```
#-*- coding:utf-8 -*-

"""
OOMS Aurélien
102618
GROUPE 2
"""

"""Constantes déclarées en fonction des fichiers plateau."""

S_HORS_TAB = 'X'
S_VIDE = '_'
S_PIECE = '1'

from copy import deepcopy
import time
from os import system
from random import randint

class Tablier:

    def __init__(self,nom,x0,y0,n,m):
        """
        Initialise le tablier sur base du contenu d'un fichier de texte contenant les n
        lignes et m colonnes de caractères séparés par des espaces,
        chaque ligne étant suivie d'une retour à la ligne
        (la signification des caractères du fichiers est définie par les constantes
        S_HORS_TAB, S_VIDE, S_PIECE).
        """

        self.__tablier = []

        fichier = open(nom,'r')
        for ligne in fichier:                           #crée la matrice du plateau
            self.__tablier.append(ligne.split())
        fichier.close()

        self.__posSpeciale = (x0,y0)
        self.__n = n
        self.__m = m
        c=0
        for ligne in range(n):
            for colonne in range(m):
                if self.__tablier[ligne][colonne] == S_PIECE:
                    c += 1
        self.__nombrePieces = c
        self.deltaSauts = 0


        """
        Pour afficher le déroulement de la solution à la fin.
        """
        self.mouvements = []
        self.copietablier = deepcopy(self)



    def getTablier(self):
        """retourne le contenu du tablier"""
        return self.__tablier

    def estVide(self,x,y):
        """retourne True si la case de coordonnées (x,y) est vide"""
        try:
            return self.voir(x,y) == S_VIDE
        except:
            return False
    def estPiece(self,x,y):
        """retourne True si une pièce est présente sur la case (x,y)"""
        try:
            return self.voir(x,y) == S_PIECE
        except:
```

```
            return False

    def voir(self,x,y):
        """retourne le caractère représentant ce qui se trouve en position (x,y)"""
        try:
            return self.getTablier()[x][y]
        except:
            return False

    def obtenirPosSpeciale(self):
        """
        retourne les coordonnées (xd , yd ) de la position de la case vide en début de
        partie et où la dernière pièce doit aussi se trouver en fin de partie
        """
        return self.__posSpeciale

    def getN(self):
        return self.__n

    def getM(self):
        return self.__m

    def getCompteur(self):
        return self.deltaSauts

    def getPieces(self):
        return self.__nombrePieces



    def sauter(self,x1,y1,x2,y2):
        """x2,y2 est l'arrivée et x1,y1 le départ"""
        try:
            done = False
            if self.estVide(x2,y2) and self.estPiece(x1,y1) and not(x1<0 or x2<0 or y1<0 or y2<0):
#attention aux indices négatifs
                if x1 == x2 and y1 == y2+2 and self.estPiece(x2,y2+1):

                    self.getTablier()[x2][y2+2] = S_VIDE
                    self.getTablier()[x2][y2] = S_PIECE
                    self.getTablier()[x2][y2+1] = S_VIDE
                    done = True
                elif x1 == x2 and y1 == y2-2 and self.estPiece(x2,y2-1):

                    self.getTablier()[x2][y2-2] = S_VIDE
                    self.getTablier()[x2][y2] = S_PIECE
                    self.getTablier()[x2][y2-1] = S_VIDE
                    done = True
                elif x1 == x2-2 and y1 == y2 and self.estPiece(x2-1,y2):

                    self.getTablier()[x2-2][y2] = S_VIDE
                    self.getTablier()[x2][y2] = S_PIECE
                    self.getTablier()[x2-1][y2] = S_VIDE
                    done = True
                elif x1 == x2+2 and y1 == y2 and self.estPiece(x2+1,y2):

                    self.getTablier()[x2+2][y2] = S_VIDE
                    self.getTablier()[x2][y2] = S_PIECE
                    self.getTablier()[x2+1][y2] = S_VIDE
                    done = True
            return done
        except:

            return False

    def antiSauter(self,x1,y1,x2,y2):
        """x1,y1 est l'arrivée et x2,y2 le départ"""
        try:
            done = False
            if self.estPiece(x2,y2) and self.estVide(x1,y1) and not(x1<0 or x2<0 or y1<0 or y2<0):
                if x1 == x2 and y1 == y2+2 and self.estVide(x2,y2+1):
                    self.getTablier()[x2][y2+2] = S_PIECE
```

```
                    self.getTablier()[x2][y2] = S_VIDE
                    self.getTablier()[x2][y2+1] = S_PIECE
                    done = True
                elif x1 == x2 and y1 == y2-2 and self.estVide(x2,y2-1):
                    self.getTablier()[x2][y2-2] = S_PIECE
                    self.getTablier()[x2][y2] = S_VIDE
