```
#99236 Ines Pissarra

def eh_tabuleiro(tab):
    # universal -> booleano
    '''A funcao recebe um tabuleiro, tab.
    Devolve True se o argumento corresponder a um tabuleiro, \
False caso contrario.'''
    if type(tab)!=tuple or len(tab) != 3:
        return False
    for i in range(3):
        if type(tab[i])!=tuple or len(tab[i]) != 3:
            return False
        for j in range(3):
            if type(tab[i][j])!=int or tab[i][j] not in (-1,0,1):
                return False
    return True


def eh_posicao(p):
    # universal -> booleano
    '''A funcao recebe uma posicao p. \nDevolve True se o argumento \
corresponder a uma posicao, False caso contrario.
As posicoes sao enumeradas da esquerda para a direita e de cima para baixo.'''
    return (type(p)==int and p in (1,2,3,4,5,6,7,8,9))


def obter_coluna(tab, c):
    # tabuleiro X inteiro -> vector
    '''A funcao recebe um tabuleiro, tab, e um inteiro, c (1, 2 ou 3), \
e devolve a coluna correspondente a esse inteiro.
As colunas sao enumeradas da esquerda para a direita.'''
    if eh_tabuleiro(tab) and type(c)==int and c in (1,2,3):
        coluna = ()
        for i in range (3):
            coluna = coluna + (tab[i][c-1],)
        return coluna
    raise ValueError('obter_coluna: algum dos argumentos e invalido')


def obter_linha(tab, l):
    # tabuleiro X inteiro -> vector
    '''A funcao recebe um tabuleiro, tab, e um inteiro, l (1, 2 ou 3), \
e devolve a linha correspondente a esse inteiro.
As linhas sao enumeradas de cima para baixo.'''
    if eh_tabuleiro(tab) and type(l)==int and l in (1,2,3):
        return tab[l-1]
    raise ValueError('obter_linha: algum dos argumentos e invalido')


def obter_diagonal(tab, d):
    # tabuleiro X inteiro -> vector
    '''A funcao recebe um tabuleiro, tab, e um inteiro, d (1 ou 2), \
e devolve a diagonal correspondente a esse inteiro.
Diagonal 1 = posicoes 1,5,9 ; Diagonal 2 = posicoes 7,5,3.'''
    if eh_tabuleiro(tab) and type(d)==int and d in (1,2):
        if d == 1:
            diagonal = (tab[0][0],) + (tab[1][1],) + (tab[2][2],)
        else:
            diagonal = (tab[2][0],) + (tab[1][1],) + (tab[0][2],)
        return diagonal
    raise ValueError('obter_diagonal: algum dos argumentos e invalido')


def tabuleiro_str(tab):
    # tabuleiro -> cad. caracteres
    '''Esta funcao recebe um tabuleiro, tab, em formato de tuplo \
e devolve-o com o formato tradicional.'''
    if eh_tabuleiro(tab):
        tab_str = ''
```

```
        for i in range(len(tab)):
            if i != 0:
                tab_str = tab_str + '\n-----------\n'
                # desta forma apenas ira ter '-' a partir da segunda vez
                # que passa no ciclo 'for', e nao ira repeti-la no final
            for j in range(len(tab[i])):
                if j != 0:
                    tab_str = tab_str + '|'
                    # desta forma apenas ira ter '|' a partir da segunda vez
                    # (de cada linha) que passa no ciclo 'for'
                    # e nao ira repeti-la no final de cada linha
                if tab[i][j] == (-1):
                    tab_str = tab_str + ' O '
                elif tab[i][j] == (0):
                    tab_str = tab_str + '   '
                elif tab[i][j] == (1):
                    tab_str = tab_str + ' X '
        return tab_str
    raise ValueError('tabuleiro_str: o argumento e invalido')


def eh_posicao_livre(tab, p):
    # tabuleiro X posicao -> booleano
    '''Esta funcao recebe um tabuleiro, tab, e uma posicao, p.
