```
##
from random import randint
from random import random
from numpy import sqrt
from math import log
from time import time
##

#On va définir ici une classe d'objet noeud

class noeud (object):

    def __init__ (self, coordonnees, parent, estRacine = False, n = -1,
couleurRacine = BLEU):

        self.coord = coordonnees #tuple
        self.parent = parent #noeud
        self.estRacine = estRacine #bool
        self.enfants = [] #Liste de noeuds
        self.couleur = couleurRacine
        self.nbVict = 0 #int
        self.nbParties = 0 #int

        if n == -1 :
            self.n = parent.n #int, taille du plateau
        else:
            self.n = n

        if not estRacine :

            parent.enfants.append(self) #on ajoute le noeud actuel dans la
liste des enfant de son parent

            self.couleur = autreCouleur(parent.couleur) # le noeud est un
enfant, il est donc de la couleur opposée de son parent




    def cheminNoeud (self): #Donne la liste des tuples des noeuds parents
        if self.estRacine :
            return []
        chemin = self.parent.cheminNoeud()
        chemin.append(self.coord)
        return chemin


    def estFeuille(self): #définis si le noeud est feuille de l'arbre

        L = self.cheminNoeud()
        N = self.n
        if len(L) == N*N :
            return True
        return False

    def etatPlateau(self): #renvoie l'état du plateau pour le noeud
        N = self.n
        T = [[0 for _ in range(N)] for _ in range (N)]
        coupJoues = self.cheminNoeud()
```

```
        couleurJoueur = BLEU

        for coord in coupJoues :
            x = coord[0]
            y = coord[1]
            T[x][y] = couleurJoueur
            couleurJoueur = autreCouleur(couleurJoueur)

        return T

    def noeudGagnant(self): #définis si le noeud est gagnant

        T = self.etatPlateau()


        return posGagnante(T, self.couleur)




##

def creerNoeudsFils (noeudP): #Permet de créer tout les noeuds fils d'un
noeud parent donné

    T = noeudP.etatPlateau()
    CNV = listeCasesNonVides(T)
    for coord in CNV :
        noeudFilsR = noeud(coord, noeudP)

#On parcours aléatoirement l'arbre
def parcoursAleatoire(noeud):

    noeud.nbParties += 1
    #on ajoute 1 aux nombres de parties jouées du noeud
    if not noeud.estFeuille()  :

        if noeud.enfants == []:
            #Si le noeud n'a pas d'enfant, on crée ses enfants
            creerNoeudsFils(noeud)
        #on sélectionne un enfant au hasard, puis on le fait remonter par
une récursion. Si le noeud enfant renvoie false, ce qui veux dire qu'il
n'est pas gagnant, alors celui-ci est gagnant, et on lui ajoute donc une
victoire.
        if not
parcoursAleatoire(noeud.enfants[randint(0,len(noeud.enfants)-1)]) :

            noeud.nbVict += 1


    if noeud.noeudGagnant :
        #Si le noeud est gagnant, on ajoute 1 à ses victoires, puis on
renvoie true, afin de remplir la condition ci dessus
        noeud.nbVict += 1
        return True
    #Si le noeud est perdant, on n'ajoute rien, et on renvoie, false
    return False
```

```
##
#On effectue un parcours aléatoir dans l'arbre dans un temps impartit

def simulMeilleurCoupDansTemps (noeud, t):

    a = time() + t
    while a > time() :

        parcoursAleatoire(noeud)
        #On effectue autant de parcours aléatoires que le temps le permet,
afin de simuler le plus de parties possibles, et donc renvoyer un des
meilleurs coups possible
    ListeEnfants = noeud.enfants

    L = [ -1 for x in ListeEnfants]

    for k in range(0, len(ListeEnfants)):

        noeudCoup = ListeEnfants[k]

        if noeudCoup.nbParties == 0 :

            L[k] = 0

        else :

            L[k] = sqrt ( 2*log(noeudCoup.nbVict) / noeudCoup.nbParties)

    maxListe = max(L)

    for k in range (len(ListeEnfants)):
        if L[k] == maxListe :
            return ListeEnfants[k].coord


##


def creerParents (coupJouesR, coupJouesB, noeudPl):


        if coupJouesR == [] and coupJouesB == [] :
            return noeudPl
        couleur = noeudPl.couleur
        if couleur == BLEU :


            coord = coupJouesR.pop()
            noeudFils = noeud(coord, noeudPl)
            return creerParents(coupJouesR, coupJouesB, noeudFils)

        coord = coupJouesB.pop()
        noeudFils = noeud(coord, noeudPl)
        return creerParents(coupJouesR, coupJouesB, noeudFils)


##
```

```
def jouerOrdi(plateau, temps):

    coupJouesB = []
    coupJouesR = []

    for x in range(len(plateau)):

        for y in range(len(plateau)):

            if plateau[x][y] == BLEU :
                coupJouesB.append([x,y])

            if plateau[x][y] == ROUGE :
                coupJouesR.append([x,y])

    if len(coupJouesR) == len(coupJouesB) :

        couleur = BLEU

    couleur = ROUGE


    noeudActuel = creerParents(coupJouesR, coupJouesB, noeud([], "Racine",
True, len(plateau), couleur))
    prochainCoup = simulMeilleurCoupDansTemps ( noeudActuel, temps)
    plateau[prochainCoup[0]][prochainCoup[1]] = noeudActuel.couleur

    return prochainCoup
##

def simulerPartie(taille, tempsBleu, tempsRouge):

    T = platGen(taille)

    while couleurGagnante(T) == False or not estPlein(T) :

        prochainCoupBleu = jouerOrdi(T, tempsBleu)



        T[prochainCoupBleu[0]][prochainCoupBleu[1]] = BLEU

        if not estPlein(T) :

            prochainCoupRouge = jouerOrdi(T, tempsRouge)

            T[prochainCoupRouge[0]][prochainCoupRouge[1]] = ROUGE


    if couleurGagnante(T) == BLEU :
        return BLEU

    if couleurGagnante(T) == ROUGE:
        return ROUGE



def moyenneVict (taille, tempsBleu, tempsRouge, nombreParties):
```

```
parties = [0,0,0]
for _ in range (nombreParties):
    parties[simulerPartie(taille, tempsBleu, tempsRouge)] +=1
partiesortie = [parties[x] for x in range (1,3)]
return partiesortie
```