DS 3 d'informatique - MPSI - Polyominos - Correction

Question 1 – On obtient :

```
P_fig7 = [[True , True , False, False, True , True ],
          [True , True , True , True , True , True ],
          [False, True , False, False, True , False]]
```

Question 2 –

```
def afficher(P):
    for i in range(len(P)):
        s = ""
        for j in range(len(P[i])):
            if P[i][j]:
                s = s + "#"
            else:
                s = s + "-"
        print(s)
```

Question 3 –

```
def test1(P):
    if len(P) == 0:
        return False
    n1 = len(P[0])
    for i in range(len(P)):
        if len(P[i]) != n1:
            return False
    return True
```

Question 4.a –

```
def cptTrue(P):
    cpt = 0
    for i in range(len(P)):
        for j in range(len(P[i])):
            if P[i][j]:
                cpt += 1
    return cpt
```

Question 4.b –

```
def test2(P, n):
    return cptTrue(P) == n
```

Question 5.a –

```
def ligne(P, i0):
    L = []
    for j in range(len(P[i0])):
        if P[i0][j]:
            L.append((i0,j))
    return L
```

Question 5.b – On applique la méthode vue en cours permettant de transformer une fonction itérative en fonction récursive.

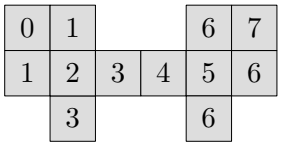
```
# Suppose que test1(P) vaut True
def colonne_aux(P, j0, L, i):
    if i >= len(P):
        return L
    if P[i][j0]:
        L.append((i,j0))
    return colonne_aux(P, j0, L, i+1)
```

```
def colonne(P, j0):
    return colonne_aux(P, j0, [], 0)
```

Question 5.c –

```
# Suppose que test1(P) vaut True
def test3(P):
    if len(ligne(P, 0)) == 0 or len(ligne(P, len(P)-1)) == 0:
        return False
    if len(colonne(P, 0)) == 0 or len(colonne(P, len(P[0])-1)) == 0:
        return False
    return True
```

Question 6 – On obtient :



Question 7 – On obtient :

```
dist_fig5 = [[(0, 0)],
             [(1, 0), (0, 1)],
             [(1, 1)],
             [(2, 1), (1, 2)],
             [(1, 3)],
             [(1, 4)],
             [(2, 4), (0, 4), (1, 5)],
             [(0, 5)],
             []]
```

Question 8.a –

```
def dansI(P, i, j):
    if i < 0 or i >= len(P):
        return False
    if j < 0 or j >= len(P[0]):
        return False
    if not P[i][j]:
        return False
    return True
```

Question 8.b –

```
def parcLarg(P, i0, j0):
    vu = {(i0,j0): None}
    dist = [[(i0,j0)]]
    while len(dist[-1]) > 0:
        dist.append([])
        for (i1,j1) in dist[-2]:
            for (i2,j2) in [(i1+1,j1), (i1-1,j1), (i1,j1+1), (i1,j1-1)]:
                if dansI(P, i2, j2) and not (i2,j2) in vu:
                    vu[(i2,j2)] = None
                    dist[-1].append((i2,j2))
    return vu, dist
```

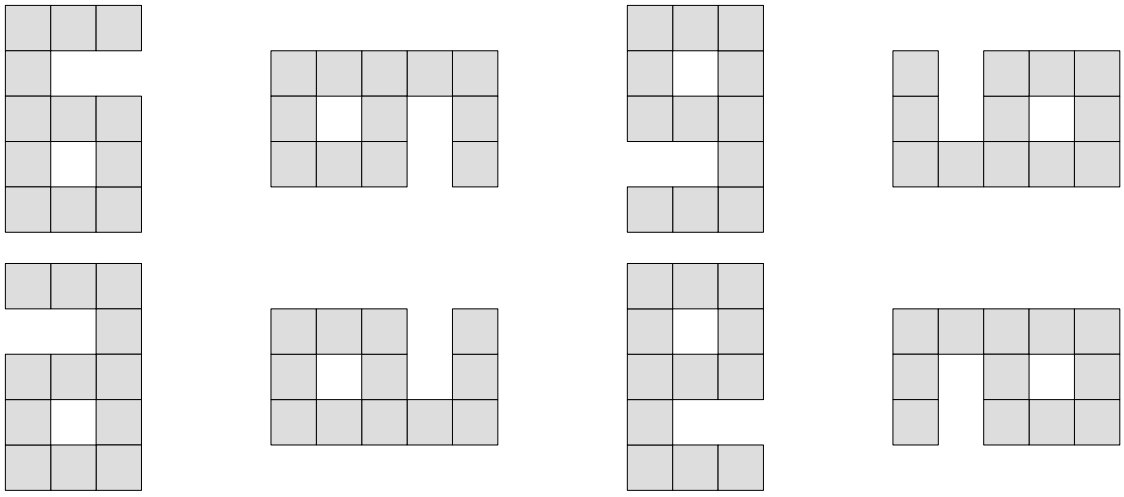
Question 8.c –

```
# Remarque: la liste C est non vide car P vérifie (C1) et (C3)
def test4(P):
    C = colonne(P, 0)
    (i0,j0) = C[0]
    vu, _ = parcLarg(P, i0, j0)
    return len(vu) == cptTrue(P)
```