                    self.getTablier()[x2][y2-1] = S_PIECE
                    done = True
                elif x1 == x2-2 and y1 == y2 and self.estVide(x2-1,y2):
                    self.getTablier()[x2-2][y2] = S_PIECE
                    self.getTablier()[x2][y2] = S_VIDE
                    self.getTablier()[x2-1][y2] = S_PIECE
                    done = True
                elif x1 == x2+2 and y1 == y2 and self.estVide(x2+1,y2):
                    self.getTablier()[x2+2][y2] = S_PIECE
                    self.getTablier()[x2][y2] = S_VIDE
                    self.getTablier()[x2+1][y2] = S_PIECE
                    done = True
            return done
        except:

            return False

    def jouer(self):
        """
        Fonction s'appelant récursivement de la manière suivante:
            1 Essaye un déplacement:
                - Si il est possible, il l'effectue et rejoue sur le tablier modifié. (Revient au 1
avec le nouveau tablier résultant du déplacement)
                    1 ...
                        - ...
                            1 ...
                                etc
                            2 ...
                        - ...
                    2 Déconstruit la solution
                - Si il n'est pas possible, il essaye un autre déplacement.

        Notez qu'une fois qu'il a essayé tous les déplacements suivant un premier déplacement sans en
trouver un qui convient,
        il déconstruit la solution qu'il avait commencé à créer en revenant à l'étape précédent le
déplacement qui nous mettait dans une situation insolvable.
        Une fois qu'il a trouvé la solution il s'interrompt immédiatement, sans déconstruire la
solution en cours et affiche les étapes de la résolution du problème.
        """

        liste_vides = []
        for ligne in range(self.getN()):
            for colonne in range(self.getM()):
                if self.estVide(ligne,colonne):
                    liste_vides.append((ligne,colonne))

        if self.getPieces() == self.getCompteur()+1 and self.estPiece(self.obtenirPosSpeciale()
[0],self.obtenirPosSpeciale()[1]):   #solution trouvée
            return True

        for vide in liste_vides:
            for k in [(0,2),(2,0),(-2,0),(0,-2)]:
                if self.sauter(vide[0]+k[0],vide[1]+k[1],vide[0],vide[1]):
                    self.deltaSauts += 1


                    if self.jouer():
                        self.mouvements.append((vide[0]+k[0],vide[1]+k[1],vide[0],vide[1])) #PUSH des
déplacements corrects pour la résolution


                        if len(self.mouvements) == self.getPieces()-1: #une fois qu'on a mit tous les
déplacements sur la pile, on les applique successivement
                            affichage = 3
                            while affichage not in ['1','2']:
                                system("clear")
```

```
                                print self.copietablier
                                affichage = raw_input("\nSOLUTION TROUVEE:\n1 --> Affichage dynamique;
\n2 --> Affichage coup par coup.\n")
                            if affichage == '1':
                                system("clear")
                                while self.mouvements != []: # tant qu'il y a un déplacement à
effectuer
                                    print self.copietablier
                                    time.sleep(0.5)
                                    jmp = self.mouvements.pop() #POP des déplacements corrects
                                    self.copietablier.sauter(jmp[0],jmp[1],jmp[2],jmp[3])
                                    system("clear")
                                print self.copietablier
                            else:
                                system("clear")
                                raw_input("APPUYEZ SUR ENTRER POUR AFFICHER SUCCESSIVEMENT CHAQUE
ETAPE")
                                system("clear")
                                while self.mouvements != []: # tant qu'il y a un déplacement à
effectuer
                                    print self.copietablier
                                    raw_input("")
                                    jmp = self.mouvements.pop() #POP des déplacements corrects
                                    self.copietablier.sauter(jmp[0],jmp[1],jmp[2],jmp[3])
                                    print
                                print self.copietablier


                        return True


                    self.deltaSauts -= 1
                    self.antiSauter(vide[0]+k[0],vide[1]+k[1],vide[0],vide[1])



        return False


    def __repr__(self):
        """Renvoie une représentation du tableau."""