    Verifica se a posicao no tabuleiro esta livre, devolvendo True caso esteja \
e False caso contrario.'''
    if eh_tabuleiro(tab) and eh_posicao(p):
        # primeiro indice da posicao p na tabela:
        l = (p - 1)//3
        # 0,1,2//3 = 0  3,4,5//3 = 1  6,7,8//3 = 2
        #segundo indice da posicao p na tabela:
        c = (p - 1) % 3
        # 0,3,6 % 3 = 0  1,4,7 % 3 = 1  2,5,8 % 3 = 2
        return tab[l][c] == 0
    raise ValueError('eh_posicao_livre: algum dos argumentos e invalido')


def obter_posicoes_livres(tab):
    # tabuleiro -> vector
    '''Esta funcao recebe um tabuleiro, tab, e devolve as posicoes do tabuleiro\
 que ainda estao livres.'''
    if eh_tabuleiro(tab):
        livres = ()
        for p in range(1,10):
            if eh_posicao_livre(tab, p):
                livres = livres + (p,)
        return livres
    raise ValueError('obter_posicoes_livres: o argumento e invalido')


def jogador_ganhador(tab):
    # tabuleiro -> inteiro
    '''Esta funcao recebe um tabuleiro, tab.
    Indica qual o jogador \
que ganhou o jogo (-1, 0, 1 para 'O', empate, 'X' respetivamente).'''
    if eh_tabuleiro(tab):
        for i in range(1,4):
            coluna = obter_coluna(tab, i)
            if coluna[0] == coluna[1] == coluna[2] != 0:
                return coluna[0]
            linha = obter_linha(tab, i)
            if linha[0] == linha[1] == linha[2] != 0:
                return linha[0]
        for d in range(1,3):
            diagonal = obter_diagonal(tab, d)
            if diagonal[0] == diagonal[1] == diagonal[2] != 0:
                return diagonal[0]
        return 0
    raise ValueError('jogador_ganhador: o argumento e invalido')
```

```
def marcar_posicao(tab, j, p):
    # tabuleiro X inteiro X posicao -> tabuleiro
    '''A funcao recebe um tabuleiro, tab, um inteiro correspondente ao jogador \
(-1 se 'O', 1 se 'X'), e uma posicao, p.
    Marca a nova jogada, devolvendo o tabuleiro atualizado'''
    if eh_tabuleiro(tab) and type(j)==int and j in (-1,1) and eh_posicao(p)\
       and eh_posicao_livre(tab, p):
        l = (p - 1)//3
        c = (p - 1) % 3 #mesmo raciocinio da funcao eh_posicao_livre
        tab = tab[0:l] + (tab[l][0:c] + (j,) + tab[l][c+1:],) + tab[l + 1:]
        # tab = linhas anteriores a linha de p + valores da linha de p
        # ate a posicao de p + simbolo do jogador
        # + restantes valores da linha de p + restantes linhas
        return tab
    raise ValueError('marcar_posicao: algum dos argumentos e invalido')


def escolher_posicao_manual(tab):
    # tabuleiro -> posicao
    '''A funcao recebe um tabuleiro, tab, e realiza a leitura da posicao \
introduzida manualmente pelo jogador, p.
Retorna essa mesma posicao caso esta seja valida'''
    if not eh_tabuleiro(tab):
        raise ValueError('escolher_posicao_manual: o argumento e invalido')
    p = eval(input('Turno do jogador. Escolha uma posicao livre: '))
    if not eh_posicao(p) or not eh_posicao_livre(tab, p):
        raise ValueError('escolher_posicao_manual: a posicao introduzida \
e invalida')
    return p


def escolher_posicao_auto(tab, j, e):
    # tabuleiro X inteiro X cad. carateres -> posicao
    '''A funcao recebe um tabuleiro, tab, um inteiro correspondente ao jogador \
(-1 se 'O', 1 se 'X'), \n e uma estrategia ('basico', 'normal' ou 'perfeito')
 Devolve a posicao onde o computador ira jogar quando for a sua vez.'''
