TP10.2– JEU DE LA VIE

EXPLICATION WIKIPEDIA TRES BIEN FAITE : HTTPS://FR.WIKIPEDIA.ORG/WIKI/JEU_DE_LA_VIE

OBJECTIFS

- Comprendre les règles du Jeu de la Vie de Conway.
- Manipuler des grilles et apprendre à coder une simulation discrète.
- Concevoir des visualisations simples.
- Étudier la complexité temporelle d'une itération.

---

1. Présentation du problème

Le Jeu de la Vie est un automate cellulaire à deux dimensions. Une cellule peut être **vivante** ou **morte** et évolue selon des règles locales en fonction de ses voisins.

Chaque cellule possède **8 voisins** (adjacence de Moore).

Pour une cellule vivante :

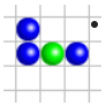
- Elle **survit** si elle a **2 ou 3 voisins vivants**.

- Elle **meurt** (sous-population ou surpopulation) sinon.

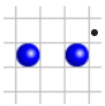
Pour une cellule morte :

- Elle **naît** si elle a **exactement 3 voisins vivants**.

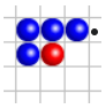
On peut également formuler cette évolution ainsi :



- Si une cellule a exactement trois voisines vivantes, elle est vivante à l'étape suivante.
  C'est le cas de la cellule verte dans la configuration de gauche ;



- Si une cellule a exactement deux voisines vivantes, elle reste dans son état actuel à l'étape suivante.
  Dans le cas de la configuration de gauche, la cellule située entre les deux cellules vivantes reste morte à l'étape suivante ;



- si une cellule a strictement moins de deux ou strictement plus de trois voisines vivantes, elle est morte à l'étape suivante.
  C'est le cas de la cellule rouge dans la configuration de gauche.

---

2. Premières étapes

2.1. Représentation de la grille

1. Représenter la grille comme une matrice de 0 (mort) et 1 (vivant).
2. Écrire une fonction `afficher(grille)` affichant la grille de façon lisible.

2.2. Comptage des voisins

Écrire une fonction :

```
def nb_voisins_vivants(grille, i, j):
    ...
```

qui renvoie le nombre de voisins vivants de la cellule `(i, j)`.

2.3. Calcul de la génération suivante

Écrire une fonction :

```
def prochaine_generation(grille):
    ...
```

qui renvoie une nouvelle grille correspondant à l'évolution de toutes les cellules.

2.4. Simulation

Écrire une fonction :

```
def simulation(grille, n):
    ...
```

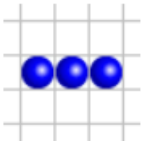
qui affiche successivement n générations.

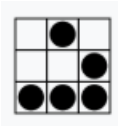
3. Tests

Tester avec :

- le **clignotant (blinker)**,

- le **planeur (glider)**.





Analyse algorithmique

1. Donner la complexité temporelle d'un calcul de génération.
2. Proposer une amélioration possible.

Correction

1. Comptage des voisins

```python
def nb_voisins_vivants(grille, i, j):
    n, m = len(grille), len(grille[0])
    voisins = [(-1,-1), (-1,0), (-1,1),
               (0,-1),          (0,1),
               (1,-1),  (1,0),  (1,1)]
    c = 0
    for di, dj in voisins:
        ni, nj = i + di, j + dj
        if 0 <= ni < n and 0 <= nj < m:
            c += grille[ni][nj]
    return c
```

2. Génération suivante

```python
def prochaine_generation(grille):
    n, m = len(grille), len(grille[0])
    nouvelle = [[0]*m for _ in range(n)]
    for i in range(n):
        for j in range(m):
            v = nb_voisins_vivants(grille, i, j)
            if grille[i][j] == 1:
                if v in (2, 3):
                    nouvelle[i][j] = 1
            else:
                if v == 3:
                    nouvelle[i][j] = 1
    return nouvelle
```

3. Simulation

```python
import time

def afficher(grille):
    for ligne in grille:
        print(''.join('' if x else ' ' for x in ligne))
    print()

def simulation(grille, n):
    for _ in range(n):
        afficher(grille)
        grille = prochaine_generation(grille)
        time.sleep(0.2)
```

4. Exemples

Clignotant

```
clignotant = [
    [0,0,0,0,0],
    [0,0,0,0,0],
    [0,1,1,1,0],
    [0,0,0,0,0],
    [0,0,0,0,0]
]
```

Planeur

```
glider = [
    [0,1,0],
    [0,0,1],
    [1,1,1]
]
```

5. Complexité

Pour une grille de taille `n × m` :

- Pour chaque cellule, on regarde 8 voisins : coût constant.

