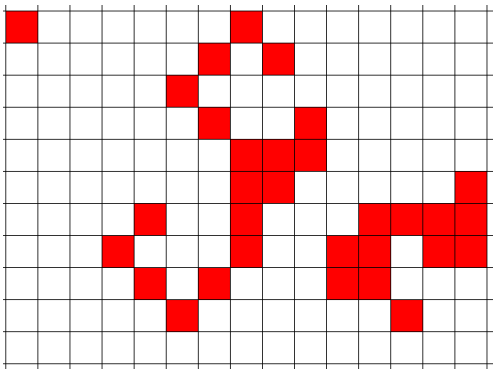
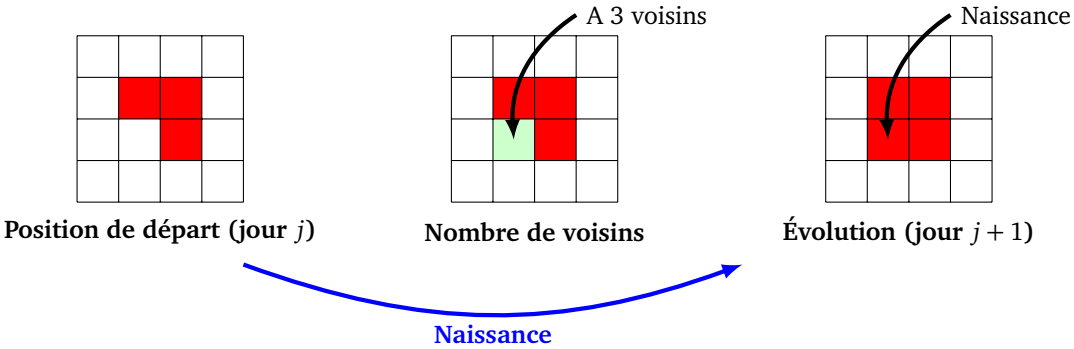
Jeu de la vie

Le jeu de la vie *est un modèle simple de l'évolution d'une population de cellules qui naissent et meurent au cours du temps. Le « jeu » consiste à trouver des configurations initiales qui donnent des évolutions intéressantes : certains groupes de cellules disparaissent, d'autres se stabilisent, certains se déplacent...*

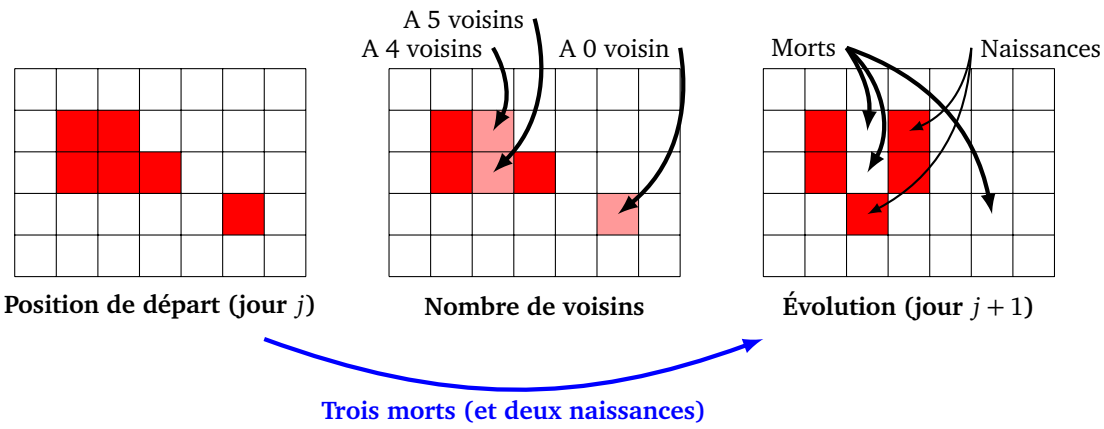


Cours 1 (Règles du jeu de la vie)**.**
Le *jeu de la vie* se déroule sur une grille. Chaque case peut contenir une cellule. Partant d'une configuration initiale, chaque jour des cellules vont naître et d'autres mourir en fonction du nombre de ses voisins. Voici les règles :