    if not eh_tabuleiro(tab) or type(j)!=int or j not in (-1,1):
        raise ValueError('escolher_posicao_auto: \
algum dos argumentos e invalido')
    livres = obter_posicoes_livres(tab)
    canto = (1, 3, 7, 9)
    lat = (2, 4, 6, 8)

    if e == 'basico':
        if 5 in livres: #centro vazio
            return 5
        for p in canto: #canto vazio
            if p in livres:
                return p
        for p in lat: #lateral vazio
            if p in livres:
                return p

    elif e == 'normal':
        # vitoria - ir testando as posicoes livres se encontrar alguma que
        # faca dele ganhador entao escolhe essa posicao
        for p in livres:
            if jogador_ganhador(marcar_posicao(tab, j, p)) == j:
                return p
        # bloqueio - ir testando as posicoes livres se encontrar alguma
        # que faca do seu adversario ganhador entao escolhe essa posicao
        for p in livres:
            if jogador_ganhador(marcar_posicao(tab, -j, p)) == -j:
                return p
        if 5 in livres: #centro vazio
            return 5
        for p in canto: #canto oposto
```

```
            # descobrir qual o primeiro e segundo indice da posicao p na tabela:
            l, c = (p - 1)//3, (p - 1) % 3
            # se p estiver livre e o adversario estiver no canto oposto
            # entao escolher p. tab[2-l][2-c] = canto oposto
            if p in livres and tab[2-l][2-c] == -j:
                return p
        for p in canto: #canto vazio
            if p in livres:
                return p
        for p in lat: #lateral vazio
            if p in livres:
                return p

    elif e == 'perfeito':
        for p in livres: #vitoria
            if jogador_ganhador(marcar_posicao(tab, j, p)) == j:
                return p
        for p in livres: #bloqueio
            if jogador_ganhador(marcar_posicao(tab, -j, p)) == -j:
                return p
        for p in livres: #bifurcacao
            i = bifurcacao(tab, j, p)
            if i: # se i != 0 entao escolher i
                return i
        i = bloq_bifurcacao(tab, j, livres) #bloquear bifurcacao
        if i:
            return i
        if 5 in livres: #centro vazio
            return 5
        for p in canto: #canto oposto
            l, c = (p - 1)//3, (p - 1) % 3
            if p in livres and tab[2-l][2-c] == -j:
                return p
        for p in canto: #canto vazio
            if p in livres:
                return p
        for p in lat: #lateral vazio
            if p in livres:
                return p
    raise ValueError('escolher_posicao_auto: algum dos argumentos e invalido')


def bifurcacao(tab, j, p):
    # tabuleiro X inteiro X inteiro -> inteiro
    '''A funcao recebe um tabuleiro, tab, o inteiro correspondente ao jogador\
(-1 se 'O', 1 se 'X') e uma posicao livre, p.
    Devolve p caso a jogada nessa posicao crie uma bifurcacao, \
caso contrario devolve 0.'''
    l1, c1 = (p - 1)//3 + 1, (p - 1) % 3 + 1 #
    coluna = obter_coluna(tab, c1)
    linha = obter_linha(tab, l1)
    if l1 == c1: # diagonal 1
        diagonal = obter_diagonal(tab, 1)
    elif l1 == (4 - c1) : # diagonal 2 (posicoes com linha-coluna 3-1;2-2;1-3)
        diagonal = obter_diagonal(tab, 2)
    else: # caso nao esteja contido numa diagonal
        diagonal = ()
    # se as duas linhas/colunas/diagonais que se intersetam em p contem
    # uma peca do jogador (cada uma), e nao contem pecas do adversario que
    # bloqueiem a bifurcacao entao escolhe-se p, criando uma bifurcacao
    if j in linha and -j not in linha:
        if j in coluna and -j not in coluna:
            return p
        elif j in diagonal and -j not in diagonal:
            return p
    elif j in coluna and -j not in coluna:
        if j in diagonal and -j not in diagonal:
            return p
    return 0
```

```
def bloq_bifurcacao(tab, j, livres):
    # tabuleiro X inteiro X vector -> inteiro
    '''A funcao recebe um tabuleiro, tab, o inteiro correspondente ao jogador\
(-1 se 'O', 1 se 'X') \n e o vector que contem as posicoes livres, livres.
