme certaines dimensions, le chemin vers certaines resources, etc.

— Si on retire la dernière ligne du programme (ligne 10), la fenêtre ne sera pas créée : l'application graphique tourne forcément dans la `mainloop`.

— Si on écrit du code après la ligne contenant le lancement de la mainloop (ligne 10), il ne sera pas exécuté, à moins de tuer la fenêtre de l'application graphique.

— Beaucoup d'éléments d'une application graphique sont des objets au sens de la programmation objet. Voilà pourquoi on utilise des notations pointées comme `cnv.pack()` (ligne 8 ou 10) pour faire des appels de méthodes.

— La ligne 6 est indispensable : on est obligé d'indiquer explicitement que l'on construit une fenêtre racine.

— La ligne 8 (ou un équivalent) est aussi indispensable : il faut indiquer au toolkit comment un widget est placé dans son contenant. Ici, le placement est effectué avec le gestionnaire `pack`.

— Notre programme ici est en attente d'événements. Mais, comme il est minimal, le seul événement qu'il peut enregistrer est un clic de fermeture sur la croix de la fenêtre.

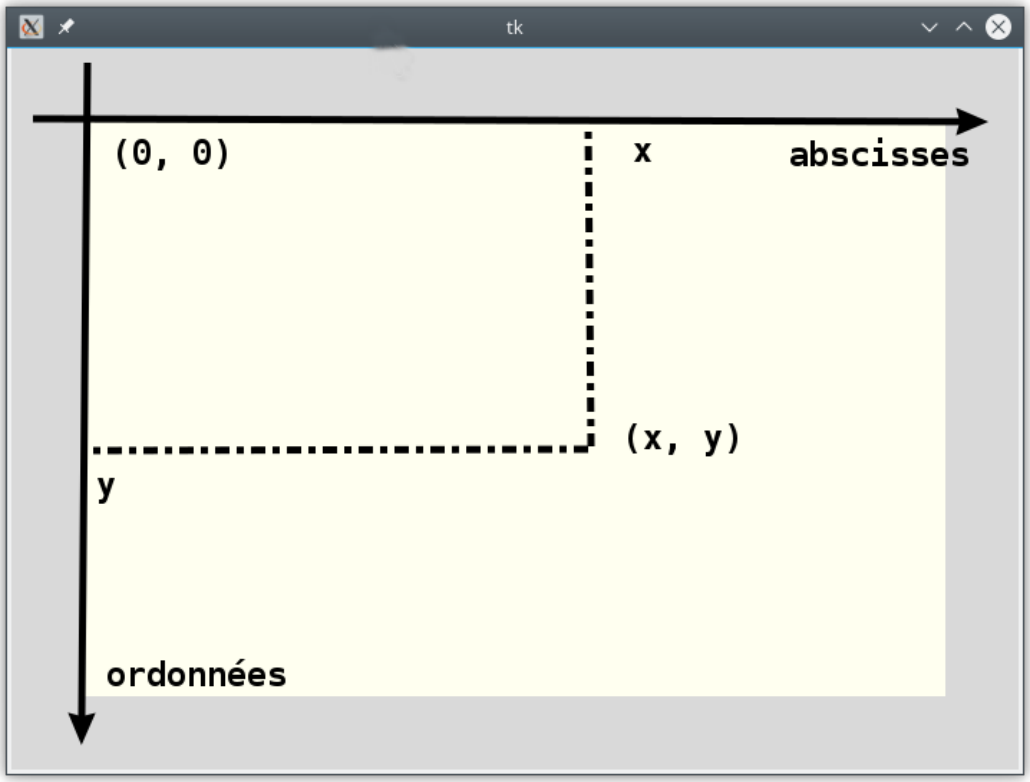
— Programmer une interface graphique n'est pas incompatible avec la lecture de message dans la console produit avec la fonction `print` (ça permet de déboguer un minimum).

Repérage dans un canevas

Soit le code suivant construisant un canevas :

```
from tkinter import Tk, Canvas

root=Tk()
cnv=Canvas(root, width=200, height=200, bg="ivory")
cnv.pack()
root.mainloop()
```

Un canevas (défini par le constructeur `Canvas`) est muni d'un système de repérage (invisible) dans lequel chaque point du canevas a deux coordonnées :



Attention, le système de repérage est orienté **différemment** du système utilisé en mathématiques :

— l'origine est le coin en haut à gauche
— l'axe des abscisses est, comme en math, l'axe horizontal orienté de gauche à droite
— l'axe des ordonnées est, comme en math, un axe vertical mais il est orienté vers le bas (assez logique vue la configuration, non ?).

Les coordonnées repèrent des pixels. Pour être plus précis, le premier pixel est numéroté 0, le suivant 1, etc. Si un canevas est créé avec l'option `width=200` le pixel le plus à droite du canevas est numéroté 199.

Inclusion d'images sur le canevas

Le canevas supporte le placement d'images présentes sur une unité de stockage. Seuls les formats

— png (depuis Python 3.4 sous Windows et peut-être Python 3.6 pour Mac Os)
— gif

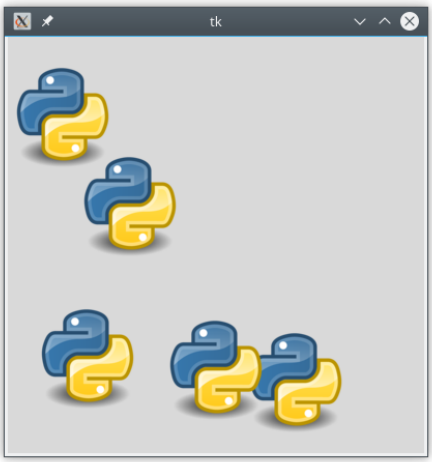
sont pris en charge.
Exemple de code :

inclusion_images.py

```
1 from tkinter import *
2 from random import randrange
```

```
3
4 SIDE=400
5 root = Tk()
6 cnv = Canvas(root, width=SIDE, height=SIDE)
7 cnv.pack()
8
9 logo = PhotoImage(file="python.gif")
10
11 for i in range(5):
12     centre= (randrange(SIDE),randrange(SIDE))
13     cnv.create_image(centre, image=logo)
14
15 root.mainloop()
```

et qui affiche



Le code ci-dessus place aléatoirement 5 images sur un canevas.

Avant de pouvoir utiliser une image (ligne 13), il faut la convertir dans un format propre à Tkinter. Pour cela (ligne 9), on doit utiliser la classe `PhotoImage` en lui précisant l'adresse du fichier image sur le disque. Une fois l'image convertie par appel à `PhotoImage`, on peut l'incorporer dans un canevas en utilisant la méthode du canevas appelée `create_image`. Comme les dimensions d'une image sont invariables, il suffit de donner à `create_image` les coordonnées du point où on souhaite que le centre de l'image se trouve sur le canevas.

Remarquer qu'on a besoin de créer juste **une seule** instance de `PhotoImage` même si elle sert à la création de **plusieurs** images.

L'image ne s'adapte pas automatiquement au canevas où l'image est placée, si l'image est plus large que le canevas, elle ne sera qu'en partie visible dans le canevas.

Noter qu'il existe une syntaxe peut-être moins lisible pour désigner les coordonnées du centre de l'image : au lieu d'écrire `cnv.create_image(c, image=logo)` après avoir défini la variable `c=(x,y)`, on peut écrire `cnv.create_image(x, y, image=logo)`.

Pour le support du png sous macOS, voir IDLE and tkinter with Tcl/Tk on macOS.

Identifiant d'image

Tout image placée sur un canevas Tkinter et créée avec la méthode du canevas `create_image`, lors de la création, reçoit une id unique. Cette id est un identifiant entier strictement positif. Il permet de garder trace de l'image créée.

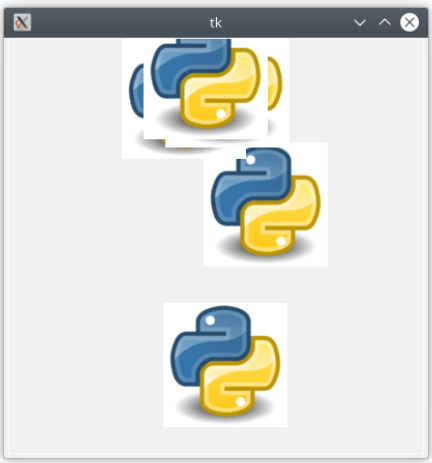
Exemple :

id_images.py

```
from tkinter import *
from random import randrange

SIDE=400
root = Tk()
cnv = Canvas(root, width=SIDE, height=SIDE)
cnv.pack()

logo = PhotoImage(file="python.gif")

for i in range(5):
    x, y= randrange(SIDE),randrange(SIDE)
    id_image=cnv.create_image(x, y, image=logo)
    print(id_image)

root.mainloop()
```

```
1
2
3
4
5
```

qui produit :



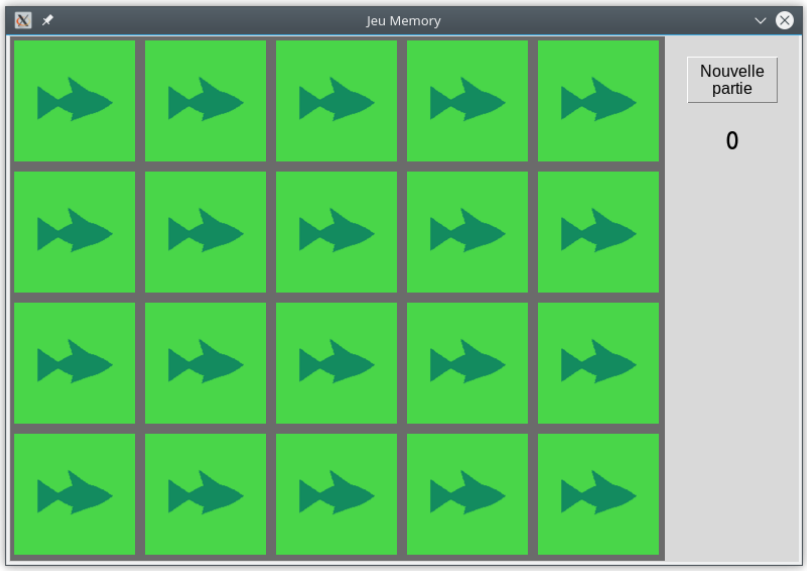
Dans ce code, on fait apparaître, à des endroits aléatoires, 5 images sur le canevas. L'appel à

`cnv.create_image` est récupéré dans une variable et qui contient alors l'id de l'image créée. Les ids sont alors affichées. On remarque que les id ne sont pas des entiers aléatoires mais qu'il s'agit d'entiers consécutifs croissants.

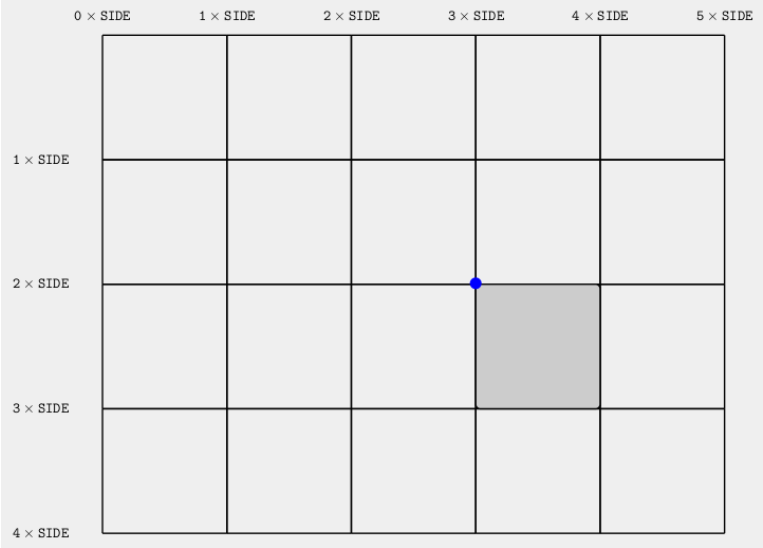
Placement des images en grille

Dans un memory game, les images sont placées sur une grille. L'utilisation de grilles est très courante dans l'implémentation de jeux.

On veut réaliser le placement suivant (sans le bouton ni le label) :



Ici, chaque image est carrée, de taille `PICT_SIZE = 120` pixels. En réalité, on va ignorer cette dimension et chaque image sera placée dans un carré plus grand de taille `SIDE = 130`. Il y a donc un espacement `PAD=10` pixels entre les deux. La grille est formée de `NB_LINES=4` lignes et de `NB_COLS=5` colonnes. Désormais, on ne raisonne plus que sur les grands carrés de côté `SIDE`. Chaque image est placée au centre de l'emplacement, on ne calcule donc que les centres des carrés. La figure ci-dessous aide à comprendre la configuration :



Fixons notre attention sur la carte grisée. En regardant la grille comme un tableau 2D avec des indices de lignes et de colonnes commençant à 0, notre carte est d'indices `(line, col) = (2,3)`. Il est clair alors que le coin supérieur gauche de la carte (en bleu sur le dessin) a pour coordonnées $(3 \times \texttt{SIDE}, 2 \times \texttt{SIDE})$. Plus généralement, l'image en positions `line` et `col`, est telle que son coin supérieur gauche a pour coordonnées $(\texttt{col} \times \texttt{SIDE}, \texttt{line} \times \texttt{SIDE})$. Par suite le centre de cette image est de coordonnées :

$$\texttt{centre} = (\texttt{X0} + \texttt{col} \times \texttt{SIDE}, \texttt{Y0} + \texttt{line} \times \texttt{SIDE}).$$

où $(\texttt{X0}, \texttt{Y0}) = (\texttt{SIDE}//2, \texttt{SIDE}//2)$.

Pour le placement de toutes les images de la grille, typiquement, il est effectué avec deux boucles `for` imbriquées.

Voici le code pour placer uniquement la face cachée de chaque image :

grille_couvertures.py

```
from tkinter import *

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=4
NB_COLS=5

WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
X0=Y0=SIDE//2

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray')
cnv.pack()

```

```
18 cover = PhotoImage(file="./images/cover.gif")
19
20 # Placement des images
21 for line in range(NB_LINES):
22     for col in range(NB_COLS):
23         centre=(X0+col*SIDE, Y0+line*SIDE)
24         cnv.create_image(centre, image=cover)
25
26 root.mainloop()
```

— Lignes 24 : création des images. Le centre de l'image située à la ligne `line` et à la colonne `col` a pour coordonnées `(X0+col*SIDE, Y0+line*SIDE)`.

La vue et le modèle

Le point qui suit est ESSENTIEL même s'il ne prend pas beaucoup de place : il oriente votre façon de coder.

En programmation graphique, il est très important de distinguer la partie visible de l'application et ce qu'elle représente avec ses données et son traitement. Ainsi, nous allons bien séparer :

— les données de visualisation (la vue, ie Tkinter)

— le traitement de ses données (le modèle).

Pour notre jeu, pendant tout le déroulement de la partie, le programme disposera d'un tableau 2D (une liste de listes), nommé `board`, indexé par ligne et colonne, et représentant l'état de la grille visible par le joueur. Typiquement, `board` pourra avoir le contenu suivant :

```
1 [[2, 3, 6, 3, 9], [9, 4, 4, 7, 5], [8, 1, 7, 2, 0], [1, 8, 6, 0, 5]]
```

On voit bien que chaque numéro figure deux fois dans la grille.

En particulier, figureront dans ce tableau les cartes qui ont déjà été appariées et les cartes qui sont encore faces cachées. C'est aussi `board` qui fournira, en début de partie, le plateau mélangé. Les cartes seront placées dans le tableau avec une valeur entière, comme dans l'exemple ci-dessus.

Suppression d'items du canevas

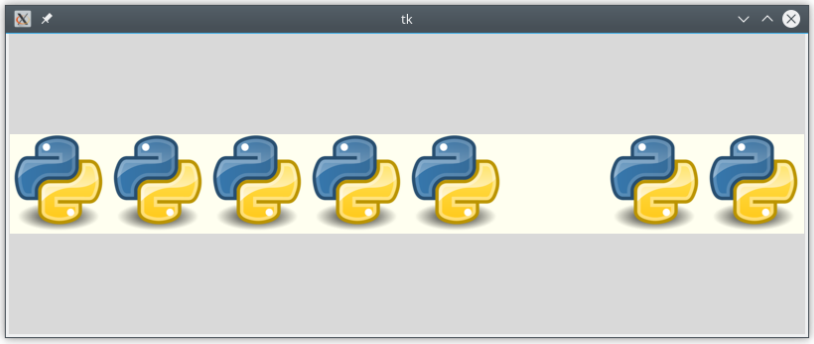
Il est possible de supprimer n'importe quel objet créé sur le canevas. Pour cela il suffit de connaître l'id de l'objet à supprimer et d'utiliser la méthode `delete` du canevas. Illustrons en supprimant une image d'une rangée d'images :

`suppression_image.py`

```
1 from tkinter import *
2 from random import randrange
3
4 NB_IMG=8
5 SIDE=100
6 WIDTH=SIDE*NB_IMG
7 X0=Y0=SIDE//2
8
```

```
9  root = Tk()
10 logo = PhotoImage(file="python.gif")
11 cnv = Canvas(root, width=WIDTH, height=SIDE, bg="ivory")
12 cnv.pack(pady=100)
13
14 ids=[]
15 for k in range(NB_IMG):
16     id_image=cnv.create_image(X0+k*SIDE, Y0, image=logo)
17     ids.append(id_image)
18
19 j=randrange(NB_IMG)
20 print(j)
21 mon_id=ids[j]
22
23 cnv.delete(mon_id)
24
25 root.mainloop()
```

```
26 5
```



On utilise une même image `python.gif` carrée de taille `SIDE=100`. Dans la boucle (lignes 15-17) :

— on dessine côte à côte `NB_IMG=8` images

— on mémorise l'id de chaque image dans une liste `ids`

Puis, on choisit au hasard une id d'image (ligne 19 et ligne 20 pour l'affichage de l'indice) et on supprime du canevas l'objet ayant cet id (ligne 23). Comme on le voit sur la copie d'écran, il y a bien un trou dans l'alignement des images à l'indice choisi.

