Pentaminos

JeanDuSud – Problème 1

Nous avons résolu la majorité des questions posées dans ce problème. Nous avons utilisé l'algorithmique en grande partie ce qui nous donne la certitude quand au nombre de pentaminos et quant à leurs différences (pas de doublons). La résolution graphique de la question 2 permet une visualisation rapide du pavage et des erreurs potentielles (non trouvées). Questions restantes : 3.b 3.c et 4 en partie.

```
cester(a,b):
brotated=rotate(b)
fa,fb,fc,fd=pluspetit(brotated)
brotated=formatarray(brotated,fa,fb,fc,fd)

#On commence en partant du principe que les matrices sont identiques
#Si tests=="pareil" alors c'est que ce sont les memes pintaminos
tests="pareil"

#print ("\nRéduction aux plus petits rectangles...")
#reduction au rectangle le plus petit pour les deux figures...

#-----------------------------------
#pour a
#print ("Matrice 1")
fa,fb,fc,fd=pluspetit(a)
a=formatarray(a,fa,fb,fc,fd)
#pour b
#print ("Matrice 2")
fa,fb,fc,fd=pluspetit(b)
b=formatarray(b,fa,fb,fc,fd)
if len(a)!=len(b) and len(a)!=len(b[0]):
   tests="diff"
   #print ("Ces deux matrices ne sont pas les memes !")
elif len(a[0])!=len(b[0]) and len(a[0])!=len(b):
   tests="diff"
```

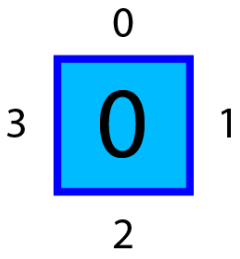
Exercice 1 – Lister les pentaminos, utilisation d'un algorithme Python

Les pentaminos sont des formes mathématiques formées de 5 carrés dont le côté mesure 1 unité. On remarquera qu'un pentamino possède toujours une aire égale à 5 unités. De plus, un pentamino est un polygone, c'est à dire qu'il est formé d'un seul et même morceau. Il est donc possible de générer un pentamino à l'aide d'un code informatique.

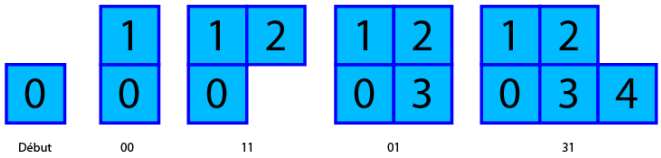
Principe de génération :
Afin de lister tout les pentaminos, on part d'un nombre, qui servira de *graine* dans la génération, en listant toutes les *graines* on liste tout les pentaminos.
Pour passer de la *graine* au pentamino, j'ai inventé un petit système dont voici le fonctionnement :
Imaginez que l'on créée un carré que l'on nommera 0.
La graine de départ est donnée comme ceci :

00110131

Comment l'utilise-t-on ? C'est simple, celle-ci est lisible par groupes de deux chiffres, le premier groupe est 00, le premier chiffre de ce groupe définit le numéro du carré, soit le carré 0. Le second chiffre definit la position du prochain carré comme ceci :

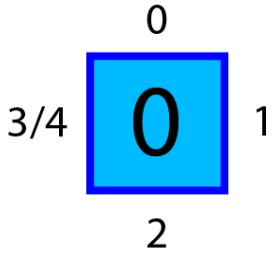


Le nouveau carré ainsi créé sera nommé 1. Ensuite, on lit le groupe de chiffres numéro deux : 11. Celui-ci indique qu'il faut prendre le carré 1 et en ajouter un, nommé 2, à droite. En lisant le code entier, on obtient ceci :



Nous venons en effet de générer le pentamino P.

Vous vous posez surement la question : mais que ce passe t-il si on utilise la graine *95487616* ?
C'est très simple, cette graine n'existera jamais car lorsque nous listerons les graines, nous compterons en base 4, c'est à dire que nous partirons de *00000000* et arriverons à *44444444*. La rotation 4 n'existe pas mais ce n'est pas grave car nous modifierons le schéma pour obtenir ceci :

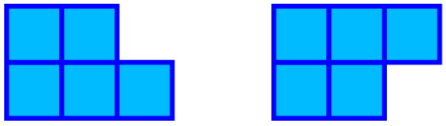


Mais tous les problèmes ne sont pas résolus, il en

reste un : si j'écris *44003322* ? En effet il y a un problème, lorsqu'on va lire le premier groupe, le carré numéro 4 n'existera pas encore. On aura alors ce que j'appelle une *fausse graine*. Celles-ci formerons une erreur sur python et seront ignorées.