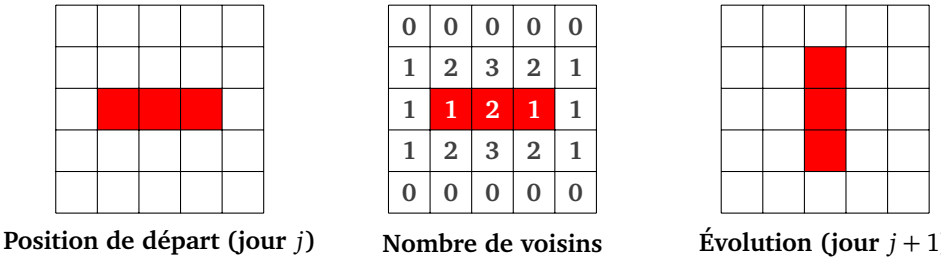
- Pour une case vide au jour j et ayant exactement 3 cellules voisines : une cellule naît au jour $j+1$.



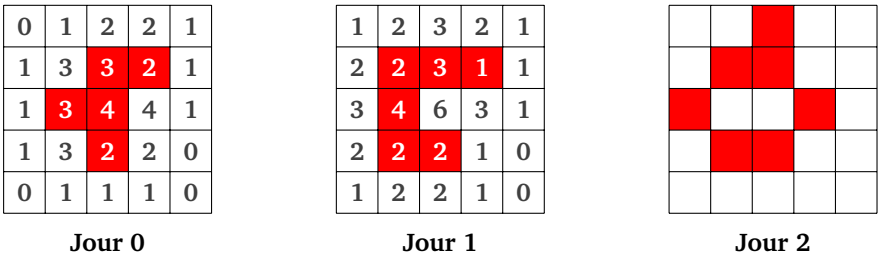
- Pour une case contenant une cellule au jour j, ayant soit 2 ou soit 3 cellules voisines : alors la cellule continue de vivre. Dans les autres cas la cellule meurt (avec 0 ou 1, elle meurt d'isolement, avec plus de 4 voisins, elle meurt de surpopulation !).



Voici un exemple simple, le « clignotant » :



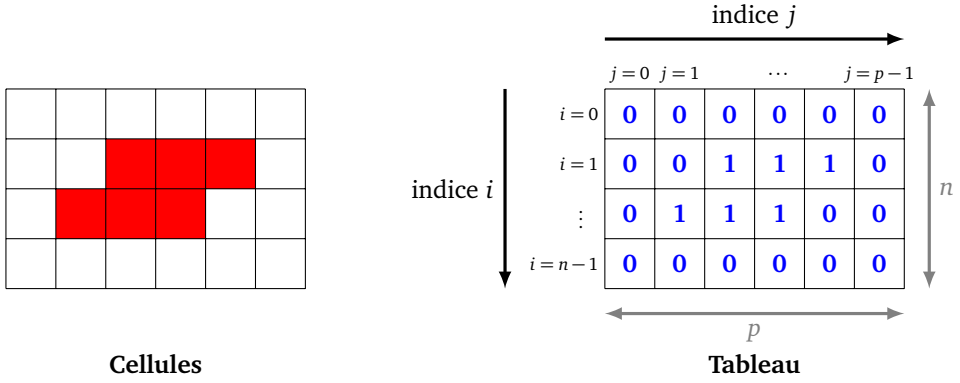
Voici un autre exemple (avec directement le nombre de voisins), la première évolution et l'évolution de l'évolution.



Activité 1 (Tableaux à deux dimensions)**.**

Objectifs : définir et afficher des tableaux indexés par deux indices.