Mélanger la grille

La partie qui suit est **indépendante** de Tkinter. Le plateau contient 20 images qui se regroupent en 10 paires. Dans le tableau 2D `board`, chaque image sera représentée par un entier entre 0 et 9, ce qui fait 10 entiers, chaque entier apparaissant deux fois.

Il faut donc construire un tableau 2D `board` d'entiers mélangés. Une fois ce tableau créé, on pourra l'utiliser pour dessiner la grille du jeu.

Voici un code qui génère `board` :

melanger_board.py

```
1  from random import shuffle
2
3  NB_LINES=4
4  NB_COLS=5
5  NB_PICT=NB_LINES*NB_COLS // 2
6
7  L=[v % NB_PICT for v in range(2*NB_PICT)]
8  shuffle(L)
9  print(L)
10
11 board=[]
12 k=0
13 for line in range(NB_LINES):
14     row=[]
15     for col in range(NB_COLS):
16         row.append(L[k])
17         k+=1
18     board.append(row)
19
20 print(board)
```

```
21 [2, 3, 6, 3, 9, 9, 4, 4, 7, 5, 8, 1, 7, 2, 0, 1, 8, 6, 0, 5]
22 [[2, 3, 6, 3, 9], [9, 4, 4, 7, 5], [8, 1, 7, 2, 0], [1, 8, 6, 0, 5]]
```

On commence par générer une liste d'entiers aléatoires (ligne 7). Pour cela, on place, sous forme de nombres, toutes les cartes

0 0 1 1 2 2 ... 9 9

dans une liste provisoire `L`. Puis, on mélange cette liste avec la fonction `shuffle` (attention à la façon de le faire, cf. lignes 8 : `L` est modifiée). On lit le résultat ligne 21.

Il ne reste plus qu'à placer ligne par ligne et colonne par colonne cette liste mélangée `L` dans le tableau 2D `board` (cf. lignes 13-18 et affichage ligne 22).

Pour un code moins chargé, il était possible de définir `board` en une seule ligne de code en utilisant des listes en compréhension imbriquées puis de remplir :

```
1 board=[[None for col in range(NB_COLS)] for line in range(NB_LINES)]
2 k=0
3 for line in range(NB_LINES):
4     for col in range(NB_COLS):
5         board[line][col]=L[k]
6         k+=1
```

voire de définir `board` directement :

```
board=[[L[line*NB_COLS+col] for col in range(NB_COLS)]
                            for line in range(NB_LINES)]
```

Pour un code plus propre, écrivons une fonction, même si elle est un peu factice puisqu'elle ne prend pas de paramètres :

make_board.py

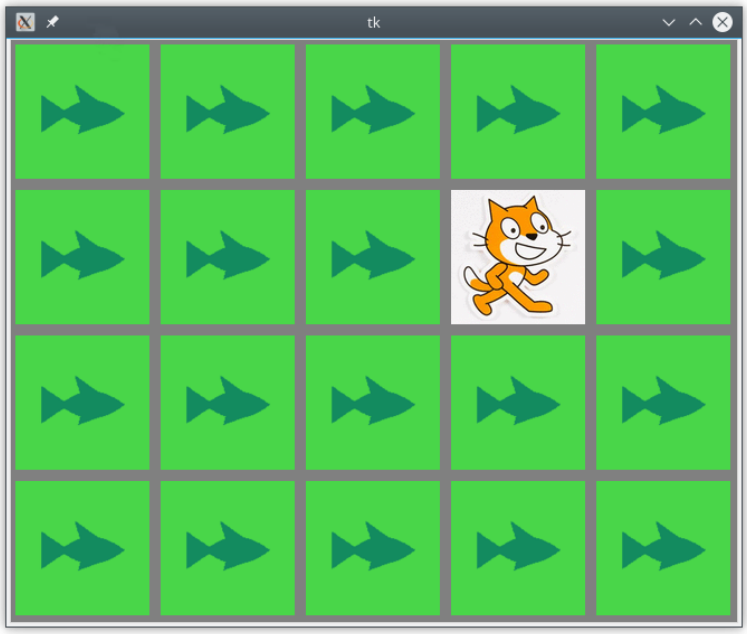
```
from random import shuffle


NB_LINES=4
NB_COLS=5
NB_PICT=NB_LINES*NB_COLS // 2

def make_board():
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

print(make_board())
```

```
[[4, 3, 9, 2, 1], [1, 0, 7, 5, 8], [5, 9, 8, 4, 2], [6, 6, 7, 0, 3]]
```

Dessiner la grille complète

On va maintenant placer dans le jeu toutes les images. Pour cacher les figures, le dos de chaque image est placé au-dessus de la figure. Pour rendre visible une carte, on fera disparaître le dos de l'image. Et puis pour cacher l'image, il faudra replacer au-dessus un dos d'image. On comprend donc qu'il faut disposer d'un identifiant pour chaque dos d'image. Il va donc falloir récupérer dans un tableau 2D les ids des différentes images de couverture.

Le code ci-dessous va produire une figure comme celle-ci :



Le plateau ci-dessus contient les 20 figures (invisibles sauf une) recouvertes par l'image de couverture (ce qui fait au total 40 images à placer sur le canevas). Pour que la réalisation soit bien claire, à la fin de la construction des figures cachées, le programme choisit au hasard une ligne et une colonne de la grille et retire l'image de couverture, ce qui explique le logo visible sur la copie d'écran.

Voici le code :

grille_toutes_images.py

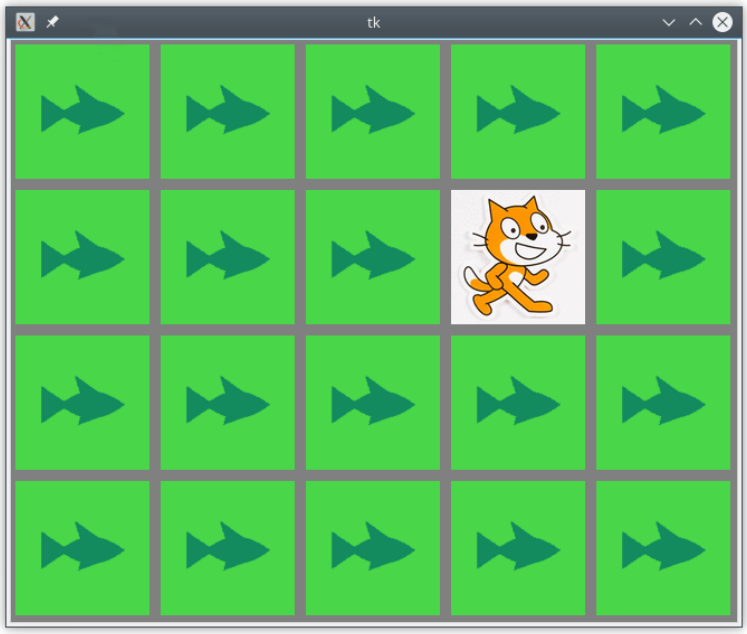
```
from tkinter import *
from random import shuffle, randrange

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=4
NB_COLS=5
NB_PICT=NB_LINES*NB_COLS//2
WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
XO=YO=SIDE//2

LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def make_board():
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
```

```
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray')
cnv.pack()

cover = PhotoImage(file="./images/cover.gif")
logos=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]

board=make_board()
print(board)

# Placement des images
for line in range(NB_LINES):
    for col in range(NB_COLS):
        center=(X0+col*SIDE, Y0+line*SIDE)
        nro_image=board[line][col]
        mon_image=logos[nro_image]
        cnv.create_image(center, image=mon_image)
        id_cover=cnv.create_image(center, image=cover)
        ids_cover[line][col] = id_cover

line=randrange(NB_LINES)
col=randrange(NB_COLS)
print(line, col)
im_id=ids_cover[line][col]
cnv.delete(im_id)

root.mainloop()
```

```
[[2, 7, 0, 4, 5], [5, 8, 1, 9, 7], [4, 6, 0, 8, 2], [9, 1, 6, 3, 3]]
1 3
```

Le principal ajout concerne l'incorporation des images des figures. Ces images sont situées dans un répertoire `images` à la racine du programme. Par exemple, l'image représentant le langage à Java est accessible par l'adresse `./images/java.gif`. Toutes les images sont au format `gif`, de forme carrée et de taille unique (120 pixels). Pour un logo, le nom de fichier est toujours de la forme `lang_programmation.gif`.

Les indices de ligne et de colonne choisis au hasard, qui sont affichés à la fin du programme et qui représentent les indices du logo visible dans la grille sont dans notre exemple (repris ci-dessous pour une meilleure lisibilité) `line=1` et `col=3` :



Si on regarde la sortie console (ligne 60 ci-dessus), dans `board`, à la position `(1, 3)`, on doit trouver une valeur (ici 9) qui est l'indice du langage représenté dans la liste des langages (lignes 15-16), ici le langage Scratch. Et on constate en effet, c'est bien ce logo qui est visible.

Chargement des images

Tout se passe à la ligne 36. On crée une liste (en compréhension) `logos` qui convertit avec la méthode `PhotoImage` chaque image gif en un format utilisable par Tkinter. Les résultats des conversions sont placées dans une liste de 10 objets.

IMPORTANT : chaque logo `gif` représentant un langage est donc identifié par un indice unique entre 0 et 9 inclus. Ces indices permettent de distribuer les images dans la grille à l'aide du tableau `board` d'entiers aléatoires.

Remplissage de la grille d'images

C'est peut-être la partie la moins facile à comprendre du code ci-dessus.

On commence par créer le tableau `board`, de taille 4 x 5 et formé de 20 entiers entre entre 0 et 9, chacun présent exactement deux fois. Chaque entier (disons `i`) présent dans `board` **représente** un logo de langage de programmation, plus précisément le logo d'indice `i` de la liste `LANG` (lignes 15-16).

C'est exactement cette disposition qui va être représentée sur le canevas. Pour cela, on parcourt la future grille par indice de ligne et de colonne et, en même temps, on accède à la même position dans `board`. Et on demande au canevas de dessiner d'abord le logo *puis* de le recouvrir avec l'image de couverture (bien sûr, il faut dessiner le logo avant la couverture). Pour dessiner le logo, il suffit de récupérer l'équivalent entier de ce logo dans `board` (ligne 46), puis d'accéder à la version `PhotoImage` correspondante (cf. ligne 47).

Récupération des id des images

Pour des besoins ultérieurs, on récupère dans une liste `ids_cover` (ligne 49) l'id de chacune des 20 images de couverture utilisées. Elles sont placées dans dans la liste `ids_cover` au fur et à mesure que l'on parcourt les 20 positions possibles.

On utilise cette liste pour retourner une carte au hasard. Pour cela

— on choisit au hasard une ligne et une colonne de la grille (lignes 52-53)

— on récupère, grâce au tableau `ids_cover`, l'id de l'image correspondante (ligne 55)

— on efface l'image en utilisant la méthode `delete` sur l'id trouvée.

Constantes magiques

Le code de ce tutoriel s'efforce de ne pas utiliser de « Constantes magiques ». Utiliser une telle variable consisterait à écrire par exemple :

```
NB_PICT = 10
```

au lieu de

```
NB_PICT = NB_LINES * NB_COLS // 2
```

L'intérêt d'utiliser des variables et non pas des constantes littérales, est

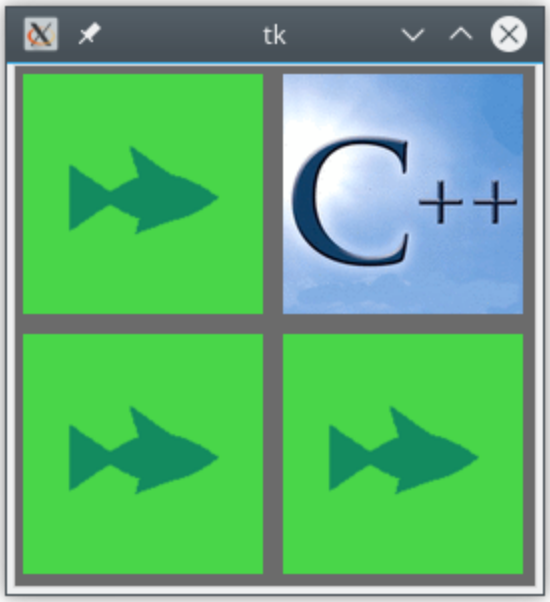
— d'offrir une meilleure lisibilité : au lieu d'être confronté à une constante anonyme et énigmatique, on a un nom explicite ;

— de permettre, de modifier sans effort le programme.

Ainsi, il est immédiat d'avoir un code qui marche pour un plateau 2 x 2 au lieu de 4 x 5, il suffit juste de changer deux constantes dans le code (lignes 8-10) et toutes les autres constantes s'adaptent :

```
NB_LINES=2
NB_COLS=2
NB_PICT=NB_LINES*NB_COLS//2
```

ce qui afficherait le plateau suivant :



Si vous devez devez tester votre jeu pour savoir s'il fonctionne jusqu'au bout sans bug, il est plus simple et rapide de le tester sur un plateau de 4 cartes qu'un plateau de 20 !

Anomalie de PhotoImage

Le lecteur attentif aura peut-être remarqué que la création de la liste `logos` ligne 36 n'est pas justifiée. En effet, la liste `logos` est utilisée ligne 47 et on pourrait obtenir le même résultat en créant une image `PhotoImage` à ce moment.

En fait si l'on procèdait ainsi, seule la dernière image serait visible. Cela est dû à un comportement particulier de PhotoImage.

Factorisation en fonctions

Le code ci-dessus est assez mal organisé et placé en vrac, non enveloppé dans des fonctions. On peut placer le code de remplissage du plateau par les images dans une fonction `fill` et la construction des widgets et le lancement du jeu dans une fonction `play` :

grille_toutes_images_fonctions.py

```
1  from tkinter import *
2  from random import shuffle, randrange
3
4  PICT_SIZE=120
5  PAD=10
6  SIDE=PICT_SIZE+PAD
7
8  NB_LINES=4
9  NB_COLS=5
10 NB_PICT=NB_LINES*NB_COLS//2
11 WIDTH=SIDE*NB_COLS
12 HEIGHT=SIDE*NB_LINES
```

```
X0=Y0=SIDE//2

LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def make_board():
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, board, cover, logos, ids_cover):
    # Placement des images
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            center=(X0+col*SIDE, Y0+line*SIDE)
            nro_image=board[line][col]
            mon_image=logos[nro_image]
            cnv.create_image(center, image=mon_image)
            id_cover=cnv.create_image(center, image=cover)
            ids_cover[line][col] = id_cover

def play():
    root=Tk()
    cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray')
    cnv.pack()

    cover = PhotoImage(file="./images/cover.gif")
    logos=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
    ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
    board=make_board()

    print(board)
    fill(cnv, board, cover, logos, ids_cover)

    line=randrange(NB_LINES)
    col=randrange(NB_COLS)
    print(line, col)
    im_id=ids_cover[line][col]
    cnv.delete(im_id)

```

```
    root.mainloop()

play()
```

2 Les événements, les animations

L'événement du clic de souris

La programmation d'interfaces graphiques est de la programmation événementielle : une boucle infinie (la mainloop) attend des **événements** (c'est comme ça qu'on dit) tel qu'un appui sur une touche du clavier, un clic de souris ; éventuellement, le programme réagit à ces événements.

Pour surveiller un clic de souris sur le canevas, il suffit

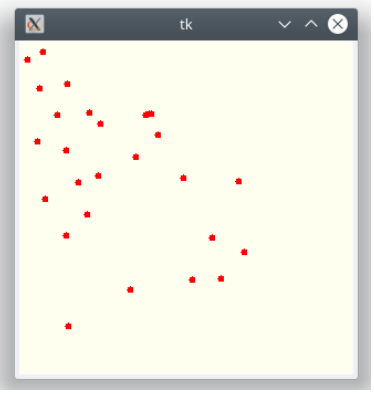
— d'écrire une fonction (disons `clic`) qui sera appelée (automatiquement) chaque fois qu'un clic aura lieu sur le canevas ;

— de **lier** l'événement clic de souris à l'appel de la fonction `clic`.

On dit que la fonction `clic` est une fonction de rappel (*callback function* en anglais). La *liaison* est appelée *binding* en anglais, cf. le code ci-dessous.