Les doublons :

Reprenons notre pentamino P. Notre *graine* est celle-ci : *00110131*. Mais il en existe beaucoup d'autres ! Par exemple : *00111231* , ou encore : *01102321* Cette dernière est particulière car elle créée ce pentamino (à gauche la graine 1 et à droite la graine 2) :



Ces deux pentaminos sont les même, symétriquement. Pour éviter de se retrouver avec des dizaines de pentaminos égaux, il faut créer un algorithme anti-doublons. En effet avec l'algorithme ci-dessus on obtient une liste de 5 187 pentaminos différents…

Python est un outil génial pour l'utilisation des matrices. C'est pourquoi elles sont le fondement de notre algorithme. Voici ce à quoi ressemble un pentamino sortit de l'algorithme générateur :

00000
00100
00110
00110
00000

C'est une matrice de format 5x5. Pourquoi 5x5 ? Car le plus grand pentamino possible mesure 5 de hauteur, tout simplement. Le programme python adapte automatiquement la matrice en fonction de la forme, avant de la positionner dans un format adapté :

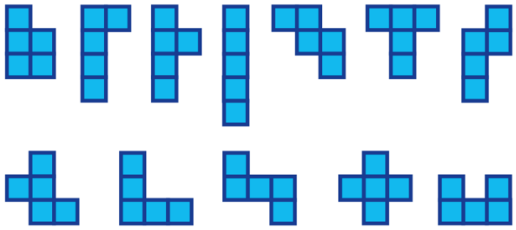
10
11
11

L'algorithme anti-doublons utilise ces matrices et les compare, on teste ainsi les symétries horizontales, verticales, diagonales, centrales, centrale-horizontales…
Il est maintenant impossible d'obtenir deux fois le même pentamino, et il est donc temps de lancer le code qui fait déja… 332 lignes (Disponible en annexe).

Le résultat :
Enfin le résultat que nous attendons tous : Combien existe t-il de pentaminos et lesquels sont-ils ? On ajoute quelques lignes python pour inscrire les résultats dans un fichier texte. On lance le code, et après 2 secondes s'affiche le resultat suivant :

```
10
11
12
Fichier texte créé, 12 matrices obtenues.
>>>
```

Douze, c'est le nombre total de pentaminos existants. Un petit tour sur internet suffira pour verifier l'information, et pour décevoir la personne qui vient de travailler longtemps pour obtenir cette jolie image :



On nommera les pentaminos par des numéros, dans l'ordre de l'image. Le programme utilisé est fourni en annexe.

Fonctionnement simplifié :

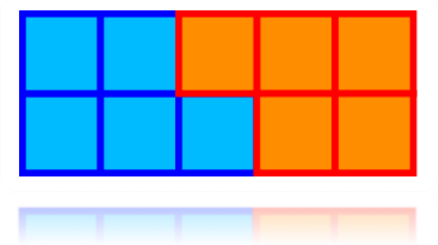
Voici une explication rapide du fonctionnement du code générateur.

Début
Pour chaque graine possible
  Générer le pentamino
  Si pas d'erreur de génération
    Comparer pentamino aux autres déjà existants (sym. et rot.)
    Si non existant
      Enregistrer pentamino dans fichier texte
Fin Pour
Fin

En théorie il suffit de modifier un nombre très faible de variables pour générer le bon nombre de tetraminos, hexaminos…

Exercice 2 – Pavage des pentaminos sur un plan 2D

Le pavage d'un plan réel, c'est quoi ? Vous avez surement déjà observé ce genre de choses sur l'image ci-contre. Un motif se repète sur le sol. La question est peut-on faire ceci avec les pentaminos ? Pour le P, c'est simple :



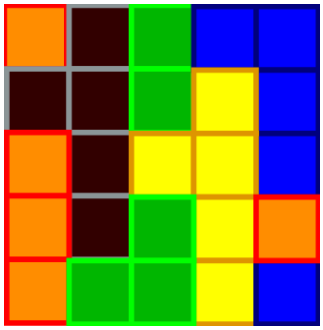
On peut aisément répéter cette forme à l'infini sur un plan.

