```
"""
Cap 9
"""

from Cell2D import Cell2D, Cell2DViewer
from scipy.signal import correlate2d
import numpy as np
import matplotlib.pyplot as plt
class Percolation(Cell2D):
    """AC di percolazione."""
    kernel = np.array([[0, 1, 0],
                       [1, 0, 1],
                       [0, 1, 0]])
    def __init__(self, n, q=0.5, seed=None):
        """Inizializza gli attributi.
        n: numero di righe
        q: probabilità di porosità
        seed: seme casuale
        """
        self.q = q
        if seed is not None:
            np.random.seed(seed)
        self.array = np.random.choice([1, 0], (n, n), p=[q, 1-q])

        # riempe la prima riga di celle bagnate
        self.array[0] = 5
    def step(self):
        """Esegue un passo nel tempo."""
        a = self.array
        c = correlate2d(a, self.kernel, mode='same')
        self.array[(a==1) & (c>=5)] = 5

    def num_wet(self):
        """Numero totale di celle bagnate."""
        return np.sum(self.array == 5)

    def bottom_row_wet(self):
        """Numero di celle bagnate nella prima riga."""
        return np.sum(self.array[-1] == 5)

class PercolationViewer(Cell2DViewer):
    """Disegna e anima l'oggetto della percolazione."""
    cmap = plt.get_cmap('Blues')
    options = dict(alpha=0.6,
                   interpolation='nearest',
                   vmin=0, vmax=5)
n=int(input('Numero  di passi  '))
iters=int(input('N.passi di calcolo [tip=200] ; '))
seed = 18
q=0.6
perc = Percolation(n, q, seed)
def test_perc(perc):
    """Esegue il modello di percolazione.
    Gira finché il fluido non giunge nell'ultima riga o non vi è alcuna
    variazione.
    returns: booleano se c'è un gruppo di percolazione"""
    num_wet = perc.num_wet()
    while True:
        perc.step()
        if perc.bottom_row_wet():
           return True
        new_num_wet = perc.num_wet()
        if new_num_wet == num_wet:
           return False
```

```
        num_wet = new_num_wet
def estimate_prob_percolating(n, q, iters=100):
    """Stima la probabilità di percolazione.
    n: numero intero delle righe e delle colonne
    q: probabilità che una cella sia permeabile
    iters: numero di arrays da provare
    ritorni: probabilità di formare un gruppo di percolazione"""
    t = [test_perc(Percolation(n, q)) for i in range(iters)]
    return np.mean(t)

def find_critical(n, q, iters=100):
    """Stima q con un cammino casuale
    returns: lista dei q
    """
    qs = [q]
    for i in range(iters):
        perc = Percolation(n, q)
        if test_perc(perc):
            q -= 0.005
        else:
            q += 0.005
        qs.append(q)
    return qs
test_perc(perc)

print()

dim=perc.num_wet()
fraction= estimate_prob_percolating(n,q,iters)

print('Frazione di percolazione ',fraction)
print()


A=np.arange(1,iters+2)
qs = find_critical(n, q,iters)
fig, ax = plt.subplots(facecolor='teal', alpha=0.1)
plt.style.use('ggplot')     #sottofondo grigio e griglia bianca
plt.title('Stima della probabilità critica in funzione del numero di passi',
color='y')
plt.tick_params(labelcolor='y')
plt.xlabel('Passi di calcolo', color='y')
plt.ylabel('Stima della prob. critica', color='y')
plt.grid(True)
plt.plot(A,qs,color='r')
plt.show()
Pc=round(np.mean(qs), 6)
print('La probabilità critica è ',Pc)
```