Le programme présenté ci-dessous :

```
100 225
44 258
18 43
7 17
21 10
43 39
42 99
23 143
42 176
61 157
71 122
73 75
63 65
34 67
16 91
53 128
105 105
198 127
174 178
156 216
182 215
203 191
148 124
125 85
114 67
117 66
119 66
```



montre la réaction à un événement de clic de souris sur un canevas affichant les coordonnées du clic et dessinant un disque là où on a cliqué. Le code correspondant est :

```
from tkinter import *

root = Tk()
cnv = Canvas(root, width=300, height=300, bg="ivory")
cnv.pack()

```

```
7  def clic(event):
8      x, y = event.x, event.y
9      print(x, y)
10     cnv.create_oval(x-3, y-3, x+3, y+3,
11                     fill='red', outline='')
12
13 cnv.bind("<Button-1>", clic)
14 root.mainloop()
```

```
15 151 244
16 121 225
17 101 155
18 92 110
19 86 89
20 61 63
```

— Ligne 9 : la fonction de rappel `clic` ; chaque fois qu'on clique sur le canevas, cette fonction est appelée et reçoit en argument, sans que le programmeur ait la main dessus, un objet appelé ici `event` qui représente (et donne accès) à toutes les propriétés du clic qui a été effectié.

— Ligne 12 : la **liaison** de l'événement clic gauche de souris, qui se code en Tkinter par `"<Button-1>"` et de la fonction `clic` : chaque fois qu'un événement clic de souris est intercepté sur le canevas, la fonction `clic` est appelée. -

— La fonction `clic` effectue ici deux actions :

  — elle affiche dans la console les coordonnées sur le canevas du pixel qui a été cliqué. Dans l'exemple (lignes 15-19), on voit que l'utilisateur a cliqué 6 fois sur le canevas. Remarquez que les valeurs lues sont, bien sûr, entre 0 et 300 ;

  — elle dessine quelque chose au point du clic (ici, un petit disque rouge, c'est juste pour que l'image soit compréhensible).

L'événement `"<Button>"` (sans numéro) s'applique à tout événement relatif à un bouton de souris, le clic gauche comme le clic droit ou l'action sur la molette. Si on veut capturer exactement un clic **gauche** de souris, il faut utiliser l'événement `"<Button-1>"`.

Identifier une carte après un clic

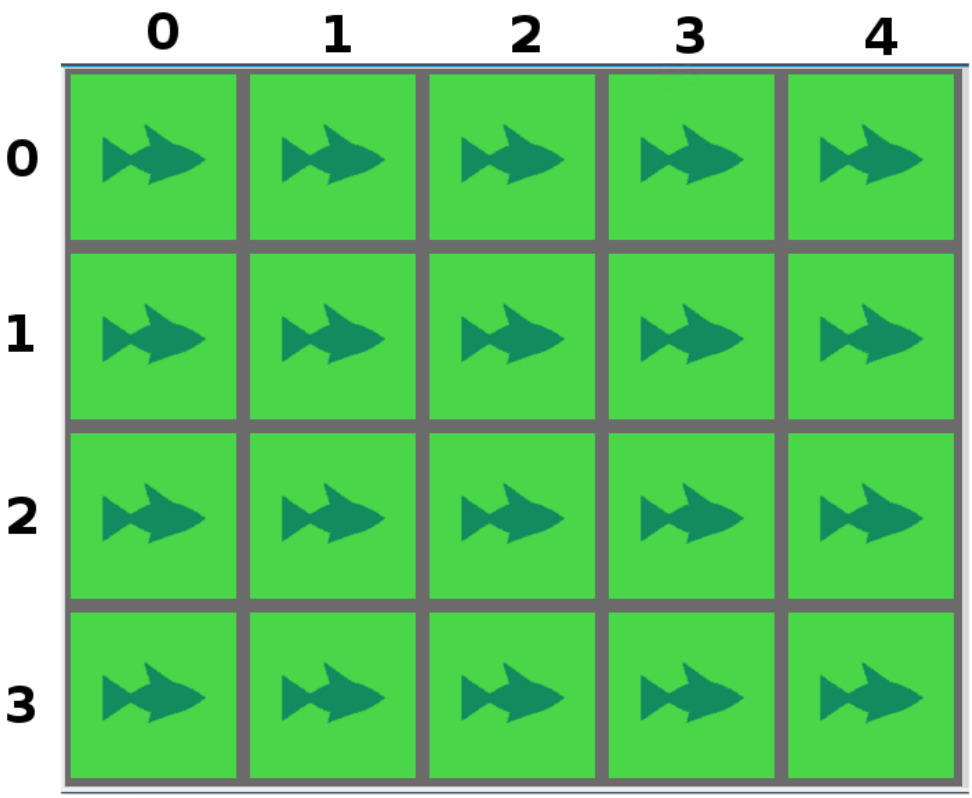
Dans notre jeu, l'utilisateur clique, dans le canevas, sur une carte. Comment le programme sait-il sur quelle carte il a cliqué ? Pour cela, rappelons-nous que

— nous pouvons récupérer les coordonnées d'un clic sur le canevas ;

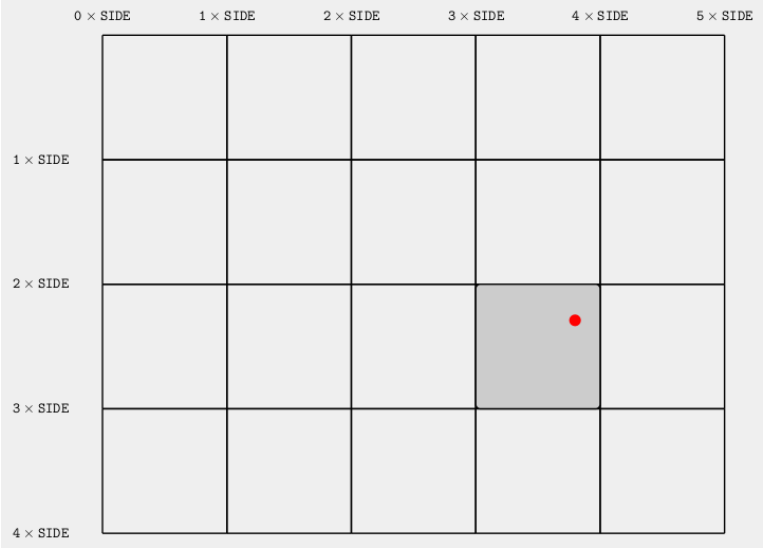
— nous connaissons la disposition de chaque carte sur le plateau.

Grâce à ces deux informations et un petit calcul, nous pouvons, récupérer les indices de ligne et de colonne dans la grille de la carte cliquée.

Le point essentiel est de voir notre grille d'images comme un tableau 2D :



Supposons que le clic ait eu lieu au point de coordonnées (x, y) du canevas, en rouge ci-dessous :



Comme les cartes sont régulièrement espacées d'une distance `SIDE`, en supposant que chaque ligne et chaque colonne de cartes soit numérotée à partir de 0, le numéro de

— la colonne `col` vaut le quotient entier de `x` par `SIDE` : `col` = `x//SIDE`,

— la ligne `lig` vaut le quotient entier de `y` par `SIDE` : `lin` = `y//SIDE`.

On fera attention à l'ordre :

— ligne ⟶ ordonnée

— colonne ⟶ abscisse

Faisons un essai de code :

`id_clic_image.py`

```
from tkinter import *

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=4
NB_COLS=5
WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
X0=Y0=SIDE//2

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray')
cnv.pack()

cover = PhotoImage(file="./images/cover.gif")

for line in range(NB_LINES):
```

```
20      for col in range(NB_COLS):
21          cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
22
23  def clic(event):
24      x=event.x
25      y=event.y
26      line=x//SIDE
27      col=y//SIDE
28      print(line, col)
29
30  cnv.bind("<Button>", clic)
31  root.mainloop()
```

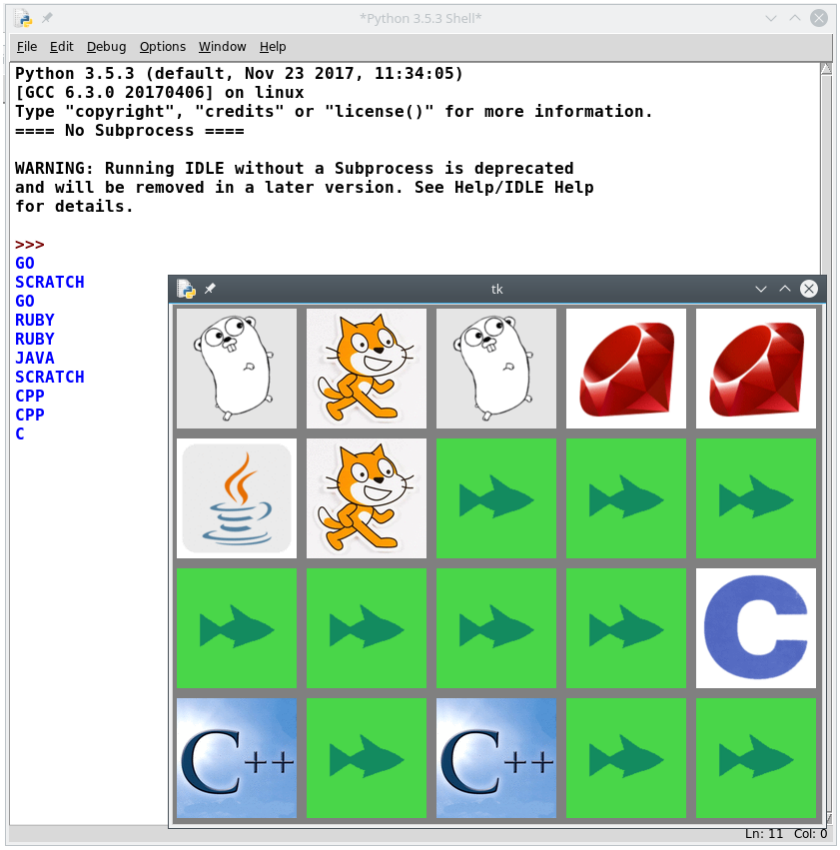
```
32  2 2
33  4 3
34  4 0
```

Dans l'affichage (lignes 32-34), on voit sur quelles cartes l'utilisateur a cliqué.

Pour être très précis, ce code fonctionne aussi si on clique légèrement en dehors de la carte puisqu'il y a un écart de 10 pixels (cf. lignes 4-5) entre la largeur de la carte et la largeur de son emplacement sur la grille du canevas. Ce genre d'approximation est courant.

Retourner une carte suite à un clic

Désormais, il va nous être possible de découvrir les cartes grâce à simple clic. Pour cela, on a besoin, bien sûr, d'un plateau avec toutes ses cartes cachées et de disposer des ids des couvertures d'image. Ensuite, il suffit de demander à la fonction de callback du clic de supprimer avec la méthode `delete` la couverture. Il est même possible d'écrire un message dans la console où on affiche le langage de programmation dont le logo est découvert (le programme ci-dessous étant exécuté sous IDLE, cf. la copie d'écran ci-dessous) :



Voici le code :

decouvrir_logo_clic.py

```
from tkinter import *
from random import shuffle, randrange

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=4
NB_COLS=5
NB_PICT=NB_LINES*NB_COLS//2
WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
XO=YO=SIDE//2

LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def make_board():
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
```

```
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, board, cover, logos, ids_cover):
    # Placement des images
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            center=(X0+col*SIDE, Y0+line*SIDE)
            nro_image=board[line][col]
            mon_image=logos[nro_image]
            cnv.create_image(center, image=mon_image)
            id_cover=cnv.create_image(center, image=cover)
            ids_cover[line][col] = id_cover

def clic(event):
    x=event.x
    y=event.y
    col=x//SIDE
    line=y//SIDE
    cnv.delete(ids_cover[line][col])
    i=board[line][col]
    print(LANG[i].upper())



root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray')
cnv.pack()
cnv.bind("<Button>", clic)

cover = PhotoImage(file="./images/cover.gif")
logos=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]

board=make_board()
fill(cnv, board, cover, logos, ids_cover)



root.mainloop()
```

qui comme l'indiquait la copie d'écran, montre ceci dans la console d'IDLE :

```
>>>
GO
SCRATCH
GO
RUBY
RUBY
JAVA
SCRATCH
CPP
CPP
C
```

— Ligne 56 : le clic gauche est associé à la fonction clic.
— Lignes 42-49 : la fonction clic :
  — lignes 43-46 : calcule la position `(line, col)` dans la grille de la carte cliquée,
  — ligne 47 : va regarder dans `ids_cover` quel est l'id de la couverture à cette position,
  — lignes 47 : supprime la couverture,
  — lignes 48 : détermine le représentant entier du logo découvert,
  — lignes 49 : affiche en majuscule dans la console le nom du langage du logo qui est découvert.

Une question qui vous intrigue peut-être : que se passe-t-il lorsqu'on re-clique sur une carte déjà découverte ? Réponse : à ce moment-là, la ligne 47 du code va toujours être exécutée : le canevas va essayer de supprimer un objet mais dont l'id n'existe plus. Il se trouve que le canevas est conçu pour ne pas réagir dans ce cas, ce qui ne cause donc aucun plantage du programme.

Illustrer `after` en créant des images

La méthode `after` permet de déclencher avec un certain délai de temps une action définie par une fonction.
Soit par exemple le code ci-dessous :

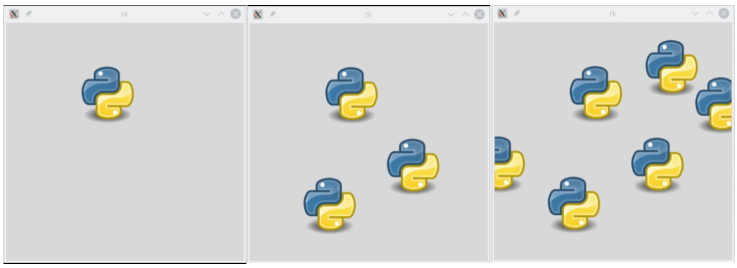
after_images.py

```python
from tkinter import *
from random import randrange

SIDE=400
root = Tk()
cnv = Canvas(root, width=SIDE, height=SIDE)
cnv.pack()

logo = PhotoImage(file="python.gif")

def action(x, y):
    cnv.create_image((x, y), image=logo)

x, y= randrange(SIDE),randrange(SIDE)
cnv.after(1000, action, x, y)
```

```
16
17 x, y= randrange(SIDE),randrange(SIDE)
18 cnv.after(2000, action, x, y)
19
20 x, y= randrange(SIDE),randrange(SIDE)
21 cnv.after(3000, action, x, y)
22
23 x, y= randrange(SIDE),randrange(SIDE)
24 cnv.after(4000, action, x, y)
25
26 x, y= randrange(SIDE),randrange(SIDE)
27 cnv.after(5000, action, x, y)
28
29 x, y= randrange(SIDE),randrange(SIDE)
30 cnv.after(6000, action, x, y)
31
32 root.mainloop()
```

On a défini une fonction `action` qui prend deux paramètres (le nom `action` n'a rien d'obligatoire).

Voici l'exécution :

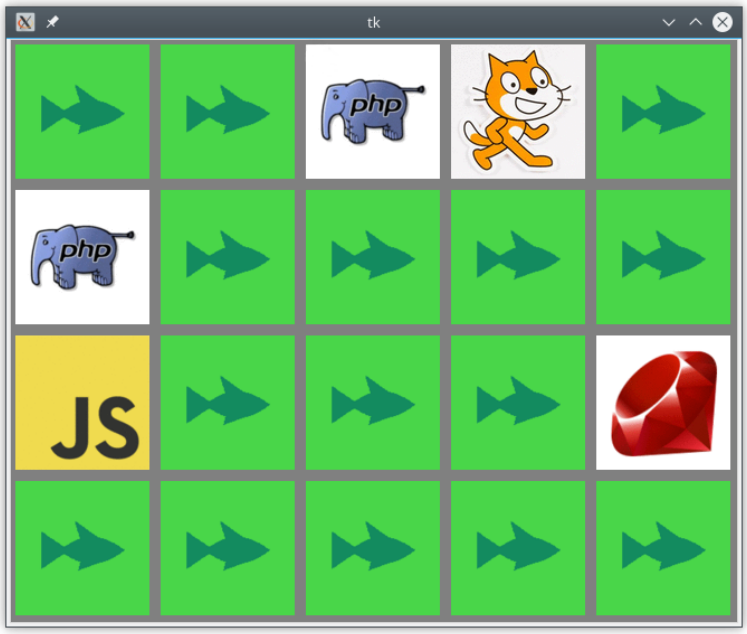


Lorsque le programme est lancé, les 6 instructions `cnv.after(ms, action, x, y)` vont être exécutées les unes après les autres et instantanément. Chacune de ces instructions va déclencher l'exécution de la fonction `action` mais après le délai en ms indiqué par l'appel `cnv.after(ms, action, x, y)`. Par exemple, une instruction telle que `cnv.after(4000, action, 200, 300)` va provoquer, 4 secondes (4000 ms) après avoir été activée, l'exécution de l'appel `action(200, 300)`. D'où l'effet d'animation.

La méthode `after` n'est pas spécifique de Canvas : dans le code ci-dessus, on pourrait remplacer les appels `cnv.after` par des appels `root.after`.

Retourner une carte avec la méthode `after`

Le memory game doit être animé puisque lorsque le joueur retourne deux cartes qui sont différentes, les cartes doivent, au bout d'un certain délai, revenir faces cachées.