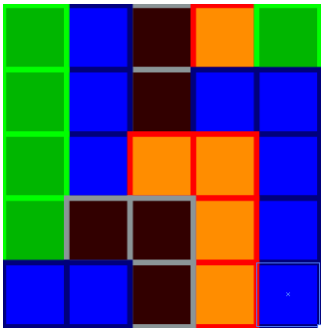
Mapping
Malheureusement, les autres pentaminos ne sont pas comme cela et il faut trouver un autre moyen de démontrer leur capacité à paver le sol. Nous allons utiliser une méthode très connue dans le jeu vidéo, le mapping de texture. Pour ceux qui ne connaissent pas c'est un moyen d'utiliser une image pour couvrir une surface en répétant l'image. Pour cela l'image doit être un peu spéciale, voici un exemple :

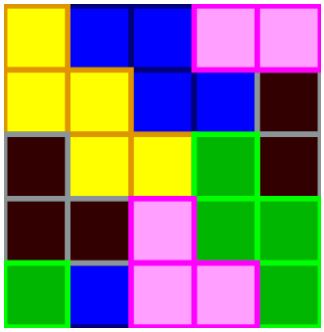
Si vous repétez cette image vous remarquerez que les côtés droit et gauche correspondent, de même pour les côtés haut et bas :

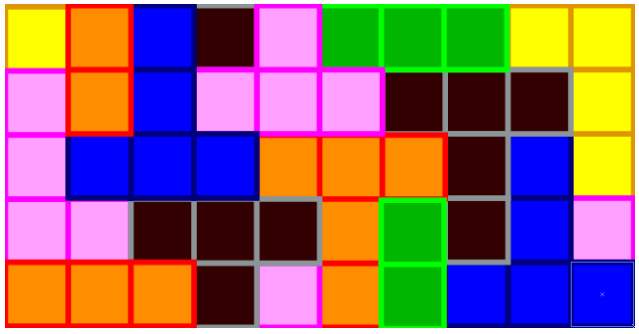
On peut utiliser cette méthode pour prouver qu'un pavage est possible à l'infini. Il suffit de trouver le bon rognage qui se repète. Nous remarquerons tout à l'heure que chaque pentamino possède son pavage.

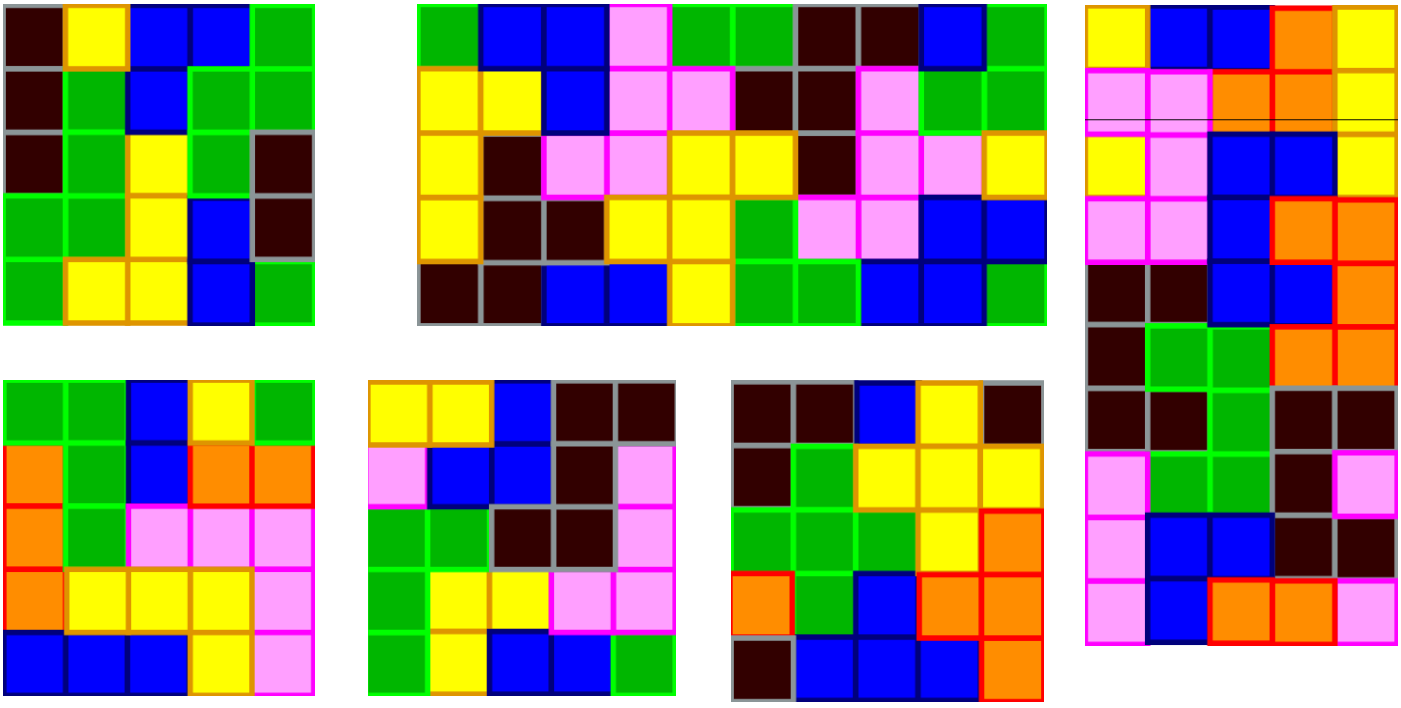
Les autres pentaminos :
P étant validé, il nous reste les autres. Je ne vais pas, cette fois-ci, m'attarder en bavardage. Voici les motifs correspondants, vous pouvez les tester dans un logiciel comme Blender pour les tester (ou les mettre bout à bout manuellement)







