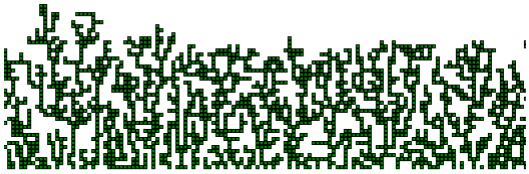
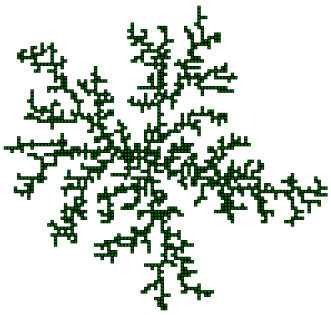
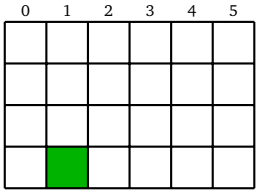
Constructions aléatoires

Tu vas programmer deux méthodes pour construire des figures qui ressemblent à des algues ou des coraux. Chaque figure est formée de petits blocs lancés au hasard et qui se collent les uns aux autres.

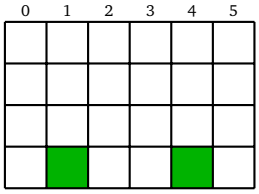




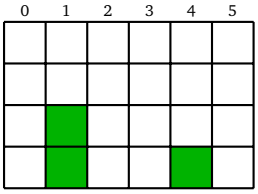
Cours 1 (Chutes de blocs)**.**
On fait tomber des blocs carrés dans une grille, sur le principe du jeu « Puissance 4 » : après avoir choisi une colonne, un bloc tombe du haut vers le bas. Les blocs se posent sur le bas de la grille ou sur des autres blocs, mais il y a une grosse différence avec le jeu « Puissance 4 », ici les blocs sont « collants », c'est-à-dire qu'un bloc reste collé dès qu'il rencontre un voisin à gauche ou à droite.
Voici un exemple de lancer de blocs :



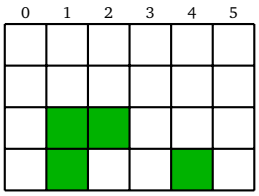
1. Bloc lancé en col. 1



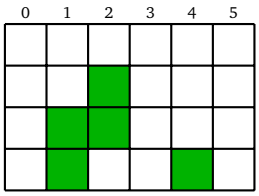
2. Bloc lancé en col. 4



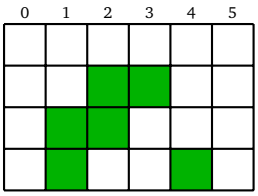
3. Bloc lancé en col. 1



4. Bloc lancé en col. 2

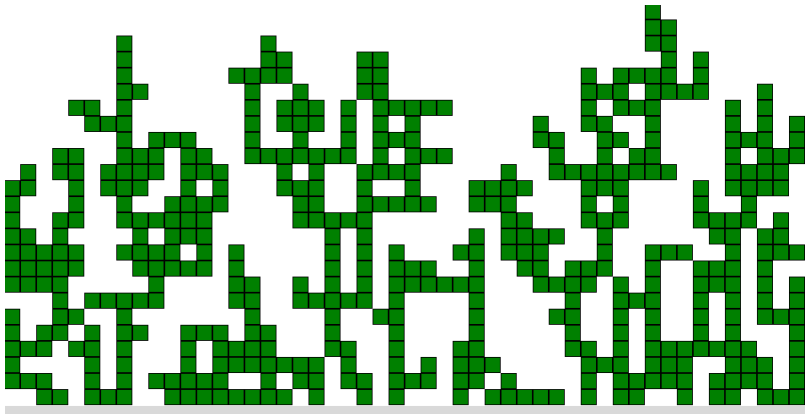


5. Bloc lancé en col. 2



6. Bloc lancé en col. 3

Par exemple à l'étape 4, le bloc lancé dans la colonne numéro 2 ne descend pas jusqu'en bas mais reste « scotché » à son voisin, il se retrouve donc définitivement suspendu.
Le lancer aléatoire de centaines de blocs sur une grande grille produit de jolies formes géométriques ressemblant à des algues.