Pour se familiariser avec la méthode `after`, on va écrire un code qui, à partir d'un plateau de cartes faces cachées, retourne aléatoirement certaines cartes, à raison d'une chaque seconde (c'est à cause de cette question de délai qu'on utilise `after`) :

after_decouvre_alea.py

```
from tkinter import *
from random import shuffle, randrange

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=4
NB_COLS=5
NB_PICT=NB_LINES*NB_COLS//2
WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
XO=YO=SIDE//2

LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def make_board():
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
```

```
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, board, cover, logos, ids_cover):
    # Placement des images
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            center=(X0+col*SIDE, Y0+line*SIDE)
            nro_image=board[line][col]
            mon_image=logos[nro_image]
            cnv.create_image(center, image=mon_image)
            id_cover=cnv.create_image(center, image=cover)
            ids_cover[line][col] = id_cover

def clic(event):
    x=event.x
    y=event.y
    col=x//SIDE
    line=y//SIDE
    cnv.delete(ids_cover[line][col])
    i=board[line][col]
    print(LANG[i].upper())

def swap(ids_cover, cnv):
    line=randrange(NB_LINES)
    col=randrange(NB_COLS)
    im_id=ids_cover[line][col]
    cnv.delete(im_id)

def play():
    root=Tk()
    cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray')
    cnv.pack()

    cover = PhotoImage(file="./images/cover.gif")
    logos=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
    ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]

    board=make_board()
    fill(cnv, board, cover, logos, ids_cover)

    cnv.after(1000, swap, ids_cover, cnv)
    cnv.after(2000, swap, ids_cover, cnv)
    cnv.after(3000, swap, ids_cover, cnv)
    cnv.after(4000, swap, ids_cover, cnv)
```

```
73      cnv.after(5000, swap, ids_cover, cnv)
74
75      root.mainloop()
76
77  play()
```

On a écrit une fonction `swap` (ligne 51) qui retourne aléatoirement une carte (rappel : retourner c'est effacer la couverture). Quand on exécute le programme, on voit 5 cartes (ou peut-être moins dans certains cas) être automatiquement retournées.

Lorsque le programme est exécuté, les cinq appels (lignes 69-73) à la fonction `after` sont effectués **quasiment simultanément**. Mais ces appels vont déclencher des appels de la fonction `swap` après l'intervalle de temps défini par le premier argument de l'appel de `after`. Par exemple, quand l'appel `cnv.after(4000, swap)` a lieu (ligne 72), c'est à dire quasiment immédiatement après le lancement du programme, il faut attendre 4 s (4000 ms) avant que la fonction `swap` ne soit appelée. Une seconde plus tard, la fonction `swap` est à nouveau appelée suite à l'appel `cnv.after(5000, swap)`, cf. ligne 73.

Animer le clic pour retourner une carte

Le retournement de carte :
s'effectue ainsi :

— un clic provoque le démasquage de la carte
— une seconde plus tard, la carte revient en position cachée.

Le code ci-dessous implémente le retournement :

clic_after.py

```
1  from tkinter import *
2  from random import shuffle
3
4  COTE = 120
5  PAD = 5
6  SIDE = COTE + PAD
7
8  NB_LIG = 4
9  NB_COL = 5
10
11 LARG = SIDE * NB_COL
12 HAUT = SIDE * NB_LIG
13 X0 = Y0 = SIDE // 2
14
15 NB_CARTES=NB_LIG*NB_COL//2
16
17 LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
18       'php', 'python', 'ruby', 'scratch']
19
20 def melanger_grille():
21     cartes=list(range(NB_CARTES))*2
```

```
22      shuffle(cartes)
23
24      P=[]
25      k=0
26      for lig in range(NB_LIG):
27          L=[]
28          for col in range(NB_COL):
29              L.append(cartes[k])
30              k+=1
31          P.append(L)
32      return P
33
34
35  fen = Tk()
36  cnv = Canvas(fen, width=LARG, height=HAUT, bg='gray')
37  cnv.pack()
38
39  plateau=melanger_grille()
40
41  # Liste logos
42  logos=[]
43
44  for i in range(NB_CARTES):
45      lang=LANG[i]
46      nom="./images/"+lang+".gif"
47      logo=PhotoImage(file=nom)
48      logos.append(logo)
49
50
51  ids_cover=[]
52  cover=PhotoImage(file="./images/cover.gif")
53
54  # Placement des images
55  for lig in range(NB_LIG):
56      L=[]
57      for col in range(NB_COL):
58          centre = (X0 + col * SIDE, Y0 + lig * SIDE)
59          i=plateau[lig][col]
60          logo=logos[i]
61          cnv.create_image(centre, image=logo)
62          id_cover=cnv.create_image(centre, image=cover)
63          L.append(id_cover)
64      ids_cover.append(L)
65
66  def clic(event):
67      X=event.x
68      Y= event.y
69      col=X//SIDE
```

```
70      lig=Y//SIDE
71      id_cover=ids_cover[lig][col]
72      cnv.delete(id_cover)
73      cnv.after(1000, recouvrir, (lig, col))
74
75  def recouvrir(pos):
76      lig, col=pos
77      c = (X0 + col * SIDE, Y0 + lig * SIDE)
78      id_cover=cnv.create_image(c, image=cover)
79      ids_cover[lig][col]=id_cover
80
81  cnv.bind("<Button>", clic)
82
83  fen.mainloop()
```

— Lignes 71-72 : quand l'utilisateur clique sur une carte cachée, la couverture est retirée, ce qui démasque la carte

— Ligne 73 : la fonction `recouvrir` (lignes 75-79) est appelée une seconde plus tard pour cacher la carte. La fonction recouvrir est appelée avec la position ligne, colonne concernée.

— Ligne 76-79 : la fonction recouvrir reçoit les indices de ligne et colonne où elle doit recouvrir la carte visible. Une couverture est créée (ligne 78) et son id est enregistrée dans la liste des id, afin de pouvoir être à nouveau retirée si l'utilisateur clique à nouveau sur la carte.

Attention aux bugs !

Le programme ci-dessus fonctionne très bien ... si on l'utilise posément. Mais si, **immédiatement** après avoir démasqué une carte, on reclique dessus, et, une fois que la carte sera remasquée, si on clique encore dessus, la couverture va rester présente et la carte ne sera pas démasquée. En effet, le premier clic va provoquer le placement d'une carte de couverture (disons cover1) au bout d'une seconde et l'id de cover1 sera enregistrée dans la liste `ids_cover` des ids de couverture. Mais, le 2e clic va provoquer (cf. ligne 73) un 2e appel de la fonction `retourner` qui va en fait rajouter une 2e carte couverture (disons cover2) et qui sera enregistrée dans la liste d'id (et écrasera celle de cover1). Lorsque la carte est cachée par cover2, si l'utilisateur re-clique dessus, il va bien retirer la carte de couverture cover2 mais il restera encore la carte de couverture cover1 ; celle-ci ne pourra être retirée avec un nouveau clic car, l'id de cover1 est perdue, elle a été écrasée par celle de cover2.

Si on veut que le programme réagisse correctement, il faut mettre un drapeau qui tienne compte d'un premier effacement de la carte de couverture. Dans le code ci-dessous, le drapeau est activé en plaçant `None`, à la bonne position ligne et colonne dans la liste 2D des id de couverture :

clic_after_no_bug.py

```
1  from tkinter import *
2  from random import shuffle
3
4  COTE = 120
5  PAD = 5
6  SIDE = COTE + PAD
7
8  NB_LIG = 4
```

```
NB_COL = 5

LARG = SIDE * NB_COL
HAUT = SIDE * NB_LIG
X0 = Y0 = SIDE // 2

NB_CARTES=NB_LIG*NB_COL//2

LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def melanger_grille():
    cartes=list(range(NB_CARTES))*2
    shuffle(cartes)

    P=[]
    k=0
    for lig in range(NB_LIG):
        L=[]
        for col in range(NB_COL):
            L.append(cartes[k])
            k+=1
        P.append(L)
    return P


fen = Tk()
cnv = Canvas(fen, width=LARG, height=HAUT, bg='gray')
cnv.pack()

plateau=melanger_grille()

# Liste logos
logos=[]

for i in range(NB_CARTES):
    lang=LANG[i]
    nom="./images/"+lang+".gif"
    logo=PhotoImage(file=nom)
    logos.append(logo)


ids_cover=[]
cover=PhotoImage(file="./images/cover.gif")

# Placement des images
for lig in range(NB_LIG):
    L=[]
```

```
57      for col in range(NB_COL):
58          centre = (X0 + col * SIDE, Y0 + lig * SIDE)
59          i=plateau[lig][col]
60          logo=logos[i]
61          cnv.create_image(centre, image=logo)
62          id_cover=cnv.create_image(centre, image=cover)
63          L.append(id_cover)
64      ids_cover.append(L)
65
66  def clic(event):
67      X=event.x
68      Y= event.y
69      col=X//SIDE
70      lig=Y//SIDE
71      id_cover=ids_cover[lig][col]
72      if id_cover is not None:
73          cnv.delete(id_cover)
74          ids_cover[lig][col]=None
75          cnv.after(2000, recouvrir, (lig, col))
76
77  def recouvrir(pos):
78      lig, col=pos
79      c = (X0 + col * SIDE, Y0 + lig * SIDE)
80      id_cover=cnv.create_image(c, image=cover)
81      ids_cover[lig][col]=id_cover
82
83  cnv.bind("<Button>", clic)
84
85
86
87  fen.mainloop()
```

— Lignes 77-81 : la fonction `retourner` est inchangée.

— Lignes 66-75 : on ne retourne la carte que si elle n'est pas en situation d'attente de retournement (cf. ligne 72). Une carte en attente de retournement se reconnaît parce que son id de couverture est placé à `None` dans la liste des des ids de couverture (cf. ligne 74)

— Si on re-clique moins de 1 seconde après un premier clic, le 2e clic est ignoré (à cause de la ligne 72).

Codage d'un plateau jouable

Pour obtenir un plateau jouable, il reste à coder la gestion des paires cliquées :
ce qui est la partie la moins abordable du programme à coder. Pour cela, on va :

— compléter la fonction `clic` ;

— créer une fonction `handle_move` qui va gérer la paire cliquée ;

— créer une fonction `hide` qui va se charger de l'animation du recouvrement des cartes lorsqu'elles sont différentes.

On va partir d'un code écrit avec quelques fonctions et qui prépare le plateau avec les cartes couvertes et qui est capable de distinguer la ligne et la colonne où l'utilisateur clique :

final0.py

```
from tkinter import *
from random import shuffle

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=4
NB_COLS=5
WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
X0=Y0=SIDE//2
NB_PICT=NB_LINES*NB_COLS//2
LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def make_board():
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, images, board, cover, ids_cover):
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            center=(X0+col*SIDE, Y0+line*SIDE)
            k=board[line][col]
            cnv.create_image(center, image=images[k])
            ids_cover[line][col]=cnv.create_image(center, image=cover)

def lin_col(x, y):
    return (y//SIDE, x//SIDE)

def clic(event):
    lin, col = lin_col(event.x, event.y)
    print(lin, col)

root=Tk()
```

```
46 cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
47 cnv.pack()
48 cnv.bind("<Button>", clic)
49
50 cover = PhotoImage(file="./images/cover.gif")
51 images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
52 ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
53
54 board=make_board()
55 fill(cnv, images, board, cover, ids_cover)
56 root.mainloop()
```

Préparation

On va définir une variable `move` qui enregistrera l'apparition des paires de clics sur des cartes à découvrir. La variable `move` sera une liste de deux éléments en sorte que :

— `move[0]` représente le premier clic sur une carte cachée,

— `move[1]` représente le 2e clic valide.

Quand il n'y a aucun clic à gérer sur des cartes à retourner, `move` vaut `[None, None]`. Sinon, `move[0]` vaut le couple `(line, col)` de positions où le premier clic a eu lieu. De même, pour `move[1]` mais pour le 2e clic.

Pour surveiller l'état du plateau de jeu, chaque fois que deux cartes identiques auront été découvertes par le joueur, le tableau `board` sera mis-à-jour en remplaçant la valeur commune des cartes par -1 aux deux positions.

Je rappelle qu'il a été convenu au début du tutoriel, que dans un but de simplification du codage, le temps que le retournement de faces visibles se fasse, les clics étaient bloqués ; on pourrait envisager une règle différente.

Voici le code de la fonction `clic` :

```
1 def clic(event):
2     if move[1] is not None:
3         return
4     col, line=event.x//SIDE, event.y//SIDE
5     if board[line][col]!=-1:
6         handle_move(line, col)
```

— Ligne 1 : si `move[1]` n'est pas à `None`, c'est qu'une paire est en cours de traitement (il y a déjà eu deux clics) ; il faut donc ignorer le clic qui vient d'être détecté puisque le jeu est en attente du retournement des deux cartes différentes.

— Ligne 4 : la fonction récupère la position ligne/colonne du clic.

— Ligne 5 : si `board[line][col]` vaut `-1`, cela signifie que la carte est définitivement retournée, donc il n'y a aucun traitement à opérer. Donc cette situation est ignorée par la fonction `clic`.

— Ligne 6 : dans le cas contraire, la fonction `handle_move` est appelée.

— Noter qu'il est possible que le 2e clic ait été effectué sur la 1re carte retournée. Il faudra tenir compte de ce cas dans un code ultérieur.

La fonction `handle_move`

C'est la fonction la plus délicate à écrire. Elle doit gérer les paires de clics et lancer l'animation si le joueur découvre des cartes différentes. Voici le code

```
1  def handle_move(line, col):
2      item=ids_cover[line][col]
3      cnv.delete(item)
4      if move[0] is None :
5          move[0]=(line, col)
6      else:
7          if move[0]==(line, col):
8              return
9          move[1]=(line, col)
10         i, j=move[0]
11         if board[i][j]==board[line][col]!=-1:
12             board[i][j]=board[line][col]=-1
13             move[0]=move[1]=None
14         else:
15             cnv.after(400, hide, i, j, line, col)
```

— Ligne 2 : on récupère l'id de la couverture qui est (ou était) à la position `(lin, col)`. Pourquoi dis-je « était » ? Parce qu'il se peut que la couverture ait disparu au premier clic et que le joueur ait re-cliqué sur la même carte, cas qui sera définitivement réglé aux lignes 7-8.

— ligne 3 : si en position `(line, col)` la carte n'était pas retournée, qu'il s'agisse de la 1re carte ou de la 2e carte, elle doit être retournée. Sinon, l'action de suppression n'aura aucun effet (rappel : sans conséquence, on peut demander au canevas de supprimer une id qui n'existe pas ou n'existe plus).

— Ligne 4-5 : cela signifie que c'est le premier clic de la paire de clics : `move[0]` enregistre la position de la carte retournée.

— Lignes 7-15 : c'est la situation où c'est la 2e carte qui est retournée.

— Ligne 7-8 : c'est la situation où le joueur re-clique sur la 1re carte, qui a déjà été retournée. Cette action doit être ignorée car le jeu compare des cartes situées à des positions distinctes.

— Lignes 9-15 : gestion du 2e clic de la paire de clics.

— Ligne 9 : `move[1]` (qui correspond au 2e clic) enregistre la position de la 2e carte.

— Ligne 10 : `(i, j)` est la position de la 1re carte retournée.

— Lignes 11-13 : c'est le cas où les deux cartes cliquées sont identiques :

  — il faut les marquer en `-1` dans `board` (ligne 12) ;

  — il faut réinitialiser `move` (ligne 13).

— Ligne 14-15 : les cartes cliquées sont différentes (ligne 14), après une attente de 4 dixièmes de seconde, la fonction `hide` va recouvrir les deux cartes (ligne 15).

La fonction `hide`

Elle est chargée de mettre face cachée les deux cartes retournées :

```
1 def hide(i, j, line,col):
2     ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
3     ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
4     move[0]=move[1]=None
```

— Ligne 2 : La 1e carte (première à cause de `i` et de `j`) est recouverte et l'id de la couverture est enregistrée dans le tableau des id de couvertures (on en a besoin pour quand le joueur re-cliquera sur cette face)

— Ligne 3 : De même pour la 2e couverture (deuxième à cause de `line` et de `col`)

— Ligne 4 : L'opération de gestion de la paire est terminée, il faut réinitialiser `move`.