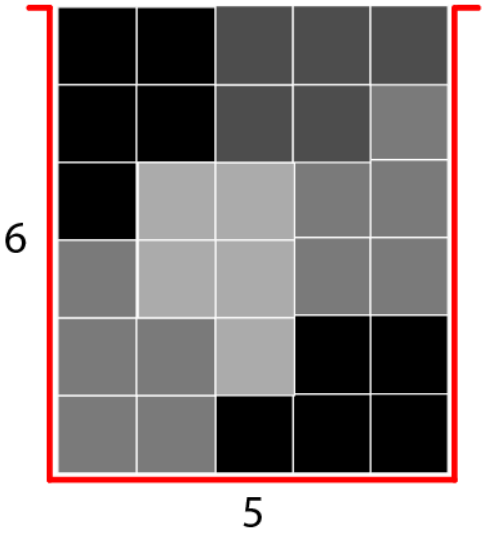




On a ainsi listé tout les pentaminos et leurs positionnements sur le plan. Grâce à cette méthode, les pentaminos sont inscrits dans un rectangle, ce qui prouve que le pavage peut être répété à l'infini.

Exercice 3 – Rectangles de base 5 avec le pentamino P

Partie A :
Imaginez que l'on possède un bac (en 2 dimensions), Ce bac possède une base de 5 unités mais la hauteur est inconnue, on la note *h*.
Quelle sont les hauteurs *h* possibles pour remplir entièrement le bac ? Par exemple, dans la figure ci-dessous, le bac est plein et *h=6*.

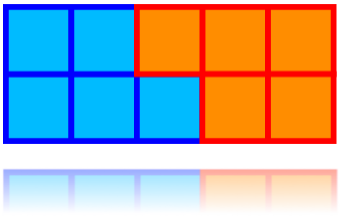


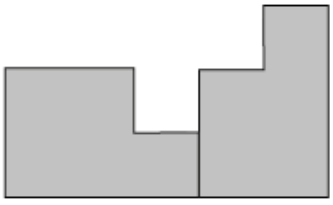
On remarque que *Aire=h*5* mais aussi que *Aire=n*5* avec n nombre de pentaminos P dans le bac (Un pentamino fait 5 unités d'aire).

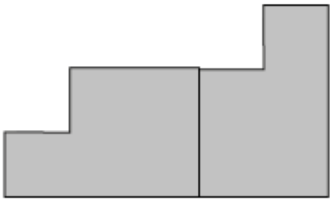
Or on cherche à donner toute les valeurs de *h*, on sait déjà que l'on peut créer un bac de hauteur *h=2* (voir figure au début de l'exercice 2). On peut donc affirmer qu'il existe *h* tel que :

$$h = 2k \quad \forall k \in \mathbb{N}^*$$

Or il suffit maintenant d'éliminer les autres possibilités soit les h impairs. Dans ce type de bac possédant une base de 5 unités, on prouve qu'il existe forcement – pour h impair – un pentamino vertical. Le seul moyen de ne pas avoir de pentaminos vertical est d'utiliser ce montage :