- Total par génération : **O(nm)**.

6. Simulation graphique avec Tkinter

Dans cette section, une visualisation graphique du Jeu de la Vie est réalisée en utilisant **Tkinter**, qui offre une interface graphique.

Objectifs

- Afficher une grille plus grande (ex : 50×50).
- Représenter chaque cellule par un rectangle rempli (vivante) ou vide (morte).
- Mettre à jour dynamiquement l'affichage à chaque génération.

Code proposé par ChatGPT !!

```
import tkinter as tk
import random

CELL_SIZE = 12
DELAY = 100  # en millisecondes

# Création de la fenêtre Tkinter
def init_tk(n, m):
    root = tk.Tk()
```

```
    root.title("Jeu de la Vie - Simulation Tkinter")
    canvas = tk.Canvas(root, width=m * CELL_SIZE, height=n * CELL_SIZE, bg="w
hite")
    canvas.pack()
    return root, canvas

# Dessine une cellule vivante ou morte
def dessine_grille(canvas, grille):
    canvas.delete("all")
    n, m = len(grille), len(grille[0])
    for i in range(n):
        for j in range(m):
            x1 = j * CELL_SIZE
            y1 = i * CELL_SIZE
            x2 = x1 + CELL_SIZE
            y2 = y1 + CELL_SIZE
            if grille[i][j] == 1:
                canvas.create_rectangle(x1, y1, x2, y2, fill="black", outline
="grey")
            else:
                canvas.create_rectangle(x1, y1, x2, y2, fill="white", outline
="grey")

# Lancement de la simulation Tkinter
def simulation_tk(grille):
    n, m = len(grille), len(grille[0])
    root, canvas = init_tk(n, m)

    def update():
        nonlocal grille
        dessine_grille(canvas, grille)
        grille = prochaine_generation(grille)
        root.after(DELAY, update)

    update()
    root.mainloop()
```

Extension : Interface interactive complète

Vous pouvez enrichir la simulation Tkinter avec une interface plus avancée comprenant :

- **Bouton Start / Pause**
- **Bouton Réinitialiser (grille aléatoire)**
- **Éditeur de grille au clic** : cliquer pour inverser l'état d'une cellule
- **Choix de la taille de la grille ou de la taille des cellules**

Code complet proposé orienté objet (version enrichie avec vitesse + motifs)

Ce code ajoute :

