```python
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 16 12:19:00 2019

@author: Alexm
"""

from __future__ import division
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt


#Part 3: Code showing the configuration of the matrix at certain intervals of the iterations
print('Alexandra Mulholland 17336557')
print('Part 3: Snapshots of the configurations of the matrix and a plot of the convergence magnetisation against iterations of the Metropolis algorithm')
#Now creating a class which consists of the Metropolis algorithm

magf=0
J=-1
#Using 'self.' which is the instance of a class (In this case the class is Ising)
#self.moncar therefore contains the attributes of the algorithm, which in this case
#is the snapshots of the configurations
class Ising():
    def moncar(self, config, N, beta, magf):
        for i in range(N):
            for j in range(N):
                    a = np.random.randint(0, N)
                    b = np.random.randint(0, N)
                    s =  config[a, b]
                    spinmag=s*magf
                    naybor = config[(a+1)%N,b] + config[a,(b+1)%N] + config[(a-1)%N,b] + config[a,(b-1)%N]
                    EC = 2*J*s*naybor+2*spinmag
                    if EC < 0:
                        s *= -1
                    elif rand() < np.exp(-EC*beta):
                        s *= -1
                    config[a, b] = s
        return config
#Initialising a NxN matrix consisting of random spins of 1 and -1
    def simulate(self):
#simulating the Ising model
        N =20
        temp= 1.0
        iterations=[] #creating an empty list
        TS=[] #total spin of configuration
        config = 2*np.random.randint(2, size=(N,N))-1 #again, ensuring each spin is +1 or -1 within the matrix. Random allocation
        f = plt.figure(figsize=(10, 10)); #dictating what size the figures showing the configuration will be
        self.configPlot(f, config, 0, N, 1); #This is the first configuration snapshot shown in a 3 by 3 grid of configuration snapshots
#The zero here indicates the very first configuration i.e. the initial, random matrix
        msrmnt = 2001 #dictates up to which point the configuration snapshot will be shown-- will be msrmnt-1
        def TOTAL(config):
            spin = np.sum(config)
            return spin
        for i in range(msrmnt): #iteration number
            self.moncar(config, N, 1.0/temp, magf) #beta, again, is 1/Temp
            if i == 1:self.configPlot(f, config, i, N, 2); #2nd configuration snapshot, shown at 'time' 1 i.e at the first loop of the monte carlo
            if i == 10:self.configPlot(f, config, i, N, 3); #3rd configuration snapshot
            if i == 100:self.configPlot(f, config, i, N, 4); #4th
            if i == 1000:self.configPlot(f, config, i, N, 5); #5th
            if i == 2000:self.configPlot(f, config, i, N, 6); #6th
            #if i == 500:self.configPlot(f, config, i, N, 7); #7th
            #if i == 1000:self.configPlot(f, config, i, N, 8); #8th
            #if i == 2000:self.configPlot(f, config, i, N, 9); #9th
            iterations.append(i+1)
            ts1=0 #initial value-- appropriatefor random lattice
            TOT=TOTAL(config)
            ts1=ts1+TOT #summing the total spin value as the iterations continue
```

```python
            TS.append(ts1)
        plt.figure()
        plt.plot(iterations,TS, 'd', color='blue',
                markersize=6,
                markerfacecolor='c',markeredgecolor='blue',
                markeredgewidth=1)
        plt.xlabel('Iteration number');
        plt.ylabel('Total spin value');
        plt.title('Plot illustating the convergence of the magnetisation');
       #The convergence will go to |N^2|

    def configPlot(self, f, config, i, N, n): #n states which configuration is which subplot ie at i=1000, n=6
#This method plots the grid of spins, and the simulate method calls it at various times
        X, Y = np.meshgrid(range(N), range(N))
        sp =  f.add_subplot(3, 3, n)  #The first two terms in the brackets dictate the ratio of the sides x and y respectively
        plt.setp(sp.get_yticklabels(), visible=False) #making the x and y axes unlabelled
        plt.setp(sp.get_xticklabels(), visible=False)
        plt.pcolormesh(X, Y, config, cmap=plt.cm.winter); #the colour of each spin is green and blue respresenting 1 and -1
        plt.title('Iteration number = %d'%i)
    plt.show()
yo = Ising() #class = Ising
yo.simulate() #calling the function inside the class Ising
```