Activité 1 (Chutes de blocs)**.**

Objectifs : programmer la chute des blocs (sans affichage graphique).

On modélise l'espace de travail par un tableau de n lignes et p colonnes. Au départ le tableau ne contient que des 0 ; ensuite la présence d'un bloc est représentée par 1.
Voici comment initialiser le tableau :

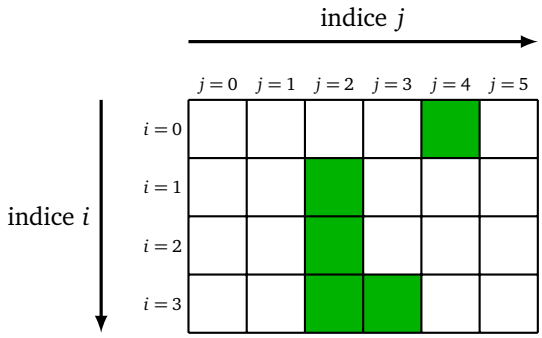
```
tableau = [[0 for j in range(p)] for i in range(n)]
```

On modifie le tableau par des instructions du type :

```
tableau[i][j] = 1
```

Voici un exemple de tableau (à gauche) pour représenter la situation graphique de droite (le bloc en haut à droite est en train de tomber).

0	0	0	0	1	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	1	0	0



Un tableau avec 5 blocs ($n=4$, $p=6$)

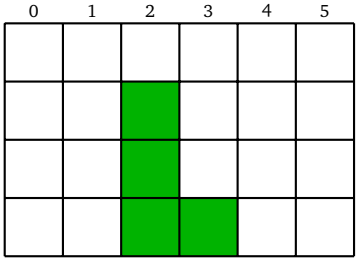
1. Programme une fonction `peut_tomber(i,j)` qui détermine si le bloc en position (i,j) peut descendre d'une case ou pas.

   Voici les cas dans lesquels le bloc *ne peut pas* tomber :
   - si le bloc est déjà sur la dernière ligne,
   - s'il y a un bloc juste en dessous,
   - s'il y a un bloc juste à droite ou juste à gauche.

2. Programme une fonction `faire_tomber_un_bloc(j)` qui fait tomber un bloc dans la colonne j jusqu'à ce qu'il ne puisse plus descendre. Cette fonction modifie les entrées du tableau.

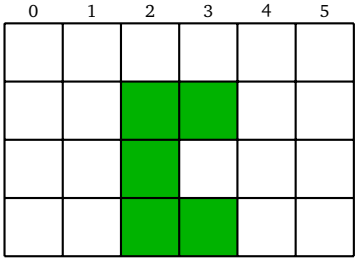
   Par exemple, voici le tableau avant (à gauche) et après (à droite) avoir fait tomber un bloc dans la colonne $j=3$.

0	0	0	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	1	0	0



Avant

0	0	0	0	0	0
0	0	1	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0

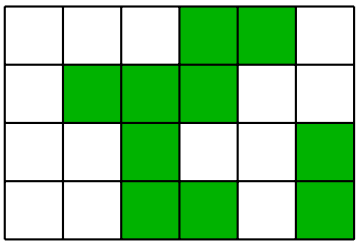


Après

3. Programme une fonction `faire_tomber_des_blocs(k)` qui lance k blocs un par un, en choisissant à chaque fois une colonne au hasard (c'est-à-dire un entier j avec $0 \leqslant j < p$).