D'où le code complet suivant :

final1.py

```
1  from tkinter import *
2  from random import shuffle
3
4  PICT_SIZE=120
5  PAD=10
6  SIDE=PICT_SIZE+PAD
7
8  NB_LINES=4
9  NB_COLS=5
10 WIDTH=SIDE*NB_COLS
11 HEIGHT=SIDE*NB_LINES
12 X0=Y0=SIDE//2
13 NB_PICT=NB_LINES*NB_COLS//2
14 LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
15       'php', 'python', 'ruby', 'scratch']
16
17 def make_board():
18     L=[v % NB_PICT for v in range(2*NB_PICT)]
19     shuffle(L)
20     board=[]
21     k=0
22     for line in range(NB_LINES):
23         row=[]
24         for col in range(NB_COLS):
25             row.append(L[k])
26             k+=1
27         board.append(row)
28     return board
29
30 def fill(cnv, images, board, cover, ids_cover):
31     for line in range(NB_LINES):
32         for col in range(NB_COLS):
33             center=(X0+col*SIDE, Y0+line*SIDE)
34             k=board[line][col]
35             cnv.create_image(center, image=images[k])
```

```
            ids_cover[line][col]=cnv.create_image(center, image=cover)

def lin_col(x, y):
    return (y//SIDE, x//SIDE)

def hide(i, j, line,col):
    ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
    ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
    move[0]=move[1]=None

def handle_move(line, col):
    item=ids_cover[line][col]
    cnv.delete(item)
    if move[0] is None :
        move[0]=(line, col)
    else:
        if move[0]==(line, col):
            return
        move[1]=(line, col)
        i, j=move[0]
        if board[i][j]==board[line][col]!=-1:
            board[i][j]=board[line][col]=-1
            move[0]=move[1]=None
        else:
            cnv.after(400, hide, i, j, line, col)

def clic(event):
    if move[1] is not None:
        return
    col, line=event.x//SIDE, event.y//SIDE
    if board[line][col]!=-1:
        handle_move(line, col)

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
cnv.pack()
cnv.bind("<Button>", clic)

cover = PhotoImage(file="./images/cover.gif")
images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]

board=make_board()
fill(cnv, images, board, cover, ids_cover)

move=[None, None]
root.mainloop()
```

Factorisation complète en fonctions

La dernière partie du code précédent n'est pas enveloppée dans une fonction (lignes 69-82). Faisons une tentative de placer cette portion de code dans une fonction `play` :

```
1  # Code des autres fonctions omis
2
3  def play():
4      root=Tk()
5      cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
6      cnv.pack()
7      cnv.bind("<Button-1>", clic)
8
9      logo = PhotoImage(file="./images/cover.gif")
10     images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
11     items_back=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
12
13     board=make_board()
14     fill(cnv, images, board, logo, items_back)
15
16     root.mainloop()
17
18 play()
```

Le code ne fonctionnera pas : en effet les fonctions `clic`, `handle_move` et `cover` dépendent d'objets (comme `board`, ligne 13) qui désormais sont définis dans la fonction `play` et sont devenus inaccessibles. Par ailleurs, on ne peut pas envisager de rendre accessibles ces objets par retour de la fonction `play` : cette dernière ne peut rien renvoyer puisqu'elle contient la mainloop du jeu (ligne 16). Enfin, on ne peut pas transmettre ces objets à `clic` car, il n'y a aucun moyen de lui transmettre des arguments, elle n'accepte que `event` comme paramètre (ce comportement est imposé par Tkinter).

Une première solution est de déclarer dans la fonction `play`, comme variables `global`, toutes les variables qui manquent aux autres fonctions (cf. ligne 67 dans le code ci-dessous). Il n'est pas nécessaire de les déclarer à l'extérieur de la fonction `play`. Ce qui donne le code suivant :

`final2.py`

```
1  from tkinter import *
2  from random import shuffle
3
4  PICT_SIZE=120
5  PAD=10
6  SIDE=PICT_SIZE+PAD
7
8  NB_LINES=4
9  NB_COLS=5
10 WIDTH=SIDE*NB_COLS
11 HEIGHT=SIDE*NB_LINES
12 X0=Y0=SIDE//2
13 NB_PICT=NB_LINES*NB_COLS//2
```

```
LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def make_board():
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, images, board, cover, ids_cover):
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            center=(X0+col*SIDE, Y0+line*SIDE)
            k=board[line][col]
            cnv.create_image(center, image=images[k])
            ids_cover[line][col]=cnv.create_image(center, image=cover)

def hide(i, j, line,col):
    ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
    ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
    move[0]=move[1]=None

def handle_move(line, col):
    item=ids_cover[line][col]
    cnv.delete(item)
    if move[0] is None :
        move[0]=(line, col)
    else:
        if move[0]==(line, col):
            return
        move[1]=(line, col)
        i, j=move[0]
        if board[i][j]==board[line][col]!=-1:
            board[i][j]=board[line][col]=-1
            move[0]=move[1]=None
        else:
            cnv.after(400, hide, i, j, line, col)

def clic(event):
    if move[1] is not None:
        return
```

```
62      col, line=event.x//SIDE, event.y//SIDE
63      if board[line][col]!=-1:
64          handle_move(line, col)
65
66  def play():
67      global cnv, board, ids_cover, cover, move
68      root=Tk()
69      cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
70      cnv.pack()
71      cnv.bind("<Button>", clic)
72
73      cover = PhotoImage(file="./images/cover.gif")
74      images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
75      ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
76
77      board=make_board()
78      fill(cnv, images, board, cover, ids_cover)
79
80      move=[None, None]
81      root.mainloop()
82
83  play()
```

La 2e solution consiste à définir les fonctions qui ont besoin des variables qu'on a placées en `global` dans le dernier code, dans le corps même de la fonction `play` (lignes 39-66 dans le code ci-dessous). Il en effet possible en Python de définir une fonction g dans une fonction `f` et la fonction g lors de son exécution, peut lire des données dans le corps de la fonction `f`. D'où le code suivant :

final3.py

```
1  from tkinter import *
2  from random import shuffle
3
4  PICT_SIZE=120
5  PAD=10
6  SIDE=PICT_SIZE+PAD
7
8  NB_LINES=4
9  NB_COLS=5
10 WIDTH=SIDE*NB_COLS
11 HEIGHT=SIDE*NB_LINES
12 X0=Y0=SIDE//2
13 NB_PICT=NB_LINES*NB_COLS//2
14 LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
15       'php', 'python', 'ruby', 'scratch']
16
17 def make_board():
18     L=[v % NB_PICT for v in range(2*NB_PICT)]
19     shuffle(L)
```

```
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, images, board, cover, ids_cover):
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            center=(X0+col*SIDE, Y0+line*SIDE)
            k=board[line][col]
            cnv.create_image(center, image=images[k])
            ids_cover[line][col]=cnv.create_image(center, image=cover)

def play():
    def hide(i, j, line,col):
        ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
        ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover
        move[0]=move[1]=None

    def handle_move(line, col):
        item=ids_cover[line][col]
        cnv.delete(item)
        if move[0] is None :
            move[0]=(line, col)
        else:
            if move[0]==(line, col):
                return
            move[1]=(line, col)
            i, j=move[0]
            if board[i][j]==board[line][col]!=-1:
                board[i][j]=board[line][col]=-1
                move[0]=move[1]=None
            else:
                cnv.after(400, hide, i, j, line, col)

    def clic(event):
        if move[1] is not None:
            return
        col, line=event.x//SIDE, event.y//SIDE
        if board[line][col]!=-1:
            handle_move(line, col)

    root=Tk()
```

```
    cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
    cnv.pack()
    cnv.bind("<Button>", clic)

    cover = PhotoImage(file="./images/cover.gif")
    images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
    ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]

    board=make_board()
    fill(cnv, images, board, cover, ids_cover)

    move=[None, None]
    root.mainloop()

play()
```

3 Compléter et améliorer le jeu

Un bouton pour montrer une image

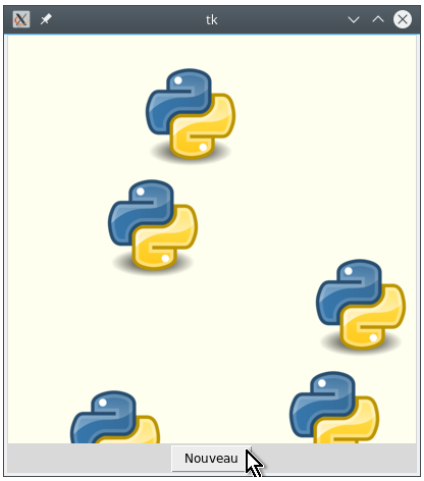
Un bouton, de même que le canevas est un widget. Le code ci-dessous place un bouton à côté d'un canevas :

bouton_image.py

```
from tkinter import *

SIDE=400
root = Tk()
cnv = Canvas(root, width=SIDE, height=SIDE, bg='ivory')
cnv.pack()

btn=Button(root, text="Nouveau")
btn.pack()

root.mainloop()
```

— Ligne 8 : un bouton est construit avec le constructeur `Button` (c'est une classe).
— Ligne 9 : comme pour tout widget, il faut l'inclure dans son environnement avec une méthode particulière, ici la méthode `pack`.
— Ligne 8 : le texte passé dans l'option `text` est affiché sur le bouton.
— Si on clique sur le bouton, rien ne se passe de visible. Pour lier une action à un bouton, il faudrait lui passer une option `command`.

Donnons une possibilité d'action au bouton : chaque fois qu'on clique le bouton, un logo 80x80 est dessiné sur le canevas :



Voici le code :

bouton_image1.py

```
1 from tkinter import *
2 from random import randrange
3
4 SIDE=400
5 root = Tk()
6 cnv = Canvas(root, width=SIDE, height=SIDE, bg='ivory')
```

```
7  cnv.pack()
8
9  logo = PhotoImage(file="python80.png")
10
11 def show():
12     center= (randrange(SIDE),randrange(SIDE))
13     cnv.create_image(center, image=logo)
14
15 btn=Button(root, text="Nouveau", command=show)
16 btn.pack()
17
18 root.mainloop()
```

— Ligne 15 : une option `command` a été donnée au constructeur `Button` : `command` référence une fonction sans paramètre, ici la fonction `show`, qui est exécutée à chaque pression sur le bouton.

— Lignes 11-13 : la fonction `show` ne peut prendre aucun paramètre ; elle dessine un logo Python aléatoire sur le canevas.

Un bouton pour rejouer

Revenons au memory game. On va ajouter un bouton pour réinitialiser le jeu (soit en cours de partie, soit en fin de partie).

On va repartir d'un code précédent :

`final1.py`

```
1  from tkinter import *
2  from random import shuffle
3
4  PICT_SIZE=120
5  PAD=10
6  SIDE=PICT_SIZE+PAD
7
8  NB_LINES=4
9  NB_COLS=5
10 WIDTH=SIDE*NB_COLS
11 HEIGHT=SIDE*NB_LINES
12 X0=Y0=SIDE//2
13 NB_PICT=NB_LINES*NB_COLS//2
14 LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
15       'php', 'python', 'ruby', 'scratch']
16
17 def make_board():
18     L=[v % NB_PICT for v in range(2*NB_PICT)]
19     shuffle(L)
20     board=[]
21     k=0
22     for line in range(NB_LINES):
```

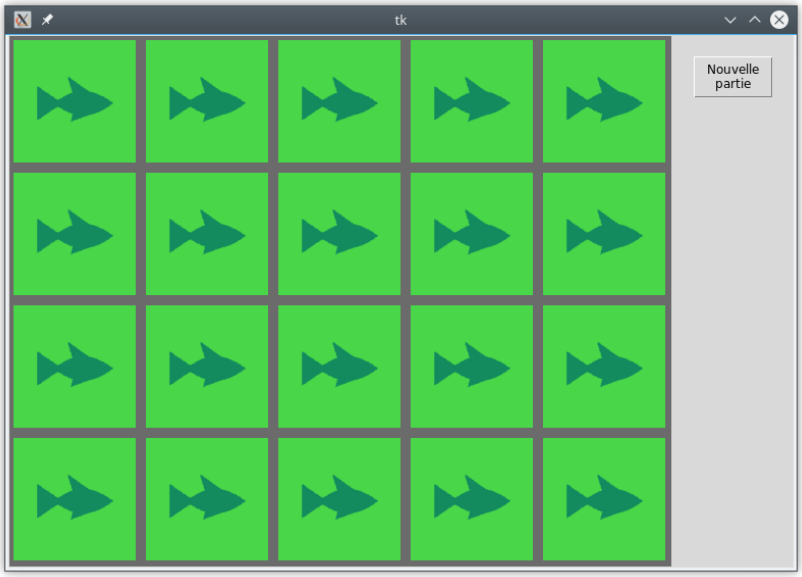
```
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, images, board, cover, ids_cover):
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            center=(X0+col*SIDE, Y0+line*SIDE)
            k=board[line][col]
            cnv.create_image(center, image=images[k])
            ids_cover[line][col]=cnv.create_image(center, image=cover)

def lin_col(x, y):
    return (y//SIDE, x//SIDE)

def hide(i, j, line,col):
    ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
    ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
    move[0]=move[1]=None

def handle_move(line, col):
    item=ids_cover[line][col]
    cnv.delete(item)
    if move[0] is None :
        move[0]=(line, col)
    else:
        if move[0]==(line, col):
            return
        move[1]=(line, col)
        i, j=move[0]
        if board[i][j]==board[line][col]!=-1:
            board[i][j]=board[line][col]=-1
            move[0]=move[1]=None
        else:
            cnv.after(400, hide, i, j, line, col)

def clic(event):
    if move[1] is not None:
        return
    col, line=event.x//SIDE, event.y//SIDE
    if board[line][col]!=-1:
        handle_move(line, col)

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
```

```
71 cnv.pack()
72 cnv.bind("<Button>", clic)
73
74 cover = PhotoImage(file="./images/cover.gif")
75 images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
76 ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
77
78 board=make_board()
79 fill(cnv, images, board, cover, ids_cover)
80
81 move=[None, None]
82 root.mainloop()
```

Le plateau est fonctionnel mais ce code, volontairement, n'est pas complètement encapsulé dans des fonctions (cf. lignes 69-82).

Placement du bouton

Pour des raisons esthétiques, je vais placer le bouton à droite :



La méthode de placement `pack` nécessite des options particulières pour réaliser cet agencement. La solution la plus adaptée serait d'utiliser un placement des widgets avec la méthode `grid` au lieu de la méthode `pack`. Mais comme il s'agit juste de placer un bouton une seule fois, je vais continuer avec la méthode `pack` et sans justifier le choix des options. Pour placer le bouton à droite, il va falloir passer à `pack` l'option `side=LEFT` pour le canevas (ligne 4 ci-dessous).

Par ailleurs, pour que le bouton ne soit pas collé à ce qui l'entoure, je vais passer (ligne 8 ci-dessous) à la méthode `pack` des options d'espacement horizontal (`padx`) et vertical (`pady`).

Le code de construction des widgets sera donc le suivant :

```
1 root=Tk()
2
```

```
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
cnv.pack(side=LEFT)
cnv.bind("<Button>", clic)

btn=Button(root, text="Nouvelle\npartie", command=new_game)
btn.pack(padx=20, pady=20)
```

ce qui affichera le bouton comme à la copie d'écran ci-dessus (j'ai ignoré les images dans le code).

La fonction pour rejouer

Il va falloir écrire la fonction `new_game` qui est référencée par l'option `command` du bouton et qui va cacher les cartes et les mélanger. Nous ne sommes pas en mesure de passer des arguments à cette fonction (ce n'est pas prévu par la lib Tkinter). Par ailleurs, comme cette fonction doit réinitialiser le jeu, elle va devoir modifier le contenu des objets suivants :

— le plateau `board`,

— le tableau des id des couvertures de logos,

— la variable `move` de gestion des coups.

Par ailleurs, elle doit accéder à certaines données comme l'image du dos de chaque logo et et des logos eux-mêmes.

Essayons le code suivant :

```
# Précédentes fonctions omises

def new_game():
    ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
    board=make_board()
    fill(cnv, images, board, cover, ids_cover)
    move[0]=move[1]=None

root=Tk()

cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
cnv.pack(side=LEFT)
cnv.bind("<Button>", clic)

btn=Button(root, text="Nouvelle\npartie", command=new_game)
btn.pack(padx=20, pady=20)

cover = PhotoImage(file="./images/cover.gif")
images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]

board=make_board()
fill(cnv, images, board, cover, ids_cover)

move=[None, None]
```

```
26 root.mainloop()
```

Ce code va fonctionner au départ (grâce à la ligne 23) mais si on clique sur le bouton, le code ne fonctionnera pas comme attendu. En effet, l'exécution de la ligne 4 ne va pas modifier la liste `ids_cover` de la ligne 20 et qui est utilisée par les autres fonctions du code. C'est analogue pour `board` (lignes 5 et 22). C'est différent pour `move` qui lui sera bien modifié. Par ailleurs, le plateau de jeu va bien être renouvelé (ligne 6). Le problème, c'est que les fonctions de jeu ne disposeront pas des nouvelles `id` des couvertures de logo (qui sont dans `ids_cover`) ni de la répartition des logos sur le plateau (qui sont dans `board`).