Nous modélisons l'espace de vie des cellules par un tableau à double entrée, contenant des entiers, 1 pour signifier la présence d'une cellule, 0 sinon. Voici en exemple la configuration « bouche » et son tableau :



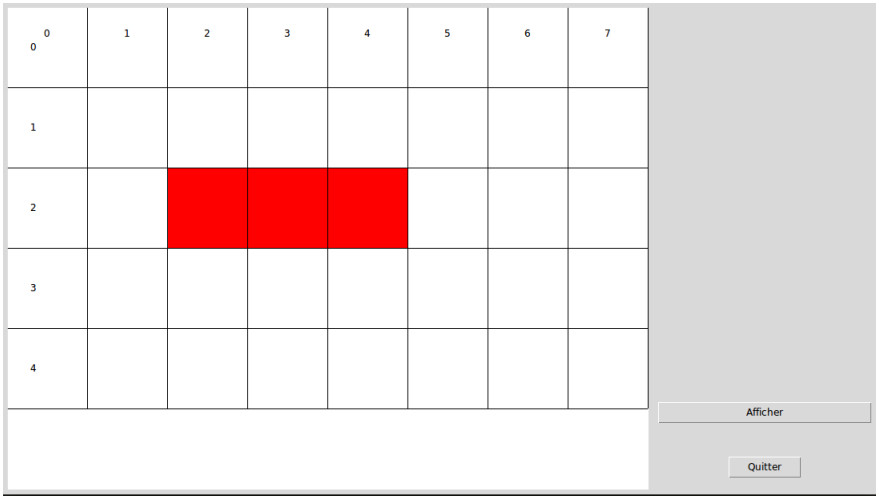
1. - Initialise deux variables `n` (la hauteur du tableau) et `p` (la largeur) (par exemple à 5 et 8).
   - Définis un tableau à deux dimensions rempli de zéros par la commande :
     ```
     tableau = [[0 for j in range(p)] for i in range(n)]
     ```
   - Par des instructions du type `tableau[i][j] = 1` remplis le tableau afin de définir la configuration du clignotant, de la bouche…
2. Programme l'affichage à l'écran d'un tableau donné. Par exemple, le clignotant s'affiche ainsi :

   ```
   00000000
   00000000
   00111000
   00000000
   00000000
   ```

   Indication : par défaut la commande `print()` passe à la ligne suivante à chaque appel (il rajoute le caractère `"\n"` qui est le caractère de fin de ligne). On peut lui spécifier de ne pas le faire par l'option `print("Mon texte",end="")`.

Activité 2 (Affichage graphique)**.**

Objectifs : réaliser l'affichage graphique d'une configuration de cellules.



1. Programme la création d'une fenêtre `tkinter` (voir la fiche « Statistique – Visualisation de données »). Écris une fonction `afficher_lignes()`, sans paramètre, qui affiche une grille vierge de coordonnées. Tu auras besoin d'une constante `echelle` (par exemple `echelle = 50`) pour transformer les indices i et j en des coordonnées graphiques x et y, suivant la relation : $x = e \times j$ et $y = e \times i$ (e étant l'échelle).

   Bonus. Tu peux aussi rajouter la valeur des indices i et j en haut et à gauche pour faciliter la lecture de la grille.
2. Construis une fonction `afficher_tableau(tab)` qui affiche graphiquement les cellules d'un tableau.

   Bonus. Rajoute un bouton « Afficher » et un bouton « Quitter » (voir la fiche « Statistique – Visualisation de données »).

Activité 3 (Évolution).

Objectifs : calculer l'évolution d'une configuration au jour d'après.

1. Programme une fonction `nombre_voisins(i,j,tab)` qui calcule le nombre de cellules voisines vivantes de la cellule (i, j).

   Indications.
   - Il y a en tout au maximum 8 voisins possibles. Le plus simple est de les tester un par un !
   - Pour le comptage, il faut faire bien attention aux cellules qui sont placées près d'un bord (et ont moins de 8 voisins).