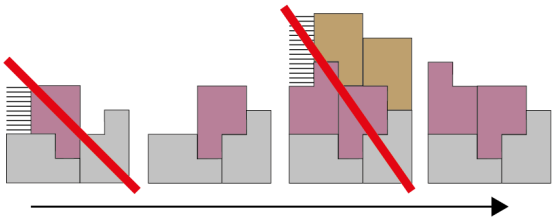




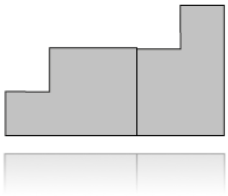


On peut donc démarrer du premier pentaminos vertical, ce qui laisse deux possibilités de démarrages présentes ci-contre (et leurs symétrie verticale). C'est aussi les deux seules possibilités de fin. Mais on remarque que le premier assemblage ne peux être continué que de cette façon (page suivante)

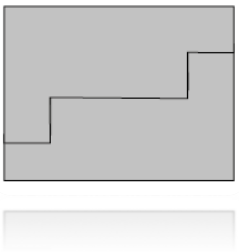
En effet, on retombe sur le même schéma que le second modèle page précedente :



On sait maintenant que le pentamino commence forcement par ce schéma, composé d'un nombre pair de pentaminos.



Logiquement, ceci est vrai pour remplir le bac, donc celui-ci commence et se termine par ce schéma (plus les symétries).



Si on ferme directement l'ouverture laissée entre le haut en le bas du bac, on obtient h paire. Il faut démontrer que l'on ne peux pas remplir l'espace laissé entre le haut et le bas par un nombre impair de pentaminos. Comme on le voit sur le graphique ci-dessous, peu de choix sont disponibles lors de l'ajout de nouveau pentaminos, on voit également :
- soit nous retombons sur le schéma précedent,
- soit nous terminons le bac avec un nombre pair de pentaminos.

Étant donné que nous retombons toujours sur l'un des schémas ci-dessous, on peut dire que *h* est forcément paire.

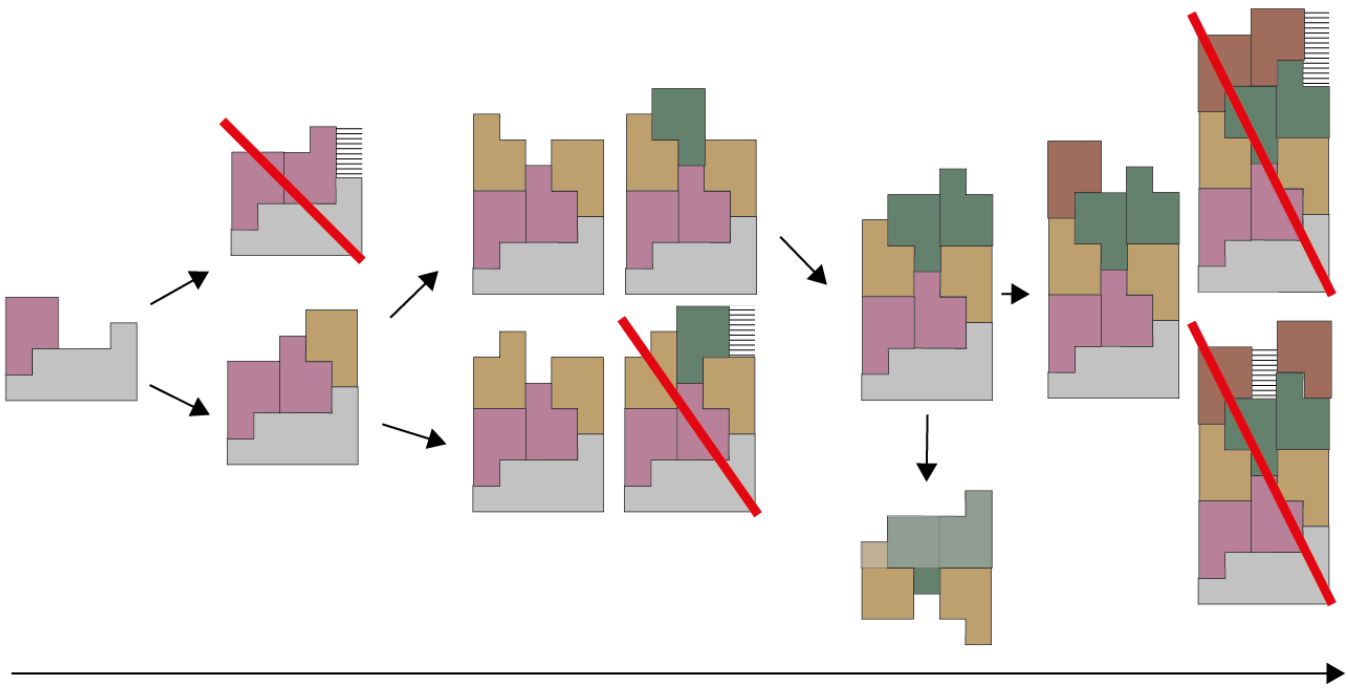
Partie B :
On a donc m*n=5*k où k est le nombre de pentaminos, donc 5 divise 5*k qui implique que 5 divise le nombre m*n et comme 5 est un nombre premier, (si p premier divise a*b, alors p divise a ou bien p divise b) il divise par exemple m. on a alors m=5*j .
Si n est pair on sait que c'est possible.
Notons (n,m) les couples possibles.
Si n est impair :
Des cas comme sont (5,10) possible mais d'autres non comme (5,15)

