```
# -*- coding: utf-8 -*-
'''ici je ne mettrai pas d'accent pour eviter tout probleme selon les versions de python'''
import numpy
import random
import math


class Ising2D:
    def __init__(self,N,c):
        self.N = N
        self.Ns = N*N
        self.spin = c
        self.nbits = 8
        self.Nspin = self.Ns*self.nbits

''' la fonction temperature est redefinis dans le but de prendre en compte les autres cas du fais'''
'''que L'hamiltonien fais apparaitre la dependance du champ les cas sont multiplié par deux si '''
'''on a 0 voisin different on a une valeur pour la premiere partie de l'hamiltonien +/-2h '''
'''de meme pour le cas 1 voisin different, on a un 5eme cas le cas deux voisins et -2h on ne le prend'''
'''pas en compte ici(il figure en commentaire'''

    def temperature(self,T,h):
        beta = 1.0/T
        a = math.exp((-8.0-2*h)*beta)
        b = math.exp((-8.0+2*h)*beta)
        c = math.exp((-4.0-2*h)*beta)
        d = math.exp((-4.0+2*h)*beta)
        '''e = math.exp(-2.0*h*beta)'''
        self.p0 = a
        self.p1 = b
        self.p2 = c
        self.p3 = d
        '''self.p4 = e'''

    def voisin(self,i,j):
        ii = i
        if ii<0:
            ii=self.N-1
        elif ii>=self.N:
            ii=0
        jj = j
        if jj<0:
            jj=self.N-1
        elif jj>=self.N:
            jj=0
        return self.spin[jj][ii]


''' on adapte la fonction metropolis pour prendre en compte les cas ajoute ci-avant puisque selon'''
''' que le spin est up ou down la procedure sera la meme ici on a donc une non symetrie selon '''
'''les cas ce qui n'est pas le cas dans ising2D'''
```

```
    def metropolis(self):
        i = random.randint(0,self.N-1)
        j = random.randint(0,self.N-1)
        l=random.randint(1,8)
        s=self.spin[j][i]
        a1 = s^self.voisin(i-1,j)
        a2 = s^self.voisin(i+1,j)
        a3 = s^self.voisin(i,j-1)
        a4 = s^self.voisin(i,j+1)
        R1 = a1|a2|a3|a4
        R2 = ((a1|a2)&(a3|a4))|((a1&a2)|(a3&a4))
        for k in range(l):
                s=s>>1
        s=(s&1)
        if s==(0):
            if random.random() < (2*self.p1):
                r1 = random.getrandbits(self.nbits)
            else:
                    r1 = 0
            if random.random() < (2*self.p3):
                    r3 = random.getrandbits(self.nbits)
            else:
                        r3 = 0
            self.spin[j][i] ^= R2|(R1&r3)|r1
        else :
              if random.random() < (2*self.p0):
                        r0 = random.getrandbits(self.nbits)
              else:
                        r0 = 0
              if random.random() < (2*self.p2):
                        r2 = random.getrandbits(self.nbits)
              else:
                    r2 = 0
'''             if random.random() < (2*self.p4):
                    r4 = random.getrandbits(self.nbits)
              else:
                    r4 = 0   '''
              self.spin[j][i] ^= (R2)|(R1&r2)|r0'''on a ((R2&r4)|(R1&r2)|r0'''



    def moment(self):
        m = 0.0
        for i in range(self.N):
            for j in range(self.N):
                s = self.spin[j][i]
                for b in range(self.nbits):
                    m += 2*(s&1)-1
                    s = s >> 1
        return m*1.0/self.Nspin

    def boucle(self,n):
```

```
    m = numpy.zeros(n)
    for k in range(n):
        m[k] = self.moment()
        for i in range(self.Ns):
            self.metropolis()
    return (m,numpy.mean(m),numpy.std(m),self.spin)

def couche(self,b):
    mask = numpy.ones((self.N,self.N),dtype=numpy.uint8)*2**b
    return numpy.bitwise_and(self.spin,mask)
```