        tableau = ''

        for ligne in self.getTablier():
            for e in ligne:
                char = '1'
                if e == S_HORS_TAB:
                    char = 'X'
                elif e == S_VIDE:
                    char = '_'
                tableau+= char+'  '
            tableau = tableau[:len(tableau)]
            tableau += '\n'
        tableau = tableau[:len(tableau)-1]

        return tableau

"""
MAIN
"""
system("clear")

raw_input("Ce programme vous permet de trouver une solution possible à un tablier du jeu du solitaire.
\n\nPour créer un tablier, il suffit de l'enregistrer \
sous la forme d'une matrice n*m dans un fichier texte.\nLes pièces sont représentées par des '1', les
vides par des '_' et les cases hors tablier par des 'X'.\nTous ses \
caractères sont séparés par des espaces et après chaque ligne il est nécessaire d'utiliser un retour à
la ligne.\n\nAPPUYEZ SUR ENTRER")
system("clear")
choix = raw_input("Entrez le chemin/nom de votre fichier ou appuyez simplement sur ENTRER pour voir un
exemple de tablier.\nSi vous n'avez pas encore créé de\
```

```
 tablier vous pouvez entrer 'Quitter', créer votre fichier et relancer le programme.\n\n")
if choix == "Quitter":
    system("clear")

elif choix == "":
    system("clear")
    exemple = 'CDE'[randint(0,2)]
    tablierExemple = Tablier(exemple,3,3,7,7)
    print tablierExemple
    raw_input("\nAPPUYEZ SUR ENTRER POUR LANCER LA RESOLUTION")
    system("clear")
    print tablierExemple
    print "\nRecherche d'une solution en cours..."
    tablierExemple.jouer()
    raw_input("\nAPPUYEZ SUR ENTRER POUR QUITTER")
    system("clear")

else:
    tablier_cree = False
    while not tablier_cree:

        try:
            test = open(choix,'rw')
            test.close()
            system("clear")
            x0 = raw_input("Ligne de la position spéciale : ")
            y0 = raw_input("Colonne de la position spéciale : ")
            l = raw_input("Nombre de lignes : ")
            c = raw_input("Nombre de colonnes : ")
            plateau = Tablier(choix,int(x0),int(y0),int(l),int(c))
            tablier_cree = True

        except IOError:
            raw_input("ERREUR : Le nom du fichier est incorrect.")
            system("clear")
            choix = raw_input("Entrez un nouveau chemin/nom pour le fichier : ")

        except:
            raw_input ("ERREUR : Les paramètres entrés ne sont pas corrects.")
    system("clear")
    print plateau
    print "\n Recherche d'une solution en cours..."
    if not plateau.jouer():
        system("clear")
        print plateau
        print "\nIl n'y a pas de solution pour ce tablier."
    raw_input("\nAPPUYEZ SUR ENTRER POUR QUITTER")
    system("clear")
```