   Voici un exemple de tableau obtenu après avoir lancé 10 blocs :

0	0	0	1	1	0
0	1	1	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1

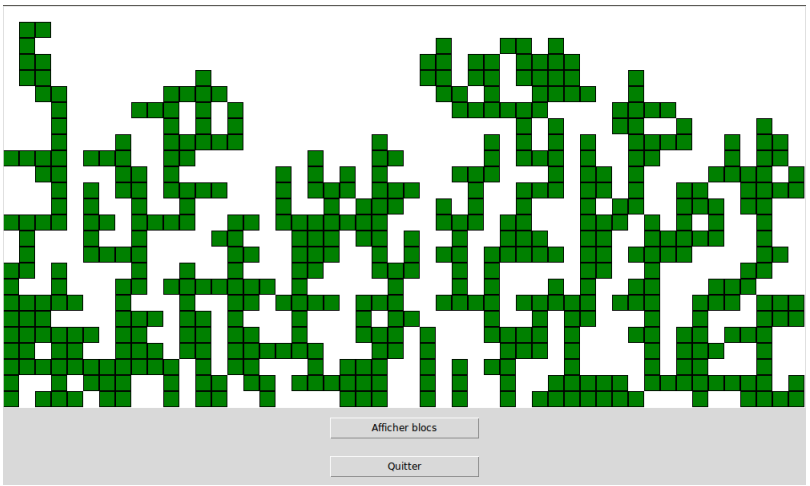


Lancer de 10 blocs

Activité 2 (Chutes de blocs (suite))**.**

Objectifs : programmer l'affichage graphique des blocs.

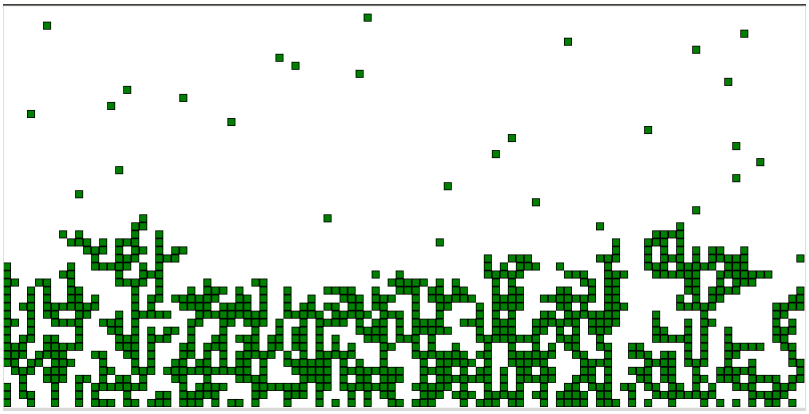
Affichage statique. Programme l'affichage graphique des blocs à partir d'un tableau.



Indications.

- Utilise le module `tkinter`, voir la fiche « Statistique – Visualisation de données ».
- Tu peux rajouter un bouton qui lance un bloc (ou plusieurs d'un coup).

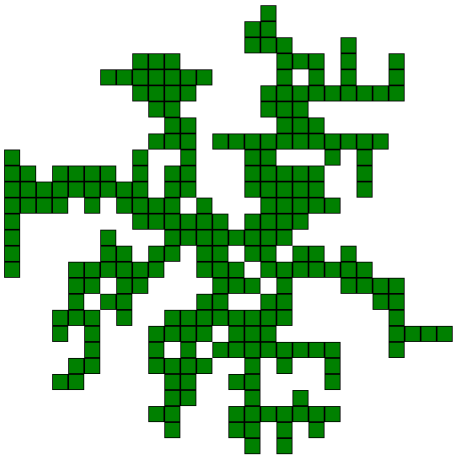
Affichage dynamique (facultatif et difficile). Programme l'affichage des blocs qui tombent.



Indications.

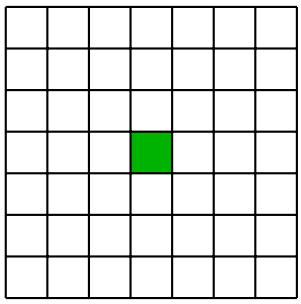
- C'est beaucoup plus compliqué à programmer, mais très joli à voir !
- Pour le déplacement des blocs, inspire-toi du programme « Mouvement avec `tkinter` » à la fin de cette fiche.
- Pour faire une « pluie de blocs » de façon régulière (par exemple tous les dixièmes de secondes) : on fait descendre tous les blocs existant d'une case et on en fait apparaître un nouveau sur la ligne du haut.

Cours 2 (Arbres browniens)**.**
Voici une construction un peu différente, beaucoup plus longue à calculer, mais qui dessine aussi de jolies figures appelées « arbres browniens ».