Pour remédier à cela, il suffit juste de déclarer en `global` les variables `ids_cover` et `board` (ligne 67 ci-dessous) :

jeu_avec_bouton_fonctions_global.py

```
1  from tkinter import *
2  from random import shuffle
3
4  PICT_SIZE=120
5  PAD=10
6  SIDE=PICT_SIZE+PAD
7
8  NB_LINES=4
9  NB_COLS=5
10 WIDTH=SIDE*NB_COLS
11 HEIGHT=SIDE*NB_LINES
12 X0=Y0=SIDE//2
13 NB_PICT=NB_LINES*NB_COLS//2
14 LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
15       'php', 'python', 'ruby', 'scratch']
16
17 def make_board():
18     L=[v % NB_PICT for v in range(2*NB_PICT)]
19     shuffle(L)
20     board=[]
21     k=0
22     for line in range(NB_LINES):
23         row=[]
24         for col in range(NB_COLS):
25             row.append(L[k])
26             k+=1
27         board.append(row)
28     return board
29
30 def fill(cnv, images, board, cover, ids_cover):
31     for line in range(NB_LINES):
32         for col in range(NB_COLS):
33             center=(X0+col*SIDE, Y0+line*SIDE)
34             k=board[line][col]
35             cnv.create_image(center, image=images[k])
36             ids_cover[line][col]=cnv.create_image(center, image=cover)
```

```
37
38 def hide(i, j, line,col):
39     ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
40     ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
41     move[0]=move[1]=None
42
43 def handle_move(line, col):
44     item=ids_cover[line][col]
45     cnv.delete(item)
46     if move[0] is None :
47         move[0]=(line, col)
48     else:
49         if move[0]==(line, col):
50             return
51         move[1]=(line, col)
52         i, j=move[0]
53         if board[i][j]==board[line][col]!=-1:
54             board[i][j]=board[line][col]=-1
55             move[0]=move[1]=None
56         else:
57             cnv.after(400, hide, i, j, line, col)
58
59 def clic(event):
60     if move[1] is not None:
61         return
62     col, line=event.x//SIDE, event.y//SIDE
63     if board[line][col]!=-1:
64         handle_move(line, col)
65
66 def new_game():
67     global ids_cover, board
68     ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
69     board=make_board()
70     fill(cnv, images, board, cover, ids_cover)
71     move[0]=move[1]=None
72
73 root=Tk()
74
75 cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
76 cnv.pack(side=LEFT)
77 cnv.bind("<Button>", clic)
78
79 btn=Button(root, text="Nouvelle\npartie", command=new_game)
80 btn.pack(padx=20, pady=20)
81
82 cover = PhotoImage(file="./images/cover.gif")
83 images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
84 ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
85
```

```
86 board=make_board()
87 fill(cnv, images, board, cover, ids_cover)
88
89 move=[None, None]
90 root.mainloop()
```

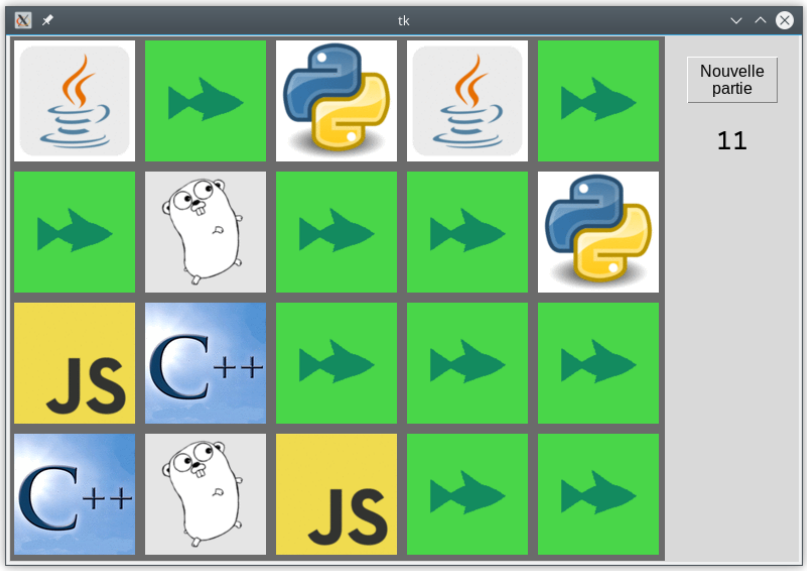
Noter qu'il n'est pas nécessaire de placer `move` en `global` (car la fonction modifie directement **son contenu**, cf. lignes 71 et 90).

Enfin, il serait possible d'éviter de déclarer en `global` les variables `ids_cover` et `board` (lignes 67) en changeant le code pour qu'il modifie le **contenu** de ces variables (même principe que pour `move`).

Pour finir, on pourrait souhaiter que tout le code soit enveloppé dans des fonctions (c'est pour l'instant incomplet, cf. lignes 73-90). C'est possible mais au prix de connaissances supplémentaires en Python (déclaration `nonlocal` au lieu de `global`). Je n'insiste pas sur ce point car la bonne façon de coder le jeu est d'utiliser une classe, cf. la fin du tutoriel.

Compteur de coups

Ajoutons un compteur de coups à notre memory game, un coup étant obtenu chaque fois qu'une paire de logos est rendue visible (provisoirement ou définitivement). En cours de jeu, l'interface aura alors la forme suivante :



Le compteur est placé dans une zone de texte obtenue en utilisant le widget **Label** (*étiquette* en français, terme que je n'emploierai pas).

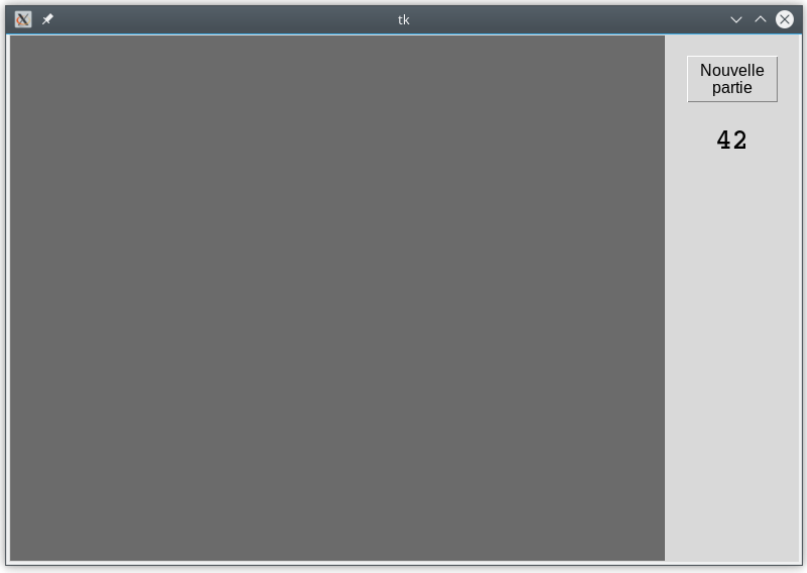
Construction du widget

Comme pour le bouton, il faut passer à la méthode `pack` les bonnes options pour que l'empilement des widgets se fasse là où on le souhaite. Voici le code nécessaire :

label.py

```
1  from tkinter import *
2
3  PICT_SIZE=120
4  PAD=10
5  SIDE=PICT_SIZE+PAD
6
7  NB_LINES=4
8  NB_COLS=5
9  WIDTH=SIDE*NB_COLS
10 HEIGHT=SIDE*NB_LINES
11
12 root=Tk()
13 cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
14 cnv.pack(side=LEFT)
15
16 btn=Button(root, text="Nouvelle\npartie", font="Arial 12")
17 btn.pack(padx=20, pady=20)
18
19 lbl=Label(root, text=42, font="courier 20 bold")
20 lbl.pack()
21
22 root.mainloop()
```

ce qui affiche :



J'ai juste rajouté le placement d'un label avec le constructeur Label (ligne 19) et il se trouve que le gestionnaire de géométrie pack (ligne 21) l'a placé, sans donner d'option particulière, à un endroit convenable. J'ai écrit un texte artificiel (42, cf. ligne 19). Parmi les options d'un label, on peut donner des indications sur la police et les fontes utilisées, cf. l'option `font` ligne 19. L'option `font` s'applique également au widget `Button` (cf. ligne 16) qui peut contenir du texte.

Mise à jour du compteur

Le compteur est mis-à-jour quand la 2e carte est retournée, ce qui se produit dans le code de la fonction `handle_move` :

```
1 def handle_move(line, col):
2     item=ids_cover[line][col]
3     cnv.delete(item)
4     if move[0] is None :
5         move[0]=(line, col)
6     else:
7         if move[0]==(line, col):
8             return
9         move[1]=(line, col)
10        i, j=move[0]
11        if board[i][j]==board[line][col]!=-1:
12            board[i][j]=board[line][col]=-1
13            move[0]=move[1]=None
14        else:
15            cnv.after(400, hide, i, j, line, col)
```

Plus précisément, il faudra mettre à jour juste entre la ligne 8 et la ligne 9.

Je rappelle le code du label :

```
1 lbl=Label(root, text=42, font="courier 20 bold")
2 lbl.pack()
```

Pour mettre à jour, il suffirait de changer la valeur de l'option `text`, ce que Tkinter permettrait. Mais, je vais en fait indiquer une méthode plus typique sous Tkinter et qui consiste à utiliser ce qu'on appelle des « variables de contrôle » (*control variable*).

Pour cela, on va définir un objet Tkinter de type `IntVar` : c'est un objet ayant un « contenu » entier et que certains widgets, comme un label, peuvent mettre à jour *automatiquement*. Notre compteur va donc être défini sous cette forme :

```
1 cpt = IntVar()
```

qui définit `cpt` comme un objet de type `IntVar`. Quand, en interne, `cpt` est mis-à-jour, on aimerait que cette mise-à-jour soit visible sur le label. Pour que ce soit possible, il faut donner au label une option prévue à cet effet et du nom (obligatoire) de `textvariable` et que l'on fait pointer vers notre compteur :

```
1 cpt = IntVar()
2 lbl=Label(root, textvariable=cpt, font="courier 20 bold")
3 lbl.pack()
```

Pour lire ou donner une valeur au compteur, il faut utiliser des « getter » et « setter » :

— la valeur du compteur s'obtient par `cpt.get()` (le « getter »),

— la valeur du compteur peut être modifiée par une instruction de la forme : `cpt.set(42)` qui placera 42 dans le compteur (le « setter »).

Nous devons initialiser notre compteur à 0, donc le code évolue en :

```
cpt = IntVar()
cpt.set(0)
lbl=Label(root, textvariable=cpt, font="courier 20 bold")
lbl.pack()
```

Maintenant, il ne reste plus qu'à mettre à jour, au bon endroit, le compteur. Cela peut se faire de la manière suivante :

```
v=cpt.get()
cpt.set(v + 1)
```

— On récupère la valeur v du compteur.

— La nouvelle valeur est `v + 1` est il faut la placer dans le compteur avec le setter.

— On peut remplacer les deux lignes par `cpt.set(cpt.get() + 1)`.

Il ne reste plus qu'à placer cette instruction au bon endroit (ligne 9 ci-dessous), dans le corps de la fonction `handle_move` :

```
def handle_move(line, col):
    item=ids_cover[line][col]
    cnv.delete(item)
    if move[0] is None :
        move[0]=(line, col)
    else:
        if move[0]==(line, col):
            return
        cpt.set(cpt.get() + 1)
        move[1]=(line, col)
        i, j=move[0]
        if board[i][j]==board[line][col]!=-1:
            board[i][j]=board[line][col]=-1
            move[0]=move[1]=None
        else:
            cnv.after(400, hide, i, j, line, col)
```

Le code complet

Le code complet est le suivant :

jeu_avec_bouton_compteur.py

```
from tkinter import *
from random import shuffle

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=4
NB_COLS=5
WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
```

```
12 X0=Y0=SIDE//2
13 NB_PICT=NB_LINES*NB_COLS//2
14 LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
15       'php', 'python', 'ruby', 'scratch']
16
17 def make_board():
18     L=[v % NB_PICT for v in range(2*NB_PICT)]
19     shuffle(L)
20     board=[]
21     k=0
22     for line in range(NB_LINES):
23         row=[]
24         for col in range(NB_COLS):
25             row.append(L[k])
26             k+=1
27         board.append(row)
28     return board
29
30 def fill(cnv, images, board, cover, ids_cover):
31     for line in range(NB_LINES):
32         for col in range(NB_COLS):
33             cnv.create_image(X0+col*SIDE, Y0+line*SIDE,
34                              image=images[board[line][col]])
35             ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE,
36                                                   image=cover)
37
38 def lin_col(x, y):
39     return (y//SIDE, x//SIDE)
40
41 def hide(i, j, line,col):
42     ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE,
43                                      image=cover)
44     ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE,
45                                           image=cover)
46     move[0]=move[1]=None
47
48 def handle_move(line, col):
49     item=ids_cover[line][col]
50     cnv.delete(item)
51     if move[0] is None :
52         move[0]=(line, col)
53     else:
54         if move[0]==(line, col):
55             return
56         cpt.set(cpt.get() + 1)
57         move[1]=(line, col)
58         i, j=move[0]
59         if board[i][j]==board[line][col]!=-1:
```

```
            board[i][j]=board[line][col]=-1
            move[0]=move[1]=None
        else:
            cnv.after(400, hide, i, j, line, col)

def clic(event):
    if move[1] is not None:
        return
    col, line=event.x//SIDE, event.y//SIDE
    if board[line][col]!=-1:
        handle_move(line, col)

def new_game():
    global ids_cover, board
    ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
    board=make_board()
    fill(cnv, images, board, cover, ids_cover)
    move[0]=move[1]=None
    cpt.set(0)

root=Tk()
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
cnv.pack(side=LEFT)
cnv.bind("<Button>", clic)

btn=Button(root, text="Nouvelle\npartie", font="Arial 12", command=new_game)
btn.pack(padx=20, pady=20)

cpt = IntVar()
cpt.set(0)
lbl=Label(root, textvariable=cpt, font="courier 20 bold")
lbl.pack()

cover = PhotoImage(file="./images/cover.gif")
images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]

board=make_board()
fill(cnv, images, board, cover, ids_cover)

move=[None, None]
root.mainloop()
```

Variante en gardant une commande `text`

Une variable de contrôle est intéressante parce qu'elle permet de mettre à jour automatiquement **plusieurs** widgets devant référencer la même valeur. Dans le code ci-dessus, on ne met à jour qu'une seule valeur, donc l'intérêt est limité. Il existe une façon plus directe de parvenir au même résultat. Initialement, le label était défini ainsi :

```
lbl=Label(root, text=42, font="courier 20 bold")
lbl.pack()
```

et c'est via l'option `text` que le texte à afficher est visible. Or, on peut facilement accéder aux options d'un widget par la syntaxe d'un dictionnaire. Par exemple, le contenu du label ci-dessus peut-être accédé via `lbl[text]`, comme le montre cet exemple :

label_read.py

```
from tkinter import *

root=Tk()
lbl=Label(root,text="42", font="courier 20 bold")
lbl.pack(padx=50, pady=50)

print(lbl['text'])

root.mainloop()
```

qui crée cette fenêtre :

et qui affiche en console

```
42
```

On a aussi en accès en écriture. Par exemple si on veut changer la valeur du label en 2020, il suffit d'écrire

```
lbl['text']=2020
```

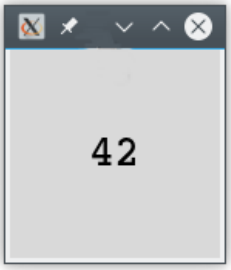
Par exemple, le code ci-dessus permet, après deux secondes d'attente, de faire basculer le contenu du label de 42 vers 2020 :

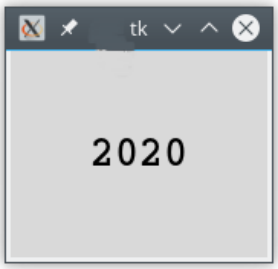
label_write.py

```
from tkinter import *

root=Tk()
lbl=Label(root,text=42, font="courier 20 bold")
lbl.pack(padx=50, pady=50)

def changer():
    lbl['text']=2020

```

```
10 root.after(2000, changer)
11
12 root.mainloop()
```

et qui affiche successivement :

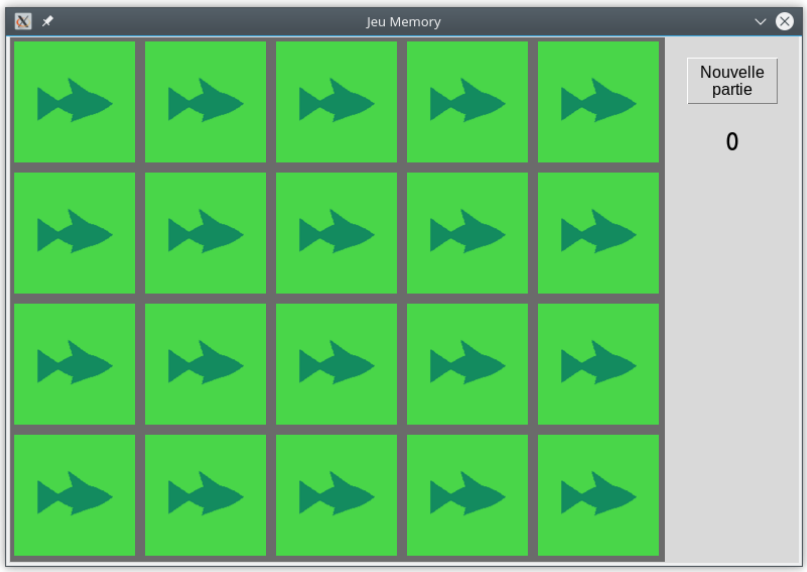




On peut alors adapter ce code pour mettre à jour dans le memory game notre compteur de coups.

Fenêtre améliorée

On va améliorer la fenêtre du jeu sur deux points (mineurs) :



D'une part, on pourrait souhaiter placer un titre de fenêtre plus approprié que le titre par défaut. Cela peut facilement se changer.

D'autre part, dans les versions précédentes du jeu, si vous modifiez, le plus souvent sans le vouloir, la dimension de la fenêtre, vous allez voir que le plateau ne s'adapte pas aux nouvelles dimensions de la fenêtre, ce que l'on peut considérer comme un comportement non souhaité et de toute façon qu'il serait compliqué à coder dans la mesure où il faudrait des images redimensionnables. Pour remédier à cela, on bloque la taille de la fenêtre maîtresse. Le bouton de réduction de la fenêtre sera alors inactif (sous Windows) voire aura même disparu (cf. copie d'écran ci-dessus).