Question 9 –

```
def convertir(P):
    return [(i,j) for i in range(len(P)) for j in range(len(P[i])) if P[i][j]]
```

Question 10 – On obtient 8 polyominos :



Question 11 –

```
def egal(P1, P2):
    if len(P1) != len(P2):
        return False
    if len(P1[0]) != len(P2[0]):
        return False
    for i in range(len(P1)):
        for j in range(len(P1[i])):
            if P1[i][j] != P2[i][j]:
                return False
    return True
```

Question 12 –

```
def sym(P):
    m1 = len(P)
    m2 = len(P[0])
    Q = [[None for _ in range(m2)] for _ in range(m1)]
    for i in range(m1):
        for j in range(m2):
            Q[i][m2-j-1] = P[i][j]
    return Q
```

Question 13 –

```
def rot(P):
    m1 = len(P)
    m2 = len(P[0])
    Q = [[None for _ in range(m1)] for _ in range(m2)]
    for i in range(m1):
        for j in range(m2):
            Q[j][m1-i-1] = P[i][j]
    return Q
```

Question 14 –

```
def appartient(P, L):
    for Q in L:
        if egal(P, Q):
            return True
    return False
```

```
def makeT(P):
    R = [P]
    S = [sym(P)]
    for _ in range(3):
        R.append(rot(R[-1]))
        S.append(rot(S[-1]))
    T = []
    for P in R + S:
        if not appartient(P, T):
            T.append(P)
    return T
```

Question 15 –

```
# Indique si la position (i,j) est libre dans G
def estLibre(i,j,G):
    m1 = len(G)
    m2 = len(G[0])
    if i < 0 or i >= m1:
        return False
    if j < 0 or j >= m2:
        return False
    if G[i][j] != -1:
        return False
    return True
```

```
def placer(G, Q, m, i, j):
    G1 = [[e for e in g] for g in G]
    for (i1,j1) in Q:
        i2 = i + i1
        j2 = j + j1
        if estLibre(i2,j2,G1):
            G1[i2][j2] = m
        else:
            return None
    return G1
```

Question 16 –

```
# Renvoie la somme des tailles des polyominos
def sommeTailles(P_list):
    s = 0
    for P in P_list:
        s += cptTrue(P)
    return s
```

```
def katamino_aux(G1, TQ_list, m):
    if m == len(TQ_list):
        return G1
    m1 = len(G1)
    m2 = len(G1[0])
    for Q in TQ_list[m]:
        for i in range(m1):
            for j in range(m2):
                G2 = placer(G1, Q, m, i, j)
                if G2 is not None:
                    G3 = katamino_aux(G2, TQ_list, m+1)
                    if G3 is not None:
                        return G3
    return None
```

```
# Renvoie None si pas de solution
def katamino(m1, m2, P_list):
    if sommeTailles(P_list) != m1 * m2:
        return None
    G = [[-1 for j in range(m2)] for i in range(m1)]
    TP_list = [makeT(P) for P in P_list]
    TQ_list = [[convertir(P) for P in L] for L in TP_list]
    return katamino_aux(G, TQ_list, 0)
```

