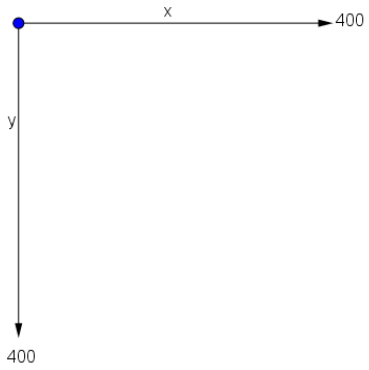
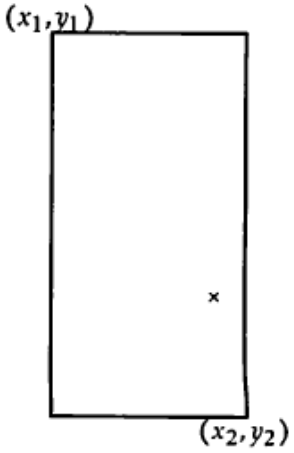
ISN 2016 ***Le damier magique***

Jalon 1

On considère un espace carré de 400 pixels de côté.



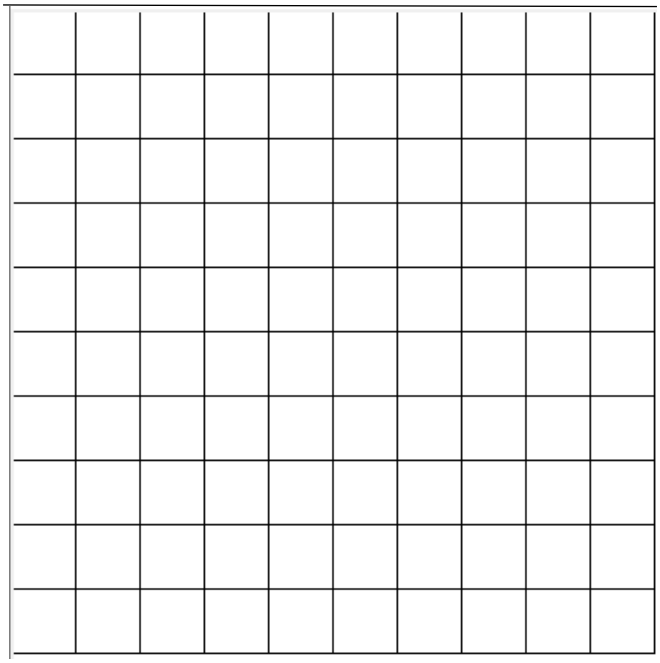
Dans cet espace, selon le repère ci–joint, chaque pixel est repéré par ses coordonnées qui forment un couple d'entiers naturels, chacun dans $[\![0; 400]\!]$.



Pour construire un rectangle, on utilise la méthode **rectangle(x_1, y_1, x_2, y_2).**

1°) Partie algorithmique : Ecrire un algorithme en pseudo–code permettant de construire le quadrillage de carrés ci-dessous à partir de la méthode rectangle. Le pixel le plus haut à gauche a pour coordonnées (0 ; 0).

2°) Partie programmation : programmer votre algorithme en Python en complétant le fichier P1.py



3°) Comparer votre solution avec celle proposée par le fichier P1_solution.py.

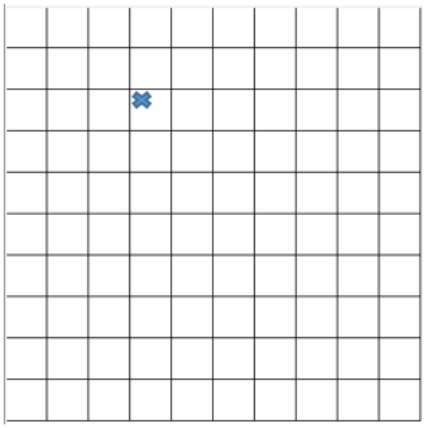
Jalon 2

On dispose maintenant d'un quadrillage de carrés de côtés 40 pixels dont le fond est blanc.