Voici un code qui apporte ces deux menues améliorations (lignes 77-78) :

jeu_avec_bouton_compteur_fen_bloquee.py

```
1  from tkinter import *
2  from random import shuffle
3
4  PICT_SIZE=120
5  PAD=10
6  SIDE=PICT_SIZE+PAD
7
8  NB_LINES=4
9  NB_COLS=5
10 WIDTH=SIDE*NB_COLS
11 HEIGHT=SIDE*NB_LINES
12 XO=YO=SIDE//2
13 NB_PICT=NB_LINES*NB_COLS//2
14 LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
15       'php', 'python', 'ruby', 'scratch']
16
17 def make_board():
18     L=[v % NB_PICT for v in range(2*NB_PICT)]
```

```
    shuffle(L)
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, images, board, cover, ids_cover):
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=images[board[line][col]
            ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=co

def lin_col(x, y):
    return (y//SIDE, x//SIDE)

def hide(i, j, line,col):
    ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
    ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
    move[0]=move[1]=None

def handle_move(line, col):
    item=ids_cover[line][col]
    cnv.delete(item)
    if move[0] is None :
        move[0]=(line, col)
    else:
        if move[0]==(line, col):
            return
        cpt.set(cpt.get() + 1)
        move[1]=(line, col)
        i, j=move[0]
        if board[i][j]==board[line][col]!=-1:
            board[i][j]=board[line][col]=-1
            move[0]=move[1]=None
        else:
            cnv.after(400, hide, i, j, line, col)

def clic(event):
    if move[1] is not None:
        return
    col, line=event.x//SIDE, event.y//SIDE
    if board[line][col]!=-1:
        handle_move(line, col)
```

```
67
68 def new_game():
69     global ids_cover, board
70     ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
71     board=make_board()
72     fill(cnv, images, board, cover, ids_cover)
73     move[0]=move[1]=None
74     cpt.set(0)
75
76 root=Tk()
77 root.resizable(False, False)
78 root.title("Jeu Memory")
79 cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
80 cnv.pack(side=LEFT)
81 cnv.bind("<Button>", clic)
82
83 btn=Button(root, text="Nouvelle\npartie", font="Arial 12", command=new_game)
84 btn.pack(padx=20, pady=20)
85
86 cpt = IntVar()
87 cpt.set(0)
88 lbl=Label(root, textvariable=cpt, font="courier 20 bold")
89 lbl.pack()
90
91 cover = PhotoImage(file="./images/cover.gif")
92 images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
93 ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
94
95 board=make_board()
96 fill(cnv, images, board, cover, ids_cover)
97
98 move=[None, None]
99 root.mainloop()
```

Une fonction d'initialisation

Si on examine la fin du code fichier `jeu_avec_bouton_compteur_fen_bloquee.py` :

```
1 def new_game():
2     global ids_cover, board
3     ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
4     board=make_board()
5     fill(cnv, images, board, cover, ids_cover)
6     move[0]=move[1]=None
7     cpt.set(0)
8
9 # ...
10 # code omis
```

```
11 # ...
12
13 cpt = IntVar()
14 cpt.set(0)
15 lbl=Label(root, textvariable=cpt, font="courier 20 bold")
16 lbl.pack()
17
18 cover = PhotoImage(file="./images/cover.gif")
19 images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
20 ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
21
22 board=make_board()
23 fill(cnv, images, board, cover, ids_cover)
24
25 move=[None, None]
26 root.mainloop()
```

on se rend compte qu'il y a un certain nombre de répétitions :

— lignes 7 et 14,

— lignes 4 et 22,

— lignes 6 et 25.

Plutôt que de lancer le jeu puis de remettre à zéro certains paramètres quand on fait une nouvelle partie, autant écrire une fonction qui

— initialise la 1re partie

— réinitialise pour chaque nouvelle partie.

On va donc remplacer la fonction `new_game` par une fonction `init`. Il suffit de retirer les lignes en double et de placer en variables globales les variables et de penser à appeler la fonction `init` avant le début d'une partie. Ce qui donne le code complet suivant :

memory.py

```
1 from tkinter import *
2 from random import shuffle
3
4 PICT_SIZE=120
5 PAD=10
6 SIDE=PICT_SIZE+PAD
7
8 NB_LINES=4
9 NB_COLS=5
10 WIDTH=SIDE*NB_COLS
11 HEIGHT=SIDE*NB_LINES
12 X0=Y0=SIDE//2
13 NB_PICT=NB_LINES*NB_COLS//2
14 LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
15       'php', 'python', 'ruby', 'scratch']
16
17 def make_board():
```

```
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, images, board, cover, ids_cover):
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=images[board[line][col]
            ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=c

def lin_col(x, y):
    return (y//SIDE, x//SIDE)

def hide(i, j, line,col):
    ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
    ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
    move[0]=move[1]=None

def handle_move(line, col):
    item=ids_cover[line][col]
    cnv.delete(item)
    if move[0] is None :
        move[0]=(line, col)
    else:
        if move[0]==(line, col):
            return
        cpt.set(cpt.get() + 1)
        move[1]=(line, col)
        i, j=move[0]
        if board[i][j]==board[line][col]!=-1:
            board[i][j]=board[line][col]=-1
            move[0]=move[1]=None
        else:
            cnv.after(400, hide, i, j, line, col)

def clic(event):
    if move[1] is not None:
        return
    col, line=event.x//SIDE, event.y//SIDE
    if board[line][col]!=-1:
```

```
66          handle_move(line, col)
67
68  def init():
69      global ids_cover, board, move
70      ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
71      board=make_board()
72      fill(cnv, images, board, cover, ids_cover)
73      move=[None, None]
74      cpt.set(0)
75
76  root=Tk()
77  root.resizable(False, False)
78  root.title("Jeu Memory")
79  cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
80  cnv.pack(side=LEFT)
81  cnv.bind("<Button>", clic)
82
83  btn=Button(root, text="Nouvelle\npartie", font="Arial 12", command=init)
84  btn.pack(side=TOP, padx=20, pady=20)
85
86
87  cpt = IntVar()
88  lbl=Label(root, textvariable=cpt, font="courier 20 bold")
89  lbl.pack()
90
91  cover = PhotoImage(file="./images/cover.gif")
92  images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
93
94  init()
95
96  root.mainloop()
```

— Lignes 68 et 83 : la fonction `new_game` a été remplacée par la fonction `init`.

— Ligne 94 : la fonction `init` est appelée juste avant de commencer une partie. Elle initialise tous les éléments spécifiques à une partie :

  — `ids_cover` : les identités des images de couverture

  — `board` : le plateau de jeu

  — `move` : la liste des drapeaux de clics

  — le compteur de coups.

Installer Pygame sous Windows

Pygame est une bibliothèque de jeux. Dans ce qui suit, on va expliquer comment installer Pygame sous Windows 10 et sous Linux.

Installation sous Windows 10

Je suppose que vous avez installé Python depuis le site python.org. Probablement que ce qui suit ne s'applique pas si vous avez installé Python via Anaconda auquel cas il faut effectuer ce qui

va suivre depuis la ligne de commande propre à Anaconda. Je suppose aussi que vous n'utilisez pas Python sous un environnement virtuel (virtualenv).

Le principe d'installation est assez simple : on tape une ligne de commande dans un shell et cela installe automatiquement Pygame.

Ce qui suit suppose que vous travaillez avec une version 3 de Python.

Pygame serait-il déjà installé ?

Pour le savoir, ouvrir IDLE, l'éditeur par défaut de Python (Menu Démarrer > Python > IDLE). Devant le prompt Python avec les trois chevrons écrire :

```
import pygame
```

Si vous avez un message d'erreur, c'est que Pygame n'est pas installé ou pas accessible.

Le programme `pip` est-il déjà installé ?

Tout ce qui suit suppose que vous disposez du programme `pip`. Vous devez donc vérifier que `pip` est installé sur votre système.

`pip` est un programme d'installation de paquets Python. Son principe est d'aller télécharger sur un dépôt Internet un programme Python et d'installer ce programme sur votre système.

Ouvrir une ligne de commande Windows ; pour cela, taper `cmd` dans Cortana, ce qui devrait montrer une icône portant le nom d'« invite de commandes », cliquer sur l'icône et un terminal noir, avec un prompt devrait s'ouvrir. Il est encore plus simple de taper `command.exe` dans la barre de recherche de Cortana. Le terminal obtenu est un terminal système et pas un terminal propre à Python. Dans ce terminal, écrire

```
pip
```

puis appuyer sur la touche `Entrée`. Si la réponse est :

```
'pip' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.
```

c'est que `pip` n'est pas installé (voir plus loin pour savoir comment y remédier).

La méthode la plus simple

Ouvrir une ligne de commande Windows ; pour cela, taper `cmd` dans Cortana, ce qui devrait montrer une icône portant le nom d'« invite de commandes », cliquer sur l'icône et un terminal noir, avec un prompt devrait s'ouvrir. Ce terminal est un terminal système et pas un terminal propre à Python. Dans ce terminal, écrire

```
pip install pygame
```

puis appuyer sur la touche `Entrée`.

`pip` est un programme d'installation de paquets Python. Son principe est d'aller télécharger sur un dépôt Internet un programme Python et d'installer ce programme sur votre système.

Si le programme `pip` est reconnu par votre système, le déroulé de l'installation devrait être retranscrit dans le terminal et en conclusion vous devriez lire la version de Pygame qui a été installée, par exemple :

```
Successfully installed pygame-1.9.3
```

Pour vérifier que Pygame est bien installé, ré-ouvrir IDLE et devant les 3 chevron du prompt Python écrire, en minuscule :

```
import pygame
```

et aucun message d'erreur ne devrait apparaître. Vous pouvez même déterminer depuis le prompt Python la version de Pygame qui est installée, en tapant :

```
pygame.version.ver
```

ce qui devrait afficher la version, par exemple

```
'1.9.3'
```

Installation via un fichier wheel

Il se peut que la méthode précédente échoue. Vous pouvez alors télécharger sur Internet un binaire d'extension `whl` (ce qui signifie "wheel") contenant Pygame et l'installer comme ci-dessus avec `pip`. L'ensemble se fait en trois étapes :

Étape 1 : Déterminer l'adressage de votre version de Python

Quand vous téléchargez Python depuis le site `python.org`, il est disponible en version 32 bits ou 64 bits. Il ne faut pas confondre la version 32 ou 64 bits de Python avec l'adressage de votre version de Windows 10 qui elle est très probablement sous 64 bits.

Par défaut, c'est la version 32 bits qui est installée. Mais, pour connaître le mode d'adressage de votre version de Python, ouvrir IDLE et regarder tout en haut de la fenêtre, à la fin de la ligne, ce qui est indiqué, soit "32 bit (Intel)" soit "64 bit (AMD64)" ce qui vous précise votre version. Vous devriez aussi lire votre version de Python, par exemple, Python 3.6.

Étape 2 : télécharger le fichier wheel

Dans Google, taper "unofficial binaries Pygame" et cliquer sur le premier lien transmis par Google. On arrive sur le site Unofficial Windows Binaries for Python Extension Packages. Une fois sur le site, chercher la section Pygame qui contient une suite de fichiers "wheel" d'extension `whl` pour différentes versions de Pygame.

Puis télécharger le fichier correspondant à votre système. Pour illustrer la suite, je vais supposer que vous êtes sous Python 3.5 en 32 bits,et donc que vous téléchargez le fichier nommé `pygame-1.9.4-cp35-cp35m-win32.whl`, le code `35` devant être compris comme se référant à Python 3.5 et le code `win32` se référant à 32 bits ; ce fichier est un binaire contenant la version 1.9.4 de Pygame.

Étape 3 : installer le fichier wheel

— Une fois le fichier téléchargé, ouvrir le répertoire de téléchargement et débrouillez-vous pour obtenir le nom complet du dossier où se trouve le fichier. Le plus simple pour cela est de cliquer sur la barre d'adresse du dossier et de copier le nom complet du dossier, typiquement `C:\Users\Moi\Downloads`.

— Ouvrir une invite de commandes, par exemple en passant par Cortana et y tapant `cmd`. Taper `cd` dans la ligne de commandes (ce qui signifie "change directory") puis après un espace, coller dans la ligne de commandes le nom de dossier que vous aviez copié, puis taper sur la touche `Entrée` : la ligne de commande est alors placée dans le dossier où se trouve le fichier `whl` à installer.

— appeler le programme `pip` sur le fichier que vous avez téléchargé. Pour cela taper dans la ligne de commande `pip` suivi d'un espace puis indiquer le nom du fichier, dans notre exemple,

c'était `pygame-1.9.4-cp35-cp35m-win32.whl`. Inutile de tout taper lettre par lettre, il suffit de taper les 4 ou 5 premières lettres et ensuite d'appuyer sur la touche TAB et l'invite de commande complètera avec le bon nom de fichier.

— Valider en appyant sur la touche `Entrée` et sauf anaomalie, cela devrait installer Pygame sur votre système. Vérifier en ouvrant IDLE et en tapant `import pygame`.

Que faire si `pip` n'est pas reconnu sur votre système ?

En principe, depuis la version 3.4 de Python, le programme `pip` est disponible sur votre système car installé en même temps que Python. **Toutefois**, si lors de l'installation de Python, vous avez oublié de cocher la case autorisant le placement des répertoires des binaires Python dans le PATH du système, `pip` ne sera pas reconnu.

Vous pouvez d'ailleurs vérifier en allant voir si le fichier `pip.exe` est présent dans le répértoire des scripts Python. On trouve ce répertoire à une adresse du type

```
C:\Users\MonNom\AppData\Local\Programs\Python\Python37-32\Scripts
```

Si `pip` n'est pas reconnu sur votre système, le plus probable est que le PATH soit incomplet. Deux solutions :

— soit vous désinstallez Python et vous le réinstallez en prenant soin de cocher dans le panneau d'installation, tout au début du processus d'installation, la case d'inclusion de Python au PATH,

— soit vous complétez vous-même le PATH en vous aidant par exemple de Add PIP to the Windows Environment Variables

Audio sous Tkinter avec Pygame

Nativement, Tkinter ne prend pas en charge la diffusion de flux audio (fichiers mp3, wav, etc). Il faut faire appel à une bibliothèque tierce pour réaliser l'incorporation d'un flux audio dans un programme Tkinter.

La bibliothèque de jeux Pygame prend en charge l'audio que ce soit sous Windows, Linux ou OsX. On peut donc l'associer à Tkinter pour faire émettre du son. C'est ce qu'on va faire ci-dessous. Toutefois, d'autres choix seraient possibles comme Pyglet.

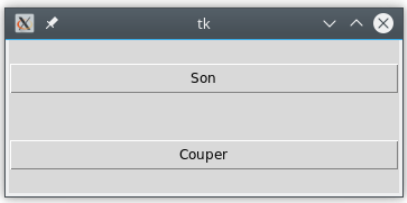
Pygame prend en charge les fichiers de format `wav`. Pour l'utilisation du format `mp3`, voir tout à la fin de cette unité consacrée à l'audio avec Pygame sous Tkinter. Le programme `audio.py` ci-dessous dépend d'un fichier `clap.wav` placé à côté du fichier `audio.py`. Le fichier wav est téléchargeable ICI. C'est un programme minimal qui associe Tkinter et un flux audio géré par Pygame :

`audio.py`

```
1 from tkinter import *
2 import pygame
3
4 pygame.mixer.init()
5
6 mon_audio=pygame.mixer.Sound("clap.wav")
7
8 def lancer():
```

```
9       mon_audio.play(-1)
10
11  def couper():
12      mon_audio.stop()
13
14  fen = Tk()
15
16  Button(fen,text="Son",command=lancer, width=40).pack(pady=20)
17  Button(fen,text="Couper",command=couper, width=40).pack(pady=20)
18
19  fen.mainloop()
```

L'interface graphique a l'allure suivante :



Le bouton du haut active un son et ce son tourne en boucle, le bouton du bas interrompt le flux audio.

Commentaire de code

- Ligne 2 : on importe Pygame
- Ligne 4 : on initialise le module mixer de Pygame (le module qui gère le son)
- Ligne 6 : on charge le fichier dont on veut écouter le flux audio en indiquant l'adresse du fichier. Cela crée un objet qui permet d'accéder au flux vidéo.
- Ligne 1 : on importe Tkinter.
- Les éléments graphiques :
  - ligne 14 : la fenêtre
  - lignes 16-17 : deux boutons
  - lignes 9 et 13 : les fonctions associées aux boutons
- Ligne 9 : le son est joué (fonction `play`). L'argument `-1` a pour effet que le flux audio est joué en boucle, indéfiniment si on ne l'interrompt pas. Le volume par défaut est maximal.
- Ligne 12 : interruption du flux audio.

Volume du son

Il est possible de définir un volume sonore avec la méthode `set_volume`, par exemple placer entre les lignes 8 et 9 l'instruction :

```
mon_audio.set_volume(0.5)
```

La méthode `set_volume` accepte en un argument un nombre flottant entre 0 (aucun son) et 1 (son d'intensité maximale), donc `0.5` correspond à une intensité médiane.

Ne pas jouer en boucle

On souhaite parfois qu'un son soit joué non pas en boucle mais juste une seule fois. Pour cela il suffit de passer un argument autre que `-1` à la fonction `pygame.mixer.music.play`. Par exemple, `pygame.mixer.music.play()` va jouer le fichier audio une seule fois. Ou encore `pygame.mixer.music.play(5)` va le jouer 6 fois (et non pas 5, au moins sous Linux en tous cas).

Pré-initialisation

Parfois, le lancement du flux audio se lance avec un certain retard ou encore le flux audio est ralenti. Pour y remédier, essayer de placer avant l'appel à `pygame.mixer.init` un appel à la fonction `pre_init`, par exemple :

```
pygame.mixer.pre_init(44100, -16, 1, 512)
```

Les valeurs ci-dessous ont été retranscrises d'une réponse sur StackOverFlow.