- un **slider pour régler la vitesse** (temps entre deux générations),

- un **menu déroulant** pour insérer automatiquement des motifs célèbres : - Glider - Pulsar - LWSS (Lightweight Spaceship)

```
import tkinter as tk
import random

CELL_SIZE = 12

# --- Motifs célèbres ---
MOTIFS = {
    "Glider": [
        [0,1,0],
        [0,0,1],
        [1,1,1]
    ],
    "Pulsar": [
        [0,0,1,1,1,0,0,0,1,1,1,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0],
        [1,0,0,0,0,1,0,1,0,0,0,0,1],
        [1,0,0,0,0,1,0,1,0,0,0,0,1],
        [1,0,0,0,0,1,0,1,0,0,0,0,1],
        [0,0,1,1,1,0,0,0,1,1,1,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,1,1,0,0,0,1,1,1,0,0],
        [1,0,0,0,0,1,0,1,0,0,0,0,1],
        [1,0,0,0,0,1,0,1,0,0,0,0,1],
        [1,0,0,0,0,1,0,1,0,0,0,0,1],
        [0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,1,1,1,0,0,0,1,1,1,0,0]
    ],
    "LWSS": [
        [0,1,1,1,1],
        [1,0,0,0,1],
        [0,0,0,0,1],
        [1,0,0,1,0]
    ]
}

class JeuDeLaVie:
    def __init__(self, n=50, m=50):
        self.n = n
        self.m = m
        self.grille = [[random.randint(0, 1) for _ in range(m)] for _ in rang
e(n)]
        self.running = False
        self.delay = 120

        # --- Fenêtre ---
```

```
        self.root = tk.Tk()
        self.root.title("Jeu de la Vie – Interface avancée")

        # --- Canvas ---
        self.canvas = tk.Canvas(self.root, width=m * CELL_SIZE, height=n * CE
LL_SIZE, bg="white")
        self.canvas.grid(row=0, column=0, columnspan=4)
        self.canvas.bind("<Button-1>", self.toggle_cell)

        # --- Boutons ---
        tk.Button(self.root, text="Start", command=self.start).grid(row=1, co
lumn=0)
        tk.Button(self.root, text="Pause", command=self.pause).grid(row=1, co
lumn=1)
        tk.Button(self.root, text="Réinitialiser", command=self.reset).grid(r
ow=1, column=2)

        # --- Slider vitesse ---
        self.speed_slider = tk.Scale(self.root, from_=20, to=500, orient=tk.H
ORIZONTAL,
                                     label="Vitesse (ms)", command=self.updat
e_speed)
        self.speed_slider.set(self.delay)
        self.speed_slider.grid(row=2, column=0, columnspan=2)

        # --- Menu motifs ---
        self.motif_var = tk.StringVar(self.root)
        self.motif_var.set("Insérer un motif")

        motif_menu = tk.OptionMenu(self.root, self.motif_var, *MOTIFS.keys(),
command=self.insert_motif)
        motif_menu.grid(row=2, column=2, columnspan=2)

        self.update_canvas()

    # --- Interaction souris ---
    def toggle_cell(self, event):
        i = event.y // CELL_SIZE
        j = event.x // CELL_SIZE
        self.grille[i][j] = 1 - self.grille[i][j]
        self.update_canvas()

    # --- Interaction vitesse ---
    def update_speed(self, value):
        self.delay = int(value)

    # --- Gestion motifs ---
    def insert_motif(self, motif_name):
        motif = MOTIFS[motif_name]
```

```
        mi, mj = len(motif), len(motif[0])
        offset_i = (self.n - mi) // 2
        offset_j = (self.m - mj) // 2
        for i in range(mi):
            for j in range(mj):
                self.grille[offset_i + i][offset_j + j] = motif[i][j]
        self.update_canvas()

    # --- Gestion simulation ---
    def start(self):
        if not self.running:
            self.running = True
            self.run()

    def pause(self):
        self.running = False

    def reset(self):
        self.grille = [[random.randint(0, 1) for _ in range(self.m)] for _ in
range(self.n)]
        self.update_canvas()

    # --- Affichage ---
    def update_canvas(self):
        self.canvas.delete("all")
        for i in range(self.n):
            for j in range(self.m):
                x1, y1 = j * CELL_SIZE, i * CELL_SIZE
                x2, y2 = x1 + CELL_SIZE, y1 + CELL_SIZE
                color = "black" if self.grille[i][j] == 1 else "white"
                self.canvas.create_rectangle(x1, y1, x2, y2, fill=color, outl
ine="grey")

    # --- Automate ---
    def nb_voisins(self, i, j):
        voisins = [(-1,-1), (-1,0), (-1,1), (0,-1), (0,1), (1,-1), (1,0), (1,
1)]
        return sum(self.grille[i+di][j+dj]
                   for di, dj in voisins
                   if 0 <= i+di < self.n and 0 <= j+dj < self.m)

    def next_gen(self):
        new = [[0]*self.m for _ in range(self.n)]
        for i in range(self.n):
            for j in range(self.m):
                v = self.nb_voisins(i, j)
                if self.grille[i][j] == 1:
                    new[i][j] = 1 if v in (2,3) else 0
                else:
                    new[i][j] = 1 if v == 3 else 0
```

```
        self.grille = new

    # --- Boucle principale ---
    def run(self):
        if self.running:
            self.next_gen()
            self.update_canvas()
            self.root.after(self.delay, self.run)

    def launch(self):
        self.root.mainloop()

# Exemple d'utilisation :
# app = JeuDeLaVie(50, 50)
# app.launch()
```

7. Améliorations possibles

- Changer la méthode d'initialisation en mettant la grille blanche
- Essayer de faire un canon à glider !
- Ne mettre à jour que les cellules vivantes et leurs voisines.
- Utiliser des structures de type `set` pour un univers infini ...