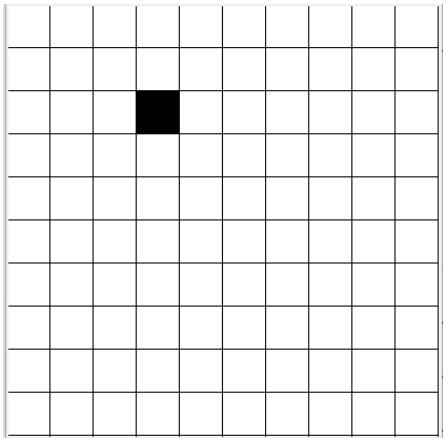
La méthode rectangle a un attribut possible, la couleur de remplissage : **rectangle(**$x_1, y_1, x_2, y_2,$ "$couleur$"**)** ***où couleur est black, red, white…***

Avec la souris, on clique sur un pixel de la zone quadrillée : une méthode permet de récupérer ses coordonnées a et b.

Par exemple, le clic à l'endroit de l'étoile bleu donne a =130 et b = 90.



1°) Connaissant a et b, écrire un algorithme, en pseudo–code, permettant de colorier tout le carré contenant « l'étoile bleu » correspondante :



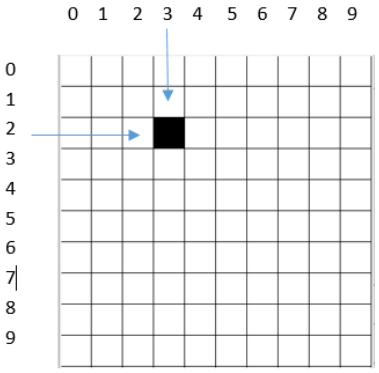
2°) Compléter l'algorithme P2.py en s'inspirant, si nécessaire, du programme souris.py (qui nécessite l'image cible.gif).

3°) Comparer votre solution avec celle fournie par le programme P2_solution.py

Jalon 3

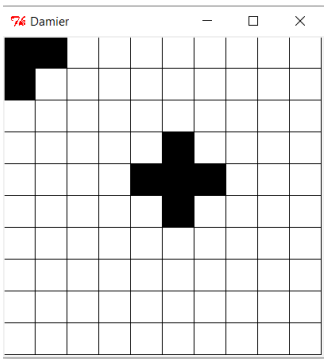
On dispose désormais d'un quadrillage par des carrés de côté 40 et on sait colorier chacun de ses carrés.

Chaque carré du quadrillage peut être repéré par un couple d'entiers, chacun dans $[\![0; 9]\!]$



Ainsi, la case noire peut être repérée par le couple (3 ;2) ou bien le couple (2 ;3) selon que l'on commence par les colonnes ou par les lignes.