Partie C :
L'exercice demande également quel est le nombre maximum de pentaminos positionnables sur un rectangle *m x n* sans superposition (il y aura donc des trous si *m x n* ne sont pas multiples de 5). Nous n'avons pas de résultat, mais nous pouvons donner une certitude.

Le nombre *nb* de pentaminos de dépassera pas :

$$nb \leq int\left(\frac{m*n}{5}\right)$$

De plus, si m<2 ou n<2 *nb*=0.



Exercice 4 – Les autres pentaminos

Le problème précédent peut être appliqué aux autres pentaminos tel que le pentamino I. Pour un bac de format $m*n$ on trouve

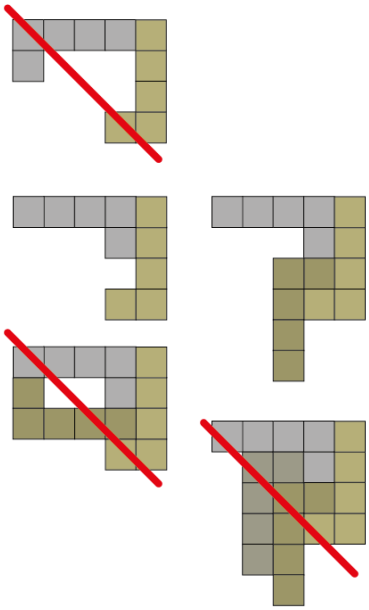
$$m = k$$
$$n = 5 \times l$$
$$\forall k, l \in \mathbb{N}^*$$

Pour un rectangle de format *m x n* le nombre *nb* de pentaminos I est donné par l'égalité suivante :

$$nb = int\left(\frac{m * n}{5}\right)$$

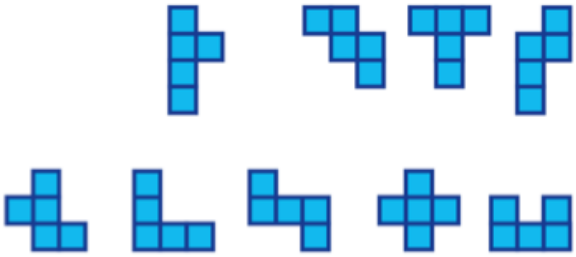
Pour le pentamino en forme de L, on peut former un rectangle de 2 * 5 et donc remplir le bac précédent avec une hauteur *h* paire.

Il est impossible de remplir un bac différement car par nécessité on arrive à cela :



Il faut maintenant étudier les 9 autres pentaminos.

Les cas simples :
Les autres pentaminos sont directement éliminés pour les premières parties du problème (création d'un rectangle *m x n* sans superposition ni trous). C'est le cas de tous ces pentaminos :



Pour ces pentaminos, il n'est pas possible de paver un rectangle quelconque car aucun de ces pentaminos hormis :

Ne peut remplir un carré de côté 2*2 cases dans le coin d'un rectangle. Pour cette dernière pièce, on ne peut continuer qu'avec une autre en position verticale et dans un seul sens ce qui créé un espace vide.

Seconde partie du problème :

Pour ces derniers pentaminos, il faut résoudre une autre partie du problème qui est le remplissage maximum d'un carré *m x n* avec espaces. Ce résultat n'a pas été obtenu. On ne dépassera pas :