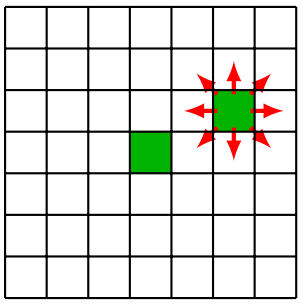


Le principe est le suivant :

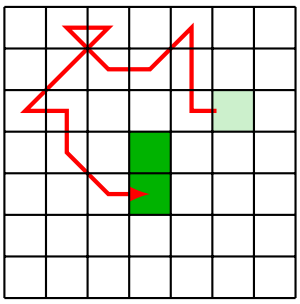
- On part d'une grille (il faut cette fois imaginer qu'elle est dessinée à plat sur une table). En son centre, on place un premier bloc fixe, le *germe*.
- On fait apparaître un bloc au hasard sur la grille. À chaque étape, ce bloc se déplace au hasard sur l'une des huit cases adjacentes, on parle d'un *mouvement brownien*.
- Dès que ce bloc touche un autre bloc par un côté, il s'y colle et ne bouge plus.
- Si ce bloc sort de la grille, il se désintègre.
- Une fois le bloc collé ou désintégré, on relance alors un nouveau bloc depuis un point aléatoire de la grille.



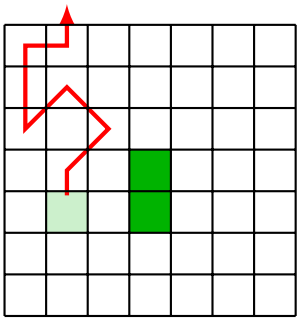
Le germe



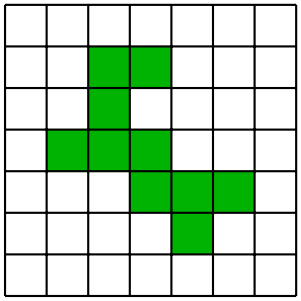
Un bloc et ses 8 mouvements possibles



Le mouvement aléatoire du bloc



Un bloc qui quitte la grille



10 blocs

On obtient petit à petit une sorte d'arbre qui ressemble à du corail. Les calculs sont très longs car beaucoup de blocs sortent de la grille ou mettent longtemps avant de se fixer (surtout au début). En plus, on ne peut lancer les blocs qu'un par un.

Activité 3 (Arbres browniens)**.**

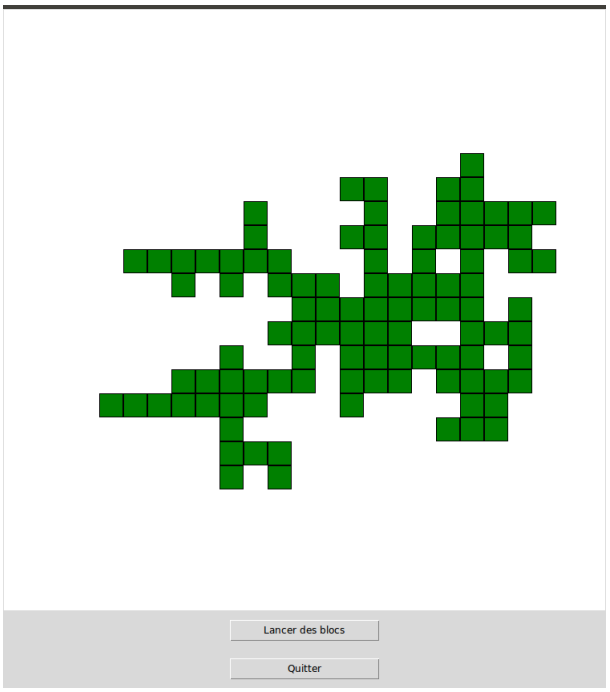
Objectifs : programmer la création d'un arbre brownien.

Première partie.