A ce quadrillage, on associe alors un tableau comportant 10 lignes et 10 colonnes (appelé « matrice ») permettant de noter, à l'aide du système de repérage précédent, la couleur de chacune des cases du quadrillage : 1 pour noir et 0 pour blanc.




```
[[ 1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  1.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

Ainsi, sur le damier ci–dessus, on associe à « la case (0 ;1) » le nombre 1 et à la case (0 ;3) le nombre 0.
Toutes les couleurs des cases de ce damier sont ainsi listées dans la matrice proposée.

1°) On introduit les méthodes suivantes :

- M=**zéros(10,10)** qui permet de créer une matrice de 10 lignes et 10 colonnes, remplies de 0.
- M[c][d] =e qui permet d'affecter le nombre e à la case repéré par le couple (c ;d).
  Attention, ici c et d sont des entiers de $[\![0; 9]\!]$.

Ecrire en pseudo–code un algorithme permettant de colorier d'une part :

- la case où on a cliqué (repérée par son code « matriciel » qui nous est donné)
- les cases voisines verticalement ou horizontalement : par exemple, sur le damier ci–dessus, on a cliqué sur la case (5 ;4) ce qui a entraîné le coloriage des cases (5 ;3) ; (5 ;4) ;(5 ;5) ;(4 ;4) ;(6 ;4).
  On a cliqué ensuite sur la case (0 ;0), ce qui a entraîne le coloriage des cases (1 ;0) et (0 ;1).
  (Chaque case **soumise à un coloriage** change de couleur, **du noir au blanc** ou **du blanc au noir**).
- De récupérer le nombre total de cases noires.

2°) Compléter le programme P3.py, la bibliothèque numpy permettant la création et l'utilisation de matrices (voir lien numpy du site).

3°) Comparer votre programme avec le programme P3_solution.py.

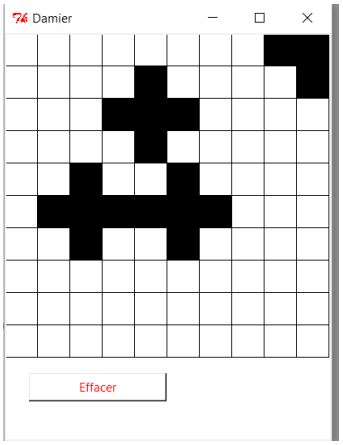
Jalon 4

1°) Compléter le programme P3_solution.py en ajoutant un widget « effacer » qui permet de relancer le jeu.

Quelques instructions qu'on pourra utiliser :

```
def effacer():
   Fond.delete("all")

B=Button(F,text="Effacer",fg="red", width=20, bg="white", command=effacer)
B.place(x=30,y=420)
```



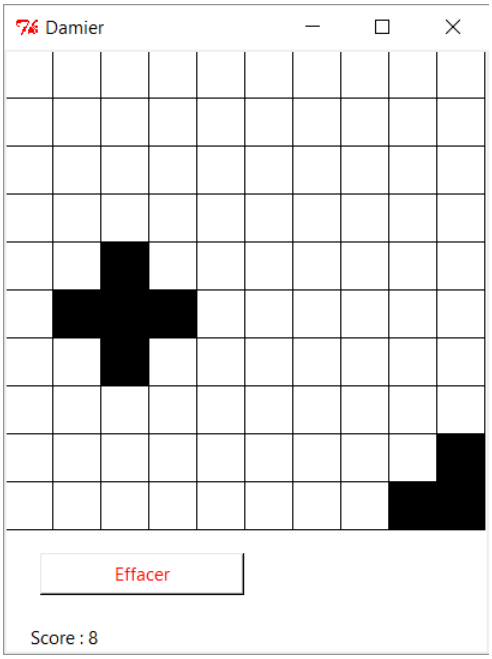
2°) Comparer votre solution avec celle proposée par le programme P4_solution.py

Jalon 5

1°) Compléter le programme P4_solution en y ajoutant l'indication du nombre de cases noires au fur et à mesure de l'avancée du jeu.

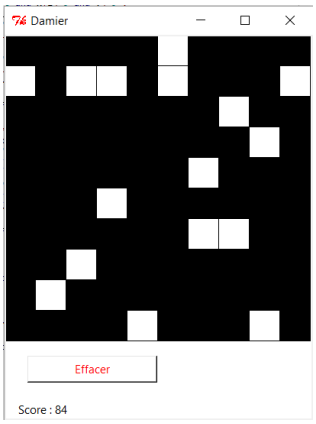
Quelques instructions qu'on pourra utiliser :

```
scorex=0
score = Fond.create_text(50,480,text="Score : "+str(scorex),anchor='n')
Fond.itemconfig(score,text = "Score : "+str(int(scorex)))
```



2°) Comparer votre solution avec celle proposée par le programme P5_solution.

3°) Battre ce record ? (84)



Jalons 6, 7, 8, 9

6 : Refaire ce jeu en demandant à l'utilisateur le nombre de lignes et de colonnes du « damier ».
7 : Construire un fichier de records…
8 : Proposer une version diffusable de votre jeu (un fichier executable)
9 : Diffuser votre jeu par le biais d'une page Internet.