$$nb \leq int\left(\frac{m * n}{5}\right)$$

Annexe 1 – programme générateur de pentaminos et anti-doublons

```
def rotate(array) :
    c=[[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]]
    larg=5
    haut=5
    i=0
    while i<haut:
        j=0
        while j<larg:
            c[i][j]=array[larg-j-1][i]
            j=j+1
        i=i+1
    #print (c,"b")
    return c

def chaine(array):
    larg=len(array[0])
    haut=len(array)
    c=""
    i=0
    while i<haut:
        j=0
        while j<larg:
            c=c+str(array[i][j])
            j=j+1
        i=i+1
    return c

def colonne(array,col):
    c=[0,0,0,0,0]
    i=0
    while i<5 :
        c[i]=array[i][col]
        i=i+1
    return c

def symetrievert(array,h,l):
    c=""
    larg=len(array[0])
    haut=len(array)
    i=0
    while i<haut:
        j=0
        while j<larg:
            val=array[i][larg-1-j]
            c=c+str(val)
            j=j+1
        i=i+1
    return c

def symetriedia(array):
    c=""
    larg=len(array[0])-1
    haut=len(array)-1
    i=0
    while i<haut:
        j=0
        while j<larg:
            c=c+str(array[j][i])
            j=j+1
        i=i+1
    array=c
    return array

def symetriecentre(array):
    larg=len(array[0])
    haut=len(array)
    c=""
    i=0
    while i<haut:
        j=0
        while j<larg:
            c=c+str(array[haut-1-j][larg-1-j])
            j=j+1
        i=i+1
    return c


def formatarray(a,fa,fb,fc,fd):
    larg=fd-fc+1
    haut=fb-fa+1
    #print ("Format plus petit rectangle : ",haut,"x" , larg)
    c=[]
    i=0
    while i<haut:
        j=0
        c.append([])
        while j<larg:
            c[i].append(a[fa+i][fc+j])
            j=j+1
        i=i+1
    return (c)

def pluspetit(a) :
    fa=0#ligne ou commence le futur tableau
    i=0
    while i<5 :
        if sum(a[i])==0 :
            i=i
        else :
            fa=i
            i=5 #stop le while
        i=i+1

    fb=0#ligne ou finit le futur tableau
    i=4
    while i>-1 :
        if sum(a[i])==0 :
            i=i
        else :
            fb=i
            i=-1 #stop le while
        i=i-1
    fc=0#colonne ou commence le futur tableau
    i=0
    while i<5 :
        if sum(colonne(a,i))==0 :
            i=i
        else :
            fc=i
            i=5 #stop le while
        i=i+1

    fd=0#colonne ou finit le futur tableau
    i=4
    while i>-1 :
        if sum(colonne(a,i))==0 :
            i=i
        else :
            fd=i
            i=-1 #stop le while
        i=i-1
    return fa,fb,fc,fd

def tester(a,b):
    brotated=rotate(b)
    fa,fb,fc,fd=pluspetit(brotated)
    brotated=formatarray(brotated,fa,fb,fc,fd)

    #On commence en disant que les matrices sont identiques
    #Si tests=="pareil" alors c'est ce sont les memes pintaminos
    tests="pareil"

    #print ("\nRéduction aux plus petits rectangles...")
    #reduction au rectangle le plus petit pour les deux figures...

    #-------------------------------------
    #pour a
    #print ("Matrice 1")
    fa,fb,fc,fd=pluspetit(a)
    a=formatarray(a,fa,fb,fc,fd)
    #pour b
    #print ("Matrice 2")
    fa,fb,fc,fd=pluspetit(b)
    b=formatarray(b,fa,fb,fc,fd)
    if len(a)!=len(b) and len(a)!=len(b[0]):
        tests="diff"
        #print ("Ces deux matrices ne sont pas les memes !")
    elif len(a[0])!=len(b[0]) and len(a[0])!=len(b):
        tests="diff"
        #print ("Test format rectangle : Ces deux matrices ne sont pas
les memes !")
    else :
        none=0
        #print ("Test format rectangle : les deux formats sont éguaux
!")

    #print (a)
    #print (b)
    #-------------------------------------
```

```
    #print ("\nTests de symétries...")
    #ici tests de symétries

    #-------------------------------------
    if tests!="diff":
        symetrieactual=0
        #0 si aucune simétrie ne correspond
        if a==b or a==brotated :
            symetrieactual=1
        #Symétrie horizontale
        if a==b[::-1] or a==brotated[::-1]:
            symetrieactual=1
        if chaine(a)==symetrievert(b,len(a),len(a[0])) or
chaine(a)==symetrievert(brotated,len(a),len(a[0])) :
            symetrieactual=1
        if chaine(a)==symetrievert(b[::-1],len(a),len(a[0])) or
chaine(a)==symetrievert(brotated[::-1],len(a),len(a[0])) :
            symetrieactual=1
        if len(a)==len(a[0]) and len(b)==len(b[0]) :