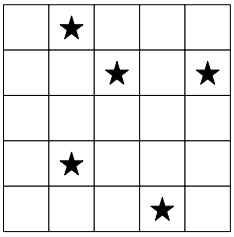
Question 17 –

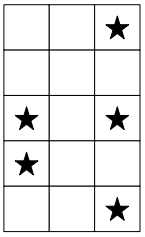
```
# Hypothèse: m > 0
def makeE(n, m):
    E1 = [[i] for i in range(n)]
    for _ in range(m-1):
        E2 = []
        for R in E1:
            for i in range(R[-1]+1, n*n):
                E2.append(R + [i])
        E1 = E2
    return E1
```

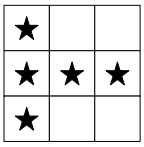
Question 18.a – Pour générer tous les n-ominos :

→ On génère toutes les listes $L \in E_{n,n}$ avec la fonction de la question précédente.

→ On construit G_L la grille obtenue à partir de $\alpha(L)$ en supprimant les dernières lignes et les dernières colonnes ne contenant pas d'étoile. Par exemple les grilles associées à `[1,7,9,16,23]`, `[2,10,12,15,22]`, `[0,5,6,7,10]` sont respectivement :







La grille G_L est représentée par une variable « `P: list[list[bool]]` » où les étoiles correspondent aux cases contenant `True`.

→ On conserve uniquement les listes `P` qui sont des n-omnios.

Question 18.b – L'entier `n1` (resp. `n2`) est le nombre de lignes (resp. colonnes) dans la grille après avoir supprimé les dernières colonnes (resp. lignes) ne contenant pas d'étoile.

```
# Suppose L non vide
def find_n1(L):
    return L[-1]//len(L) + 1
```

```
def find_n2(L):
    n = len(L)
    n2 = 0
    for k in L:
        j = k % n
        if j >= n2:
            n2 = j
    return n2 + 1
```

```
def L_to_P(L):
    n = len(L)
    n1 = find_n1(L)
    n2 = find_n2(L)
    P = [[False for j in range(n2)] for i in range(n1)]
    for k in L:
        P[k // n][k % n] = True
    return P
```

```
def cptOminos(n):
    cpt = 0
    E = makeE(n,n)
    for L in E:
        P = L_to_P(L)
        if test1(P) and test2(P, n) and test3(P) and test4(P):
            cpt += 1
    return cpt
```

Question 19 –

```
def dansGrille(i,j,n):
    if i < 0 or i >= n:
        return False
    if j < 0 or j >= n:
        return False
    return True
```

```
# Cette fonction indique si sélectionner la case (i,j) est autorisé.
# n est la taille de la grille.
# m est le numéro de l'étape.
# j0 est tel que (0,j0) est la case sélectionnée par l'algorithme lors de la
# première étape.
# jmin est la plus petite colonne sur laquelle au moins une case est
# sélectionnée.
def choixPossible(i,j,n,m,j0,jmin):
    if i == 0 and j < j0:
        return False
    if jmin >= n-m and j >= n-m:
        return False
    return True
```

```
# etat[i][j] \in {0,1,2} indique l'état de la case (i,j).
# L est la liste des cases dans l'état 1 dont le numéro est supérieur au numéro
# de la dernière case sélectionnée.
def generer(etat, m, L, j0, jmin):
    n = len(etat)
    if n == m:
        return 1
    res = 0
    for k in range(len(L)):
        (i1,j1) = L[k]
        if choixPossible(i1,j1,n,m,j0,jmin):
            L1 = L[k+1:]
            L2 = [(i1+1,j1), (i1-1,j1), (i1,j1+1), (i1,j1-1)]
            L2 = [(i2,j2) for (i2,j2) in L2
                  if dansGrille(i2,j2,n) and etat[i2][j2] == 0]
            etat[i1][j1] = 2
            for (i2,j2) in L2:
                etat[i2][j2] = 1
            res += generer(etat, m+1, L1 + L2, j0, min(jmin, j1))
            for (i2,j2) in L2:
                etat[i2][j2] = 0
            etat[i1][j1] = 1
    return res
```

```
def cptOminosBis(n):
    etat = [[0 for j in range(n)] for j in range(n)]
    res = 0
    for j in range(n):
        etat[0][j] = 1
        res += generer(etat, 0, [(0,j)], j, j)
        etat[0][j] = 0
    return res
```