 Devolve uma posicao caso a jogada nessa posicao bloqueie uma bifurcacao, \
caso contrario devolve 0.'''
    bif_a = ()
    for p in livres:
        i = bifurcacao(tab, -j, p)  # bifurcacao do adversario
        if i:
            # bif_a sao as possiveis biforcacoes do adversario
            bif_a = bif_a + (i,)
    if len(bif_a) == 1:   #se so houver uma bifurcacao entao bloqueia-a
        return bif_a[0]
    elif len(bif_a) > 1:
        # p sera a sua jogada e p2 a jogada que o seu adversario tera
        # de jogar para o bloquear
        for p in livres:
            tab2 = marcar_posicao(tab, j, p)
            # testar o que acontecera depois desta jogada
            for p2 in livres:
                # este if descarta as opcoes em que p2 cria uma biforcacao
                # (p2 not in bif_a), e as opcoes em que p nao criou um dois
                # em linha (porque para isso teria de haver um p2 tal que se
                # o adversario ecolher essa posicao bloqueia a vitoria)
                if p2 != p and p2 not in bif_a and \
                    jogador_ganhador(marcar_posicao(tab2, j, p2)) == j:
                    return p
    return 0


def jogo_do_galo(jogador, e):
    # cad. carateres X cad. carateres -> cad. carateres
    '''Esta funcao recebe um jogador 'X' ou 'O' (a escolha do utilizador)
e a estrategia a ser usada pelo computador ('basico', 'normal' ou 'perfeito')
Esta e a funcao principal que permite o utilizador jogar com o computador.'''
    if jogador not in ('O','X') or e not in ('basico', 'normal', 'perfeito'):
        raise ValueError('jogo_do_galo: algum dos argumentos e invalido')
    print('Bem-vindo ao JOGO DO GALO.\nO jogador joga com \'' + jogador + '\'.')
    tab = ((0,0,0),(0,0,0),(0,0,0))
    if jogador == 'X':
        j = 1
        p = escolher_posicao_manual(tab) # TURNO DO JOGADOR
        tab = marcar_posicao(tab, j, p)
        print(tabuleiro_str(tab))
    elif jogador == 'O':
        j = -1
    while len(obter_posicoes_livres(tab)) != 0 and jogador_ganhador(tab) == 0:
        # enquanto ainda houver posicoes livres e nao houver jogador ganhador
        # repete-se o ciclo 'while'
        print ('Turno do computador (' + e +'):') # TURNO DO COMPUTADOR
        p = escolher_posicao_auto(tab, -j, e)
        tab = marcar_posicao(tab, -j, p)
        print(tabuleiro_str(tab))
        #confirmar se ainda nao existe vencedor:
        if len(obter_posicoes_livres(tab)) != 0 and jogador_ganhador(tab) == 0:
            p = escolher_posicao_manual(tab) # TURNO DO JOGADOR
            tab = marcar_posicao(tab, j, p)
            print(tabuleiro_str(tab))
    #depois de acabar o ciclo while, ou houve um empate, ou algum jogador ganhou
    if jogador_ganhador(tab) == 0 and len(obter_posicoes_livres(tab)) == 0:
            return('EMPATE')
    elif jogador_ganhador(tab) == -1:
            return('O')
    elif jogador_ganhador(tab) == 1:
            return('X')
```