            if chaine(a)==symetriedia(b) or
chaine(a)==symetriedia(brotated) :
                symetrieactual=1
            if chaine(a)==symetriedia(b[::-1]) or
chaine(a)==symetriedia(brotated[::-1]) :
                symetrieactual=1
        if chaine(a)==symetriecentre(b) or
chaine(a)==symetriecentre(brotated) :
            symetrieactual=1

        if symetrieactual==1 :
            tests="pareil"
            #print ("Test symétrie : Ces deux matrices sont les memes !")
        else :
            none=0
            #print ("Test symétrie : les deux matrices ne sont pas
symétriques !")
    else:
        symetrieactual=1
    #-------------------------------------


    if symetrieactual!=1 :
        tests="diff"

    if tests=="pareil" :
        return "pareil"
        #print ("\n\nRésultat : Ces deux matrices sont les memes !")
    else :
        return "diff"
        #print ("\n\nRésultat : Ces deux matrices sont différentes !")

def base10toN(num,n):
    return ((num == 0) and  "0" ) or ( base10toN(num // n, n).strip("0")
+ "0123456789abcdefghijklmnopqrstuvwxyz"[:n][num % n])


def afficher(matrice) :
    hauteur=len(matrice)
    largeur=len(matrice[0])
    i=0
    tout=''
    while i<hauteur:
        tout=tout+"\n"
        j=0
        while j<largeur:
            tout=tout+str(str(matrice[i][j]))
            j=j+1
        i=i+1
    return tout

def seeder(matrice,long,seed) :
    matrice[2][2]=1
    long[0][0]=2
    long[0][1]=2
    erreur=""
    i=1
    while i<len(long):
        todo=seed[(i-1)*2]+seed[(i-1)*2+1]
        if long[int(seed[(i-1)*2])][0]!="" :
            a=long[int(seed[(i-1)*2])][0]
            b=long[int(seed[(i-1)*2])][1]
            originepos=[a,b]
            if seed[(i-1)*2+1]=='0': #0 = haut
                if a-1<0 :
                    a=1
                if matrice[a-1][b]==0:
                    matrice[a-1][b]=1
                    long[i][0]=a-1
                    long[i][1]=b
                else:
                    erreur="Il y a eu une erreur."
            if seed[(i-1)*2+1]=='1': #1 = droite
                if b+1>3 :
                    b=3
                if matrice[a][b+1]==0:
                    matrice[a][b+1]=1
                    long[i][0]=a
                    long[i][1]=b+1
                else:
                    erreur="Il y a eu une erreur."
            if seed[(i-1)*2+1]=='2': #2 = bas
                if a+1>3 :
                    a=3
                if matrice[a+1][b]==0:
                    matrice[a+1][b]=1
                    long[i][0]=a+1
                    long[i][1]=b
                else:
                    erreur="Il y a eu une erreur."
            if seed[(i-1)*2+1]=='3' or seed[(i-1)*2+1]=='4': #3 ou 4 =
gauche
                if b-1<0 :
                    b=1
                if matrice[a][b-1]==0:
                    matrice[a][b-1]=1
                    long[i][0]=a
                    long[i][1]=b-1
                else:
                    erreur="Il y a eu une erreur."
            i=i+1
        else :
            i=i+1
            erreur="Il y a eu une erreur."
    return erreur,matrice


#65536 possibilité
seed="000000"

compte=0
tout=''
toutmatrice=[]

i=0
while i<65536 :
    seed=base10toN(i,4)
    seed=(8-len(str(seed)))*'0'+str(seed)
    #print ('\nseed:'+seed)
    matrice=[[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]

    long=[["","",""],["","",""],["","",""],["","",""],["","",""]]

    erreur,matrice=seeder(matrice,long,seed)
    if erreur=="" :
        none=0
        nonmarque="nonmarque"
        w=0
        while w<compte:
            autre=toutmatrice[w]
            #autre sera les matrices précédemment inscrites dans la
variable tout
            if tester(matrice,autre)=='pareil':
                nonmarque="dejala"
            w=w+1
        if nonmarque=="nonmarque" :
            toutmatrice.append(matrice)
            fa,fb,fc,fd=pluspetit(matrice)
            matricereformat=formatarray(matrice,fa,fb,fc,fd)
            tout=tout+'\n'+afficher(matricereformat)
            print (compte)
            compte=compte+1
        #print(afficher(matrice))
        #print (i)
    else :
        #print('Erreur pour le seed :'+seed)
        none=0

    i=i+1
print (compte)

fichier = open("C:\matrices.txt", "w")
fichier.write(tout)
fichier.close()
```