Documentation

La bibliothèque Pygame permet une prise de l'audio. La documentation officielle de Pygame du package gérant le son est disponible sur pygame.mixer.music. Il pourra aussi être utile de consulter les fichiers d'exemples proposés dans le code source de Pygame.

Sortie correcte de Pygame

Si on reprend le programme `audio.py`, qu'on clique sur le bouton **Son** alors on entend le flux audio. Et si on interrompt alors le programme en cliquant sur la croix de la fenêtre alors le flux audio sera toujours audible. Pourquoi ? Parce que la croix ferme uniquement ce qui est de la responsabilité de Tkinter, ce qui n'est pas le cas de la gestion du son.

De la même façon qu'on a initialisé l'audio dans Pygame avec pygame.mixer.init, il faut quitter proprement Pygame. Pour cela, Pygame propose la méthode pygame.quit. Ci-dessous, le programme `audio.py` a été modifié pour que la sortie de Pygame soit correcte :

audio_sortie.py

```
1  from tkinter import *
2  import pygame
3
4  pygame.mixer.init()
5
6  mon_audio=pygame.mixer.Sound("clap.wav")
7
8
9  def lancer():
10     mon_audio.set_volume(1)
11     mon_audio.play(-1)
12
13 def couper():
14     mon_audio.stop()
15
16 def quitter():
```

```
17      pygame.quit()
18      fen.destroy()
19
20  fen = Tk()
21
22  Button(fen,text="Son",command=lancer, width=40).pack(pady=20)
23  Button(fen,text="Couper",command=couper, width=40).pack(pady=20)
24  fen.protocol("WM_DELETE_WINDOW", quitter)
25
26  fen.mainloop()
```

— Ligne 17 : on quittera Pygame avec la fonction dédiée `quit`.

— Ligne 24 : Cette ligne détecte une tentative de fermeture de la fenêtre en cliquant sur la croix. Fermer la fenêtre est l'événement `"WM_DELETE_WINDOW"`. Lorsque cet événement est détecté par Tkinter, la fonction `quitter`, définie lignes 16-18, sera automatiquement appelée.

— Lignes 16-18 : fonction appelée pour fermer proprement l'interface graphique et Pygame. `pygame.quit` ferme Pygame et libère les ressources que Pygame utilisait. De même, `fen.destroy()` fait disparaître le fenêtre `fen`.

Le cas du format mp3

Pour les flux audio mp3, la documentation de Pygame précise que la prise en charge est limitée. Le conseil qui est souvent donné est de convertir ses fichiers mp3 au format wav ou encore au format ogg (qui, comme le format mp3, est compressé, à la différence du format wav) qui eux sont pleinement pris en charge par Pygame.

Il semble toutefois que l'on puisse jouer des fichiers mp3 sous Tkinter en utilisant Pygame, que ce soit sous Windows comme sous Linux. Les exemples qui suivent utiliseront le fichier `clap.mp3` téléchargeable ICI et qu'on placera à côté du code source Python.

D'abord, hors de Tkinter, on peut écouter un fichier mp3 sous Python avec Pygame en suivant l'exemple suivant :

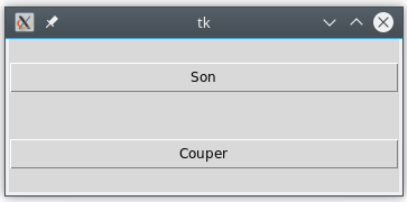
```
1  import pygame
2
3  pygame.init()
4
5  pygame.mixer.music.load("clap.mp3")
6  pygame.mixer.music.play()
7
8  while pygame.mixer.music.get_busy():
9      pass
10
11 pygame.quit()
```

Cet exemple suit la démo donnée dans le code-source de Pygame. On notera que la méthode `Sound` applicable au format `wav` ne semble pas s'appliquer au format mp3. On prendra aussi garde que certains fichiers audio au format `mp3`, parfaitement audibles dans un lecteur audio, ne seront pas décodés par Pygame, comme rappellé dans cette discussion Pygame fails to play some mp3 files but not others.

La boucle aux lignes 8-9 doit être présente sinon le son n'est pas émis et le programme s'interrompt.

Pour finir, je reprends ci-dessous l'application utilisée sous Tkinter pour décrire l'usage d'un fichier wav, sans commentaire supplémentaire puisque les codes sont assez proches :

```
from tkinter import *
import pygame

pygame.mixer.init()

pygame.mixer.music.load("clap.mp3")

def lancer():
    pygame.mixer.music.play(-1)

def couper():
    pygame.mixer.music.stop()

fen = Tk()

Button(fen,text="Son",command=lancer, width=40).pack(pady=20)
Button(fen,text="Couper",command=couper, width=40).pack(pady=20)

fen.mainloop()
```



Il semblerait que le module mixer s'exécute dans un thread distinct du thread principal dans lequel s'éxecute Tkinter.

Applaudissements en fin de jeu

On souhaite que le joueur entende des applaudissements lorsqu'il a réussi à découvrir toutes les cartes du plateau.

Quelques remarques pour parvenir à l'objectif :

— Pygame doit avoir été installé, cf. mon tutoriel d'installation pour windows et pour Linux.

— On dispose d'un fichier d'applaudissements `clap.wav`, qui sera placé à côté du fichier Python à écrire (le fichier est disponible dans les sources en haut de la page)

— Les applaudissements sont déclenchés une fois que toutes les cartes ont été appariées : il faut donc disposer d'une variable, ci-dessous appelée `couples`, qui surveille le nombre de couplages réalisés.

Ci-dessous un code basé sur le fichier `memory.py` qui incorpore la génération d'applaudissements :

`memory_applaudir.py`

```
from tkinter import *
from random import shuffle
import pygame

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=2
NB_COLS=3
WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
X0=Y0=SIDE//2
NB_PICT=NB_LINES*NB_COLS//2
LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def make_board():
    L=[v % NB_PICT for v in range(2*NB_PICT)]
    shuffle(L)
    board=[]
    k=0
    for line in range(NB_LINES):
        row=[]
        for col in range(NB_COLS):
            row.append(L[k])
            k+=1
        board.append(row)
    return board

def fill(cnv, images, board, cover, ids_cover):
    for line in range(NB_LINES):
        for col in range(NB_COLS):
            cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=images[board[line][col]
            ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=c

def lin_col(x, y):
    return (y//SIDE, x//SIDE)

def hide(i, j, line,col):
    ids_cover[i][j]=cnv.create_image(X0+j*SIDE, Y0+i*SIDE, image=cover)
    ids_cover[line][col]=cnv.create_image(X0+col*SIDE, Y0+line*SIDE, image=cover)
    move[0]=move[1]=None

def handle_move(line, col):
```

```
    global couples
    item=ids_cover[line][col]
    cnv.delete(item)
    if move[0] is None :
        move[0]=(line, col)
    else:
        if move[0]==(line, col):
            return
        cpt.set(cpt.get() + 1)
        move[1]=(line, col)
        i, j=move[0]
        if board[i][j]==board[line][col]!=-1:
            board[i][j]=board[line][col]=-1
            move[0]=move[1]=None
            couples=couples+1
            if couples==NB_PICT:
                pygame.mixer.music.play(-1)
        else:
            cnv.after(400, hide, i, j, line, col)

def clic(event):
    if move[1] is not None:
        return
    col, line=event.x//SIDE, event.y//SIDE
    if board[line][col]!=-1:
        handle_move(line, col)

def init():
    global ids_cover, board, move, couples
    ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
    board=make_board()
    fill(cnv, images, board, cover, ids_cover)
    move=[None, None]
    cpt.set(0)
    couples=0
    pygame.mixer.music.stop()

root=Tk()
root.resizable(False, False)
root.title("Jeu Memory")
cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
cnv.pack(side=LEFT)
cnv.bind("<Button>", clic)

btn=Button(root, text="Nouvelle\npartie", font="Arial 12", command=init)
btn.pack(side=TOP, padx=20, pady=20)

cpt = IntVar()
```

```
94 lbl=Label(root, textvariable=cpt, font="courier 20 bold")
95 lbl.pack()
96
97 pygame.mixer.init()
98 pygame.mixer.music.load("clap.wav")
99
100 cover = PhotoImage(file="./images/cover.gif")
101 images=[PhotoImage(file="./images/%s.gif" %filename) for filename in LANG]
102
103 init()
104
105 root.mainloop()
```

— Initialisations de Pygame :
  — ligne 3 : importation de Pygame
  — ligne 97 : initialisation du module pygame.mixer
  — ligne 98 : chargement du fichier audio `"clap.wav"`
— Ligne 80 : à chaque partie, une variable `couples` est initialisée à 0. La variable `coupleS` sert à compter le nombre de couplages corrects réalisés par le joueur.
— Ligne 60 : chaque fois qu'un couplage correct est réalisé, cf. ligne 57, la variable `couples` est incrémentée.
— Ligne 74 : la variable `couples` doit être déclarée en global puisqu'elle est réinitialisée à chaque partie dans la fonction `init`
— Ligne 46 : la variable `couples` modifiée dans la fonction `handle_move` est la même que celle qui est déclarée dans la fonction `init`. Il faut donc la déclarer en global.
— Lignes 61-62 : lorsque que le nombre de couplages corrects atteint le nombre de cartes distinctes, c'est que le joueur a apparié toutes les cartes : il faut donc envoyer les applaudissements.

Réorganiser le code avec une classe

Nous avons rencontré des difficultés à écrire tout le code dans des fonctions. Cela nous a contraint à utiliser des variables globales ou des variables `nonlocal` pour un code finalement assez peu lisible, avec des définitions de fonctions emboîtées. La difficulté vient de

— la présence de la mainloop : si on en sort, l'interface graphique disparaît ;
— ne pas avoir la main sur les fonctions de rappel (`clic`, `new_game`) : les paramètres nous sont imposés, les appels sont effectués automatiquement par le toolkit et il n'y a pas de return récupérable.

On remédie largement à ces problèmes et on améliore la lisibilité du code en le plaçant dans une ou plusieurs classes. En ajoutant diverses améliorations (commentées ci-dessous) au fichier `memory_applaudir.py`, cela pourrait donner ceci :

`jeu_classe.py`

```
1 from random import shuffle
2 from tkinter import Button, Canvas, IntVar, LEFT, Label, PhotoImage, TOP, Tk
```

```
import pygame

PICT_SIZE=120
PAD=10
SIDE=PICT_SIZE+PAD

NB_LINES=4
NB_COLS=5
WIDTH=SIDE*NB_COLS
HEIGHT=SIDE*NB_LINES
XO=YO=SIDE//2
NB_PICT=NB_LINES*NB_COLS//2
LANG=['c', 'cpp', 'go', 'java', 'js', 'ocaml',
      'php', 'python', 'ruby', 'scratch']

def make_board(n, p):
    P=n*p
    N=P//2
    L=[v % N for v in range(P)]
    shuffle(L)
    board=[[L[line*p+col] for col in range(p)] for line in range(n)]
    return board

class Memory:

    def __init__(self):
        root=Tk()
        root.resizable(False, False)
        root.title("Jeu Memory")
        self.cnv=Canvas(root, width=WIDTH, height=HEIGHT, bg='gray42')
        self.cnv.pack(side=LEFT)
        self.cnv.bind("<Button>", self.clic)

        self.btn=Button(root, text="Nouvelle\npartie",
                        font="Arial 12", command=self.new_game)
        self.btn.pack(side=TOP, padx=20, pady=20)

        self.cpt = IntVar()
        self.lbl=Label(root, textvariable=self.cpt, font="courier 20 bold")
        self.lbl.pack()

        pygame.mixer.init()
        pygame.mixer.music.load("clap.wav")

        self.cover = PhotoImage(file="./images/cover.gif")
        self.images=[PhotoImage(file="./images/%s.gif" %filename)
                     for filename in LANG]

```

```
        self.new_game()

        root.mainloop()

    def fill(self):
        for line in range(NB_LINES):
            for col in range(NB_COLS):
                center=(X0+col*SIDE, Y0+line*SIDE)
                k=self.board[line][col]
                self.cnv.create_image(center, image=self.images[k])
                self.ids_cover[line][col]=self.cnv.create_image(center,
                                                                image=self.cover)

    def hide(self, i, j, line,col):
        self.ids_cover[i][j]=self.cnv.create_image(X0+j*SIDE, Y0+i*SIDE,
                                                   image=self.cover)
        self.ids_cover[line][col]=self.cnv.create_image(X0+col*SIDE,
                                                        Y0+line*SIDE, image=self.cov
        self.move[0]=self.move[1]=None

    def handle_move(self, line, col):
        item=self.ids_cover[line][col]
        self.cnv.delete(item)
        if self.move[0] is None :
            self.move[0]=(line, col)
        else:
            if self.move[0]==(line, col):
                return
            self.cpt.set(self.cpt.get() + 1)
            self.move[1]=(line, col)
            i, j=self.move[0]
            if self.board[i][j]==self.board[line][col]!=-1:
                self.board[i][j]=self.board[line][col]=-1
                self.move[0]=self.move[1]=None
                self.couples=self.couples+1
                if self.couples==NB_PICT:
                    pygame.mixer.music.play(-1)
            else:
                self.cnv.after(400, self.hide, i, j, line, col)

    def clic(self, event):
        if self.move[1] is not None:
            return
        col, line=event.x//SIDE, event.y//SIDE
        if self.board[line][col]!=-1:
            self.handle_move(line, col)

    def new_game(self):
```

```
99          self.ids_cover=[[None for j in range(NB_COLS)] for i in range(NB_LINES)]
100         self.board=make_board(NB_LINES, NB_COLS)
101         self.fill()
102         self.move=[None, None]
103         self.couples=0
104         self.cpt.set(0)
105         pygame.mixer.music.stop()
106
107 Memory()
```

— Ligne 98 : la fonction init du fichier `memory_applaudir.py` est remplacée par une méthode `new_game` pour bien différencier de la méthode `__init__` de la classe `Memory`.

— Ligne 2 : l'importation `from tkinter import *` n'est pas recommandée, on préfère faire une importation sélective.

— Ligne 107 : le jeu est appelé en construisant une instance de la classe `Memory`. Noter que la mainloop (ligne 53) est située en fin de fonction `__init__` d'où le lancement du jeu.

— Ligne 28 : le constructeur est pauvre, il ne prend aucun argument.

— le tableau `board` (ligne 100), les images (lignes 48-49), le tableau des id (ligne 99) deviennent des attributs d'instance.

— La plupart des fonctions du code antérieur sont devenues des méthodes, avec souvent une signature plus simple. Toutefois, la fonction `make_board` (ligne 18) est restée une fonction car elle est indépendante de l'objet (`self`). Une méthode statique aurait toutefois être pu envisagée.

— Ligne 55 : cas typique où la méthode (ici `fill`) a une signature plus simple que la fonction. La raison est que la méthode accède aux attributs et qui étaient des paramètres de la fonction.

— Ligne 91 : le passage des arguments des fonctions de rappel ; on sait qu'une fonction de rappel prend un unique paramètre `event` ; or la signature de la méthode `clic` (ligne 91) en contient deux, à savoir `self` et `event`. En quoi n'est-ce pas incompatible ? Quand un clic de souris est capturé, la méthode `clic` est appelée automatiquement par Tkinter (noter que la fonction est déclarée sous la forme `self.clic` ligne 34) sous la forme `self.clic(event)`. Ensuite, c'est une spécificité de la POO en Python, l'interpréteur Python **ajoute comme premier argument** de l'appel, l'instance (`self`) si bien qu'il y a coïncidence du nombre d'arguments et du nombre de paramètres. Ce qui précède s'applique à la fonction de rappel (`new_game`, cf. ligne 98) donnée à `command` et qui ne prend aucun argument.

Améliorations à apporter

Ce jeu est susceptible d'améliorations ou de modifications, cf. des versions sur Internet ou sur le Play Store si vous êtes en panne d'inspiration. Voici quelques suggestions :

— placer un décompte du temps (chronomètre ou barre de progression en utilisant ttk)

— annoncer la victoire au joueur quand il a fini la partie, par exemple en ouvrant une fenêtre surgissante (popup) avec un message et un bouton pour refermer le popup

— laisser les cartes découvertes quelques secondes en début de jeu

— donner un résultat de performance de mémoire (compter les images non mémorisées, et celles mémorisées)

— implémenter une cheat key : si vous maintenez appuyée la touche Echap, toutes les cartes deviennent visibles, si vous relâchez, le jeu revient à son état antérieur
— autoriser le chevauchement de recherche (faire une nouvelle recherche alors que la paire de cartes différentes n'est pas encore faces cachées)
— faire des animations en entrée (et/ou sortie) de jeu.
— placer des niveaux (en fonction du nombre d'images et du temps de résolution)
— proposer des thèmes variés d'images (utiliser un widget OptionMenu)
— adapter la taille des images à la résolution de l'écran (sans doute assez difficile : nécessitera déjà un outil comme Pillow pour réaliser le redimensionnement des images)
— introduire des difficultés supplémentaires, par exemple, changer de place des images que le joueur a déjà retournées
— proposer au joueur de charger son propre pack d'images
— implémenter la possibilité de jouer uniquement au clavier
— choisir une autre disposition que la grille pour placer des images
— créer un bot
— jouer à deux et en réseau.