2. Programme une fonction `evolution(tab)` qui en entrée reçoit un tableau et qui en sortie renvoie un nouveau tableau correspondant à la situation au jour d'après, selon les règles du jeu de la vie décrites au début. Par exemple, si le tableau de gauche correspond à l'entrée, alors la sortie correspond au tableau de droite :

00000000		00000000
00000000		00010000
00111000	évolue en	00010000
00000000		00010000
00000000		00000000

   Indications. Pour définir un nouveau tableau reprend la commande :

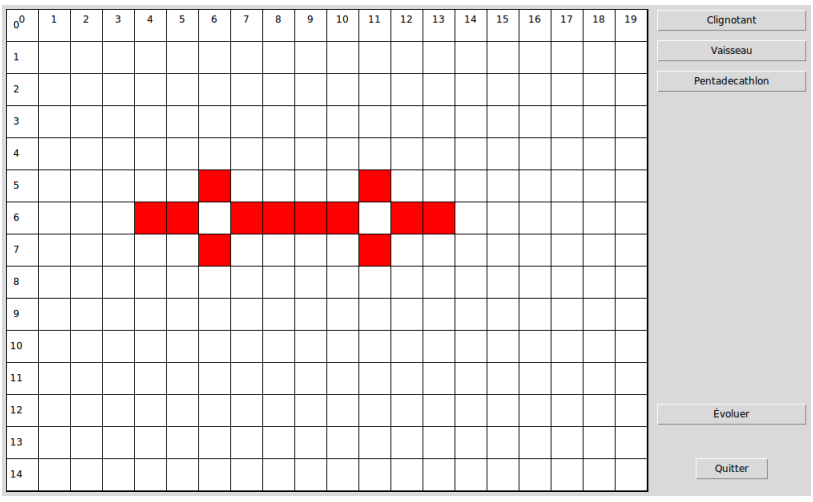
   ```
   nouveau_tableau = [[0 for j in range(p)] for i in range(n)]
   ```

   puis modifie le tableau comme voulu.

Activité 4 (Itérations).

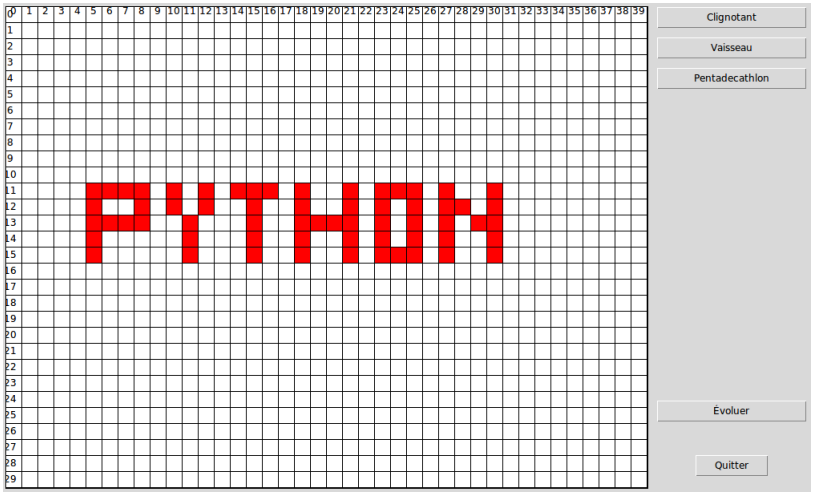
Objectifs : finir le programme graphique afin que l'utilisateur puisse définir des configurations et les faire évoluer d'un simple clic.

1. Améliore la fenêtre graphique afin de faciliter la vie de l'utilisateur :
   - Un bouton « évoluer » qui à chaque clic affiche l'évolution.
   - Des boutons permettant d'afficher des configurations pré-définies (sur la capture d'écran ci-dessous c'est la configuration « pentadecathlon »).



2. Perfectionne ton programme afin que l'utilisateur dessine la configuration qu'il souhaite par des clics de la souris. Un clic sur une case éteinte l'allume, un clic sur une case allumée l'éteint. Tu peux décomposer ce travail en trois fonctions :

- `allumer_eteindre(i,j)`, qui commute la cellule (i, j) ;
- `xy_vers_ij(x,y)` qui convertit des coordonnées graphiques (x, y) en des coordonnées entières (i, j) (utiliser la variable `echelle` et la division entière).
- `action_clic_souris(event)` pour récupérer les coordonnées (x, y) d'un clic de souris (voir le cours ci-dessous) et commuter la case cliquée.