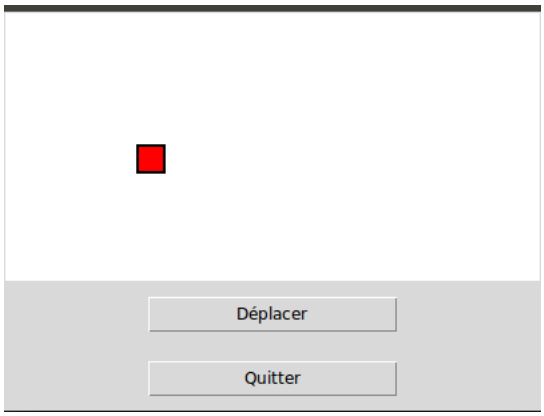
1. Modélise de nouveau l'espace de travail par un tableau de n lignes et p colonnes contenant des 0 ou des 1. Initialise toutes les valeurs à 0 sauf 1 au centre du tableau.
2. Programme une fonction `est_dedans(i,j)` qui détermine si la position (i, j) est bien dans la grille (sinon c'est que le bloc est en train de sortir).
3. Programme une fonction `est_libre(i,j)` qui détermine si le bloc en position (i, j) peut bouger (la fonction renvoie « vrai ») ou s'il est collé (la fonction renvoie « faux »).
4. Programme une fonction `lancer_un_bloc()`, sans paramètre, qui simule la création d'un bloc et son déplacement aléatoire, jusqu'à ce qu'il se colle ou qu'il quitte la grille.

   Indications.
   - Le bloc est créé à une position aléatoire (i, j) de la grille.
   - Tant que le bloc est dans la grille et libre de bouger :
     — tu choisis un déplacement horizontal en tirant au hasard un entier parmi $\{-1, 0, +1\}$, idem pour un déplacement vertical ;
     — tu déplaces le bloc selon la combinaison de ces deux mouvements.
   - Modifie alors le tableau.
5. Termine avec une fonction `lancer_des_blocs(k)` qui lance k blocs.

Seconde partie.
Programme l'affichage graphique à l'aide de `tkinter`. Tu peux ajouter un bouton qui lance 10 blocs d'un coup.



Cours 3 (Mouvement avec « tkinter »)**.**
Voici un programme qui fait se déplacer un petit carré et le faisant rebondir sur les bords de la fenêtre.



Voici les points principaux :

- Un objet `rect` est défini, c'est une variable globale, de même que ses coordonnées `x0`, `y0`.
- Cet objet est (un petit peu) déplacé par la fonction `deplacer()` qui décale le rectangle de `(dx,dy)`.
- Le point clé est que cette fonction sera exécutée une nouvelle fois après un court laps de temps. La commande :

  `canvas.after(50,deplacer)`

  demande une nouvelle exécution de la fonction `deplacer()` après un court délai (ici 50 millisecondes).
- La répétition de petits déplacements simule le mouvement.

```
from tkinter import *
```

```
Largeur = 400
Hauteur = 200

root = Tk()
canvas = Canvas(root, width=Largeur, height=Hauteur, background="white")
canvas.pack(fill="both", expand=True)

# Les coordonnées et la vitesse
x0, y0 = 100,100
dx = +5  # Vitesse horizontale
dy = +2  # Vitesse verticale

# Le rectangle à déplacer
rect = canvas.create_rectangle(x0,y0,x0+20,y0+20,width=2,fill="red")

# Fonction principale
def deplacer():
    global x0, y0, dx, dy

    x0 = x0 + dx  # Nouvelle abscisse
    y0 = y0 + dy  # Nouvelle ordonnée

    canvas.coords(rect,x0,y0,x0+20,y0+20)    # Déplacement

    if x0 < 0 or x0 > Largeur:
        dx = -dx  # Changement de sens horizontal
    if y0 < 0 or y0 > Hauteur:
        dy = -dy  # Changement de sens vertical

    canvas.after(50,deplacer)  # Appel après 50 millisecondes

    return

# Fonction pour le bouton
def action_deplacer():
    deplacer()
    return

# Boutons
bouton_couleur = Button(root,text="Déplacer", width=20, command=action_deplacer)
bouton_couleur.pack(pady=10)

bouton_quitter = Button(root,text="Quitter", width=20, command=root.quit)
bouton_quitter.pack(side=BOTTOM, pady=10)

root.mainloop()
```