Cours 2 (Clic de souris)**.**
Voici un petit programme qui affiche une fenêtre graphique. Chaque fois que l'utilisateur clique (avec le bouton gauche de la souris) le programme affiche un petit carré (sur la fenêtre) et affiche « Clic à $x = \ldots$, $y = \ldots$ » (sur la console).

```
from tkinter import *

# Fenêtre
root = Tk()
canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# Capture des clics de souris
def action_clic_souris(event):
    canvas.focus_set()
    x = event.x
    y = event.y
    canvas.create_rectangle(x,y,x+10,y+10,fill="red")
    print("Clic à x =",x,", y =",y)
    return

# Association clic/action
canvas.bind("<Button-1>", action_clic_souris)

# Lancement
root.mainloop()
```

Voici quelques explications :

- La création de la fenêtre est habituelle. Le programme se termine par le lancement avec la commande

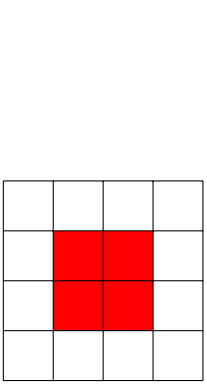
`mainloop()`.

- Le premier point clé est d'associer un clic de souris à une action, c'est ce que fait la ligne
  `canvas.bind("<Button-1>", action_clic_souris)`
  Chaque fois que le bouton gauche de la souris est cliqué, `Python` exécute la fonction `action_clic_souris`. (Note qu'il n'y a pas de parenthèses pour l'appel à la fonction.)
- Second point clé : la fonction `action_clic_souris` récupère les coordonnées du clic et ici ensuite fait deux choses : elle affiche un petit rectangle à l'endroit du clic et affiche dans la fenêtre du terminal les coordonnées (x, y).
- Les coordonnées x et y sont exprimées en pixels ; $(0, 0)$ désigne le coin en haut à gauche de la fenêtre (la zone délimitée par `canvas`).

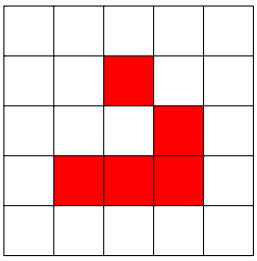
Voici quelques idées pour aller plus loin :

- Fais en sorte que la grille s'adapte automatiquement à la largeur de l'écran, c'est-à-dire calcule `echelle` en fonction de `n` et `p`.
- Rajoute une étape intermédiaire avant d'évoluer : colorie en vert une cellule qui va naître et en noir une cellule qui va mourir. Pour cela les éléments de `tableau` pourront prendre des valeurs autres que 0 et 1.

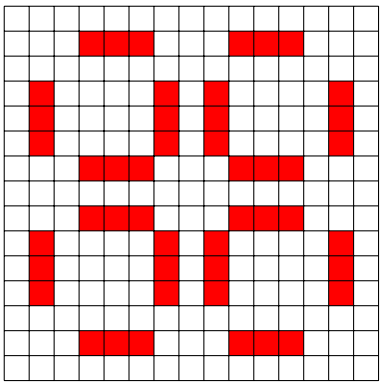
Voici quelques configurations intéressantes.



Carré



Vaisseau



Pulsar

Tu en trouveras plein d'autres sur internet mais surtout amuse-toi à en découvrir de nouvelles !
En particulier, trouve des configurations :

- qui restent fixes au cours du temps ;
- qui évoluent, puis deviennent fixes ;
- qui sont périodiques (les mêmes configurations reviennent en boucles) avec une période de 2, 3 ou plus ;
- qui voyagent ;
- qui propulsent des cellules ;
- dont la